A 'delayed layering' three layer channel routing

K.E. Chang
W.S. Feng

Indexing terms: Graph theory and topology, Algorithms

Abstract: In the paper, a new three-layer channel router with 'delayed layering' technique is presented. The delayed layering scheme in the routing can improve the capability of the router to reach comprehensive objectives. This new router not only minimises the tracks used, but also minimises the via usage and maximises the use of preferred routing layers. The delayed layering router consists of two steps: track assignment and layer assignment. The track assignment uses a topological sorting algorithm to determine the horizontal track number of every net. A layerless layout will result from the track assignment. The layer assignment heuristically determines which layers can be used for routing the wire segments in the layerless layout, such that the vias generated are as small as possible. All examples presented in Reference 5, including Deutsch's difficult example, have been used to evaluate our router. The experiments showed that the solution quality with respect to via usage was better than the previous solutions and the number of tracks used in channel was satisfactory.

1 Introduction

The channel routing problem is an important problem in VLSI/LSI layout design automation. In a channel routing problem, terminal locations at the top and bottom of a rectangle are specified, and netlists specifying the interconnection of terminals are given. A number from 0 to n is assigned to each terminal. A net i is a collection of vertical and horizontal wire segments that electrically connect a set of terminals with the same number i , while those terminals with number 0 are unconnected. The channel routing will connect the pins of nets on parallel sides of a rectangle channel. The goals of channel routing are to minimise the channel width and the via usage needed in the routing.

A netlist description is defined by the top and bottom lists $T = (T_1, T_2, \dots, T_n)$ and $B = (B_1, B_2, \dots, B_n)$. T_i (or B_i) is the net number of a terminal at the top (or bottom) in the i th column. An example of a netlist description is shown in Fig. 1. The channel width, which is the number of horizontal tracks used, is bounded by a parameter called maximum channel density, denoted as d_{max} , that refers to the maximum number of nets crossing over any column. A via is a feed-through hole or a contact where

two wire segments of a net on different layers are connected. A crossing is a point where two wire segments of different nets intersect each other. Two cases of the crossing point are shown in Fig. 2. They are two wire segments that intersect or that have portions which overlap each other.

Fig. 1 Netlist description

Fig. 2

a Intersection of two wire segments
b Overlap of two wire segments

There are two kinds of channel routing. One is restrictive routing, and the other is doglegging routing. The restrictive routing aims for a dogleg-free solution where the number of horizontal tracks on which any net can be positioned is limited to one. In this case any net is routed as a single horizontal segment with vertical branches connecting it to the terminals. The aim of restrictive routing is to determine a horizontal track number for every net. Another kind of routing is doglegging routing. The idea is to remove the constraint of restrictive routing on wire geometry, i.e. to allow net splitting between different tracks. The splitting vertical wire between different horizontal tracks is called dogleg. The introduction of doglegs may decrease the horizontal tracks needed in a channel. However, there are certain penalties to pay for this gain. Each dogleg adds one or two additional vias and this decreases the circuit performance from the electrical standpoint.

Two-layer channel routers were presented in References 1, 2, 3 and 4. They only aimed for the dogleg solutions with minimum tracks used and did not concern the vias minimisation in the routings. In the most current technologies, three layers are available for routing. Four modes, VHV, HVH, overlap, and knock-knee modes, are

Paper 7135E (C2, C6), first received 17th October 1988 and in revised form 25th July 1989

K.E. Chang and W.S. Feng are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

assumed in the three-layer restrictive channel routing algorithms [5-8]. These algorithms produced a large number of vias, although they could obtain optimal solutions for channel width. The algorithm presented in Reference 9 was allowed the doglegs in the routing. Note that the number of tracks needed in a channel for the solution of doglegging routing can be reduced. However, the doglegging routing generates more vias in its solution. From a technological point of view, the number of vias could be kept small. In IC processing, an increase of vias will cause a decrease of yield, which will affect the circuit performance significantly because of the resistance associated with vias. In addition, the increase in vias will cause an increase in routing space such that the manufacturing cost will be increased. Hence we consider the restrictive routing in this paper, and the via minimisation will also be considered in the routing algorithms.

Previous routers were only considered the track locations of nets when the layering model was fixed, and such that the number of vias introduced in the routing were not important. Delayed layering in our channel routing is an efficient way to minimise via introductions and maximise the use of a preferred routing layer under the routability of each net guaranteed. Our proposed algorithm is divided into two steps: track assignment and layer assignment. The track assignment determines the horizontal track number of each net. A solution called transient layout will result from the track assignment. The transient layout is an incomplete or layerless physical layout in which the topology of nets in the layout has become clear, whereas the wire segments in the layout have not yet been assigned to layers. The layer assignment is to determine which layers can be used for routing the wire segments in the transient layout so that the vias generated are as small as possible. For the routing problem given in Fig. 1, the layout shown in Fig. 3a is a transient layout in which the horizontal track number of every net is determined. Fig. 3b is a complete physical layout with one via only.

Fig. 3 Layout
a Transient
b Complete

2 Graph models of the channel routing problem

It must be guaranteed that no two wire segments running in different directions are run on the same layer. Both graphs, (vertical constraint graph (VCG) and horizontal constraint graph (HCG)) are used to avoid the vertical and horizontal conflicts among the nets, respectively.

Let L_i and R_i denote a column number of the leftmost and the rightmost terminals for a net i , respectively. The x -interval of a net i is defined by (L_i, R_i) , and the x -length of a net is the number of columns in its x -interval. Both nets are in horizontal conflict if their x -intervals overlap. There are two cases for non-conflict in the x -intervals of two nets i and j , as indicated in Fig. 4. The HCG can be

Fig. 4 Non-conflict on x -intervals of two nets

constructed as follows. Each vertex in the HCG represents a net. An edge is connected at vertices i and j , denoted by (i, j) , if $R_i < L_j$ or $R_j < L_i$, while both nets i and j are denoted by (L_i, R_i) and (L_j, R_j) , respectively. The graph in Fig. 5 is a HCG of netlist as given in Fig. 1. The edge (i, j) in HCG indicates that two x -intervals of net i and net j are not overlapping each other. Hence, two nets can be put into the same track if and only if their corresponding vertices are connected in HCG.

Fig. 5 Horizontal constraint graph

It is easy to see that any two nets must not overlap in a vertical column. A binary relation will be induced from the netlist. This relation, first defined in Reference 1, can be represented by a directed graph VCG where each vertex corresponds a net and a directed edge is connected from vertex i to vertex j , denoted by $i \rightarrow j$ or $\langle i, j \rangle$, if $T_k = i$ and $B_k = j$, $1 \leq k \leq n$. The VCG of the netlist in Fig. 1 is demonstrated in Fig. 6. The number of vertices in the longest path of VCG is referred to as v_{max} . The definitions and properties associated with VCG are listed below.

Fig. 6 Vertical constraint graph

Definition 1: Vertex i in VCG is a predecessor of vertex j , if and only if there is a directed path from vertex i to vertex j .

Property: The properties of VCG (denoted as G) are:

1. The precedence relation in G is transitive. In other words, j is a predecessor of m , if and only if $j \rightarrow k$ and $k \rightarrow m$.

2. The precedence relation in G is irreflexive. In other words, a directed path from vertex k to vertex j is not in G if and only if there is a directed path from vertex j to vertex k in G .

Definition 2: A precedence relation which is both transitive and irreflexive is a partial order or topological order.

The order information of nets will be derived from VCG if we route every net from the top track to the bottom track in the channel. The net i will be routed preceding the net j if there is a directed path from vertex i to vertex j in VCG. According to definition 2, the order sequence of nets in VCG is a topological order. The horizontal track number of each net can be decided based on this topological order. However, there are many orders determined by the VCG and each order will produce a possible solution. Therefore, a straightforward way of track assignment is to use a topological sorting to enumerate all orders in the VCG and to find one optimal solution amongst them.

3 Topological order of the track assignment

Note that any two nets can be put in the same track if and only if their corresponding vertices are connected in HCG. Each maximal clique (or complete subgraph) in HCG forms a group of nets which can be routed in the same track. HCG is used to determine how nets can be broken into some groups. To avoid the vertical conflicting in the routing, only nets with corresponding vertices that have no predecessors in VCG should form a subgraph of HCG and be partitioned into some groups by finding the maximal cliques in the subgraph. Nets in each group are order independent because they have no vertical conflict in VCG. Hence, the group is defined as an independent class of nets. Because at most two layers are needed for routing the horizontal wire segments and the remaining layer is preserved for vertical wire segments, two independent classes can be produced simultaneously at each time. The algorithm to determine the topological order proceeds by selecting out two independent classes with vertices in VCG that have no predecessors. Then, vertices in the two independent classes together with all edges leading out from them are deleted from VCG. These two steps are repeated until all vertices have been listed. Let V_f be a set of vertices which have no predecessors in VCG. This algorithm can be written as following.

ALGORITHM #1

1. input VCG. Let n be the number of vertices
 2. **while** ($|V_f| \neq 0$)
 3. partition the vertices in V_f into independent classes E by finding all maximal cliques in a subgraph of HCG with corresponding vertices in V_f ;
 4. output two classes A_i and A_{i+1} from E ;
 5. delete all vertices in A_i and A_{i+1} and all edges leading out of them from VCG;
 6. V_f = a set of vertices which have no predecessors in the reduced VCG.
 7. **end**
- END ALGORITHM #1

The solution for track assignment is to find a topological order of independent classes such that the number of classes in the sequence is minimised. The number of classes is proportional to the number of tracks used in a

channel. Based on the sequence, the track assigned to nets in each class can be determined. Let $A_1, A_2, \dots$, and A_m be a topological order of independent classes. Both classes A_{2i-1} and A_{2i} are generated at i -th iteration in the algorithm and nets in classes A_{2i-1} and A_{2i} are routed in the track i at two horizontal layers, respectively. It is clearly shown that only one independent class may be generated at the iteration when all vertices in V_f form only one maximal clique in HCG. If the set of vertices in V_f forms a unique maximal clique in HCG at the i -th iteration, the generated classes will be $A_{2i-1} = V_f$ and $A_{2i} = \emptyset$, where $\emptyset$ is a null set. The relation between the number of iterations in the modified algorithm and the channel width of routing problem is described by Lemma 1.

Lemma 1: Let $\{A_1, A_2, \dots, A_m\}$ be a topological order of independent classes. The channel width for three-layer routing problem equals $m/2$.

Proof: Two independent classes are generated at each iteration. The nets in one independent class will be assigned to first horizontal layer and the other for second horizontal layer on the same track. Therefore, the number of independent classes is proportional to both the channel width of three-layer routing problem and the number of iterations in the algorithm $\neq 1$.

It is noted that the low bound of channel width, denoted as ω , for the three-layer restrictive routing is $\max(\lceil d_{max}/2 \rceil, v_{max})$ since at most two layers can be used to route the horizontal wire segments. The optimal solution to three-layer restrictive channel routing problem is to find a topological order of independent classes $\{A_1, A_2, \dots, A_m\}$ such that $m = 2\omega$ and nets in the class A_i and A_{i+1} are assigned to $\lceil i/2 \rceil$ -th track on two horizontal layers, respectively. Let λ_i be the longest path length from vertex i in VCG to the vertex which has no successors, and δ_i be the dynamic local density of net i . The dynamic local density of net i is defined to be the maximal number of nets which have not been assigned to tracks and are overlapping each other in x -interval of net i . It is clear to show that the algorithm requires at least $\max(\lambda_i, \lceil \delta_i/2 \rceil)$ iterations to finish the routing problem since the independent class, which contains the net i , was generated. To prove that the sequence of independent classes is an optimal partial order, the following theorem is required.

Theorem 1: The sequence $\{A_1, A_2, \dots, A_m\}$ is an optimal partial order of independent classes if the sequence follows the condition which is defined as

$$\omega - \lceil j/2 \rceil \geq \max(\lambda_i, \lceil \delta_i/2 \rceil) \quad (1)$$

for each net i in class A_j .

Proof: The net i in the class A_j is generated at the $\lceil j/2 \rceil$ -th iteration. It is clear to show that the number of iterations needed for the completion of the algorithm is at least $\max(\lambda_i, \lceil \delta_i/2 \rceil)$ after the net i was routed in the $\lceil j/2 \rceil$ -th track. Lemma 1 shows that the channel width needed in the routing problem equals the number of iterations in the algorithm. If the low bound of channel width is ω , we have

$$\lceil j/2 \rceil + \max(\lambda_i, \lceil \delta_i/2 \rceil) \leq \omega$$

and it proves that

$$\omega - \lceil j/2 \rceil \geq \max(\lambda_i, \lceil \delta_i/2 \rceil)$$

The channel routing is a well known NP-complete problem [13]. Therefore, we conjecture that the determination of optimal partial order of independent classes is an NP-complete problem. A heuristic algorithm is presented. In the proposed algorithm, every vertex in HCG is weighted by a cost, and each time two independent classes with maximum weights are selected first. The cost estimated on all vertices in HCG does not only involve the x -length of corresponding net but also consider its longest path length and dynamic local density of the net. The cost W_i of net i in the independent class A_j is evaluated as follows.

$$W_i = \begin{cases} x\text{-length of net } i & \text{if } 2\omega - j > \max(2\lambda_i, \delta_i) \\ \rho & \text{if } 2\omega - j = \max(2\lambda_i, \delta_i) \end{cases} \quad (2)$$

The cost of A_j is the summation of cost of all nets in it. A net with cost ρ is called dominant net. The ρ is a dominant cost and forces the independent class which contains the dominant net to be selected first.

Consider the netlist as shown in Fig. 1, the process on determination of partial order of independent classes for the track assignment is presented in Fig. 7 and its corresponding transient layout is demonstrated in Fig. 3a.

Fig. 7 Steps in determining independent sets

4 Topological sorting algorithm

In the track assignment, the topological order of independent classes will be generated based on HCG and VCG. The algorithm to determine the topological order is called topological sorting in VCG. After the determination of topological order, each net can be assigned to a track in the channel based on the predetermined order. Nets in an independent class are assigned to same track. Hence, the number of classes in the topological sequence decides the number of tracks needed. In this section, we concentrate on the discussion of topological sorting algorithm. First, the data structure used in the algorithm is introduced, and the algorithm is then explained.

The algorithm assumes that VCG is represented by the adjacency list. The adjacency list for VCG is represented as n linked lists while there are n vertices and e edges in VCG. Each list includes both a headnode and list nodes. The headnodes of these lists represent vertices in VCG and contain two fields: *COUNT* and *LINK*. The *COUNT* field contains the in-degree of that vertex and the *LINK* field is a pointer to the first node in the adjacency list. Each list node contains both *VERTEX* and *LINK* fields. The *VERTEX* field records the index of vertex in VCG and the *LINK* field is a pointer to next node in the linked list.

The adjacency list for VCG is easy to be set up at the time of input. The *COUNT* field of headnode j is incremented by one when an edge $\langle i, j \rangle$ is in VCG, and the node with *VERTEX* field containing index j is linked

to the tail of list which is pointed by the headnode i . In the case of a directed graph with n vertices and e edges, this representation requires n headnodes and e list nodes. For VCG as shown in Fig. 6, the adjacency list is indicated in Fig. 8.

Fig. 8 Adjacency list of VCG

From observation of the adjacency list, we know that the headnode j with zero count corresponds to a vertex which has no predecessors. The vertices with zero count are maintained in a queue Q . We present the topological sorting algorithm as shown in the following.

ALGORITHM T_SORT

```

1. input VCG and set up the adjacency list;
2. for  $i = 1$  to  $n$  do/* put all vertices with zero count
   into  $Q$  */
3.   if  $COUNT(i) = 0$  then put vertex  $i$  into  $Q$ ;
4. end
5. while  $Q$  is not empty do
6.   set up a subgraph  $G$  with vertices in  $Q$  from HCG;
7.   output two independent classes  $A_i$  and  $A_j$ , which
   are two maximal cliques with maximal cost in  $G$ , in
   the topological sequence;
8.   for each vertex  $k$  in  $A_i$  or  $A_j$  do
9.      $ptr \leftarrow LINK(k)$ ;
10.    while  $ptr \neq 0$  do
11.       $m \leftarrow VERTEX(ptr)$ ;
12.       $COUNT(m) \leftarrow COUNT(m) - 1$ ;
13.      if  $COUNT(m) = 0$  then put vertex  $m$  into  $Q$ ;
14.    end
15.  end
16. end
END ALGORITHM T_SORT

```

Line 7 in the algorithm *T_SORT* is to determine two maximal cliques of the subgraph with vertices in Q . Since it is essential, we first explain it below. Because there is a weight assigned in each vertex in HCG, our problem is to determine the maximum weighted clique in the subgraph of HCG. In general, to find maximum weighted clique in graph is an NP-complete problem [11]. However, the problem becomes tractable when the graph is restricted to comparability graph. First, we define the comparability graph as follows.

Definition 3 [12]: Each edge can be assigned a one-way direction in such a way that the resulting oriented graph (V, F) satisfies the following transitive condition:

$$\langle a, b \rangle \in F \quad \text{and} \quad \langle b, c \rangle \in F \quad \text{imply} \quad \langle a, c \rangle \in F$$

(for all $a, b, c \in V$)

An undirected graph which is transitively orientable is called a comparability graph.

Theorem 2: The horizontal constraint graph is comparability graph.

Proof: Let two x -intervals of nets i and j be denoted as $I_i = (L_i, R_i)$ and $I_j = (L_j, R_j)$. $I_i < I_j$ means that $R_i < L_j$ or the x -interval I_i lies entirely to the left of x -interval I_j . Define an orientation F of HCG as follows:

$$\langle i, j \rangle \in F \Leftrightarrow I_i < I_j \text{ for all } i, j$$

Clearly, the transitive condition is satisfied since $I_i < I_j < I_k$ implies $I_i < I_k$. Thus F is a transitive orientation of HCG.

Theorem 3 [12]: The time complexity of maximum weight clique in the comparability graph $G_C = (V_C, E_C)$ is $O(|V_C|)$.

Theorem 4: In the case of VCG with n vertices and e edges, the complexity of algorithm T_SORT is $O(n + e)$.

Proof: For the algorithm T_SORT , the loop of lines 2–4 takes $O(n)$ time. The while loop of lines 5–16 takes ω iterations over entire program as explained in Lemma 1, where ω is channel width. In this while loop, two tasks, which are clique determination on line 7 and queue creation in for-loop of lines 8–15, are critical. The for-loop of lines 8–15 takes time order $O(d_i)$ for each vertex i , where d_i is out-degree of vertex i . Since this loop is encountered once for each vertex output, the total time for this part of algorithm is $O(\sum d_i) = O(e)$ for $1 \leq i \leq n$. The time for clique determination is $O(\sum n_i) = O(n)$ for $1 \leq i \leq \omega$ known from Theorem 3, where n_i is the number of vertices in V_i at the i -th iteration and n is the number of vertices in HCG and also is the number of vertices in VCG. Hence, the time of while-loop of lines 5–18 is $O(n + e)$, and the complexity of topological sorting is $O(n + e + n) = O(n + e)$.

5 Algorithm for layer assignment

The layer assignment assigns every wire segment in the transient layout which results from the track assignment with a layer such that the physical layout is completed with the minimum number of vias. The point where the horizontal wire segment and vertical wire segment of a net are connected in the transient layout is called the via candidate, since the point will be a real via when wire segments connected to it are located at different layers. Therefore, the layer assignment is to determine which layers can be used for routing the wire segments in the transient layout so that the via candidates which are going to be real vias can be minimised. The transient layout can be represented as a graph called Segment-Crossing Graph (SCG) which is denoted as $G_s = (V_s, E_v \cup E_c)$. The graph is constructed as follows. Each vertex in V_s represents a wire segment in the transient layout. The edges in G_s are divided into two sets which are the set of via-edges (denoted as E_v) and the set of cross-edges (denoted as E_c). If two wire segments are connected by a via candidate, an edge in E_v will connect these two corresponding vertices in SCG. An edge in E_c , which is connected two vertices in SCG, means two corresponding wire segments are cross over each other. We use dashed lines and solid lines for two different kinds of

edges. For the transient layout as given in Fig. 9, the SCG is presented in Fig. 10. For this example, $V_s = (s1, s2, \dots, s12)$, $E_v = (v1, v2, \dots, v8)$, and $E_c = (c1, c2, \dots,$

Fig. 9 The transient routing

$c10)$. $G_s = (V_s, E_v \cup E_c)$ can be divided into two sub-graphs, which are $G_s^c = (V_s, E_c)$ and $G_s^v = (V_s, E_v)$. The subgraph G_s^c is to relate the crossing between wire segments, and the subgraph G_s^v is to indicate the interconnection of nets. Each edge in G_s^c represents a via candidate. Each connected component in G_s^v means the interconnection of a net in the layout.

For a vertex $u \in V_s$, both sets of vertices are adjacent to it. For distinguishing between them, we define a set of vertices, which are adjacent to u by the edges in E_v , to be an adjacent list of u , and the other set of vertices which are connected to u by the edges in E_c is defined as a cluster of u . The adjacent list and the cluster of vertex u are denoted as $A(u)$ and $C(u)$, respectively. For SCG as shown in Fig. 10, two sets of the vertex $s2$ are $A(s2) = \{s1, s3\}$ and $C(s2) = \{s6, s7, s8, s9, s12\}$.

Fig. 10 Segment-crossing graph

Let layers in three-layer routing be represented by colours 0, 1 and 2, respectively. The layer assignment problem of the three-layer transient routing can be translated to a 3-colourable problem of G_s . The layer assignment of wire segments in the transient layout is to assign all vertices in the colourless graph G_s with colours when two vertices, which are connected by an edge in E_c , are painted by different colours, and such that two vertices, which are connected by an edge in E_v , are painted as near the same colour as possible. In other words, a colour painted on the vertex u is different from the colours assigned to the vertices in $C(u)$ and is as similar to the colours assigned to the vertices in $A(u)$ as possible. For instance, the graph SCG give in Fig. 10 can be coloured as shown in Fig. 11. Fig. 11 shows that all via candidates except $v6$ do not change into real vias. The final layout with only one via is completed as presented in Fig. 12.

A heuristic algorithm for layer assignment is proposed in this section. Let us first consider the basic concept in

Fig. 11 Colour assignment on the segment-crossing graph

the algorithm. The colours assigned to the vertices in $A(u)$ are the recommended colours of vertex u . If the vertex u is coloured with one of the recommended colours, some

Fig. 12 Physical routing
— layer 0 or colour 0
--- layer 1 or colour 1
..... layer 2 or colour 2

via candidates connected to the wire segment which corresponds to u can be eliminated. The elimination of a via candidate means the wire segments connected to it are routed on same layer such that a real via is not created. The colours assigned to the vertices in $C(u)$ are the constrained colours of vertex u . The constrained colour represents a colour to which the vertex u cannot be assigned for avoidance of violating the layout requirement. Therefore, the expected colours of vertex u to eliminate the via candidates are the colours which belong to the recommended colours of vertex u but not the constrained colours of vertex u . For formulating the heuristic algorithm, some definitions associated with each vertex in V_s are needed. Let n be the number of vertices in V_s . The definitions are listed as follows.

Definition 4: A colour that has been assigned to a vertex u is called an active colour of u denoted as $a(u)$. The vertex u that has assigned a colour is called an active vertex, and it is called an inactive vertex if it has not been assigned a colour.

Definition 5: The recommended colours of a vertex u , which is denoted as $r(u)$, are the colours that have been assigned to the vertices in $A(u)$. The $r(u)$ can be calculated as following:

$$r(u) = \{a(i) | 1 \leq i \leq n \text{ and } i \in A(u)\} \quad (3)$$

Definition 6: The constrained colours of a vertex u , which is denoted as $c(u)$, are the colours that have been assigned to the vertices in $C(u)$. The $c(u)$ can be defined as following:

$$c(u) = \{a(i) | 1 \leq i \leq n \text{ and } i \in C(u)\} \quad (4)$$

Definition 7: The expected colours of vertex u , which is denoted as $e(u)$, are the colours that can be assigned to the vertex u . The $e(u)$ can be defined as following:

$$e(u) = r(u) - (r(u) \cap c(u)) \quad (5)$$

Definition 8: A vertex u is called a p -constrained vertex (or called pC vertex for simplicity) if $|c(u)| = p$, where $|c(u)|$ is the number of disjoint colours in $c(u)$.

These definitions can be explained by the subgraph of SCG as shown in Fig. 13. The example shows that $r(u) = \{0, 1, 2, 2\}$, $c(u) = \{0, 0, 0\}$, $e(u) = \{1, 2\}$ and the vertex u is 1C vertex. On the consideration of above definitions, our proposed heuristic algorithm can be presented step by step and the details will be addressed later.

1. Input a layout and construct a segment-crossing graph G_s , and let all vertices in G_s be inactive vertices.
2. Select an inactive vertex u in G_s .
3. Assign vertex u to a colour, $a(u)$.
4. Calculate $r(i)$ by $r(i) = r(i) \cup a(u)$, for all $i \in A(u)$.

5. Calculate $c(i)$ by $c(i) = c(i) \cup a(u)$, for all $i \in C(u)$.
6. Repeat steps 2–5 until all vertices in G_s are active vertices.

Fig. 13 Example of definitions of vertex u
○ inactive vertex ⊙ active vertex with colour n

Initially, all vertices in G_s are inactive vertices. One of them will be selected for assigning a colour to it. Because the result of the algorithm depends on the order of selecting the inactive vertices, we use a priority function to determine the sequence of the selected vertices. The priority function is estimated at each vertex in G_s . Step 2 selects an inactive vertex based on the nonincreasing order of priority of the vertices. Let the valence $\text{val}(G_s^c, u)$ of a vertex u in a graph G_s^c be the number of edges incident with u and $\text{val}(G_s^c)$ be the maximum number of valence in graph G_s^c . The priority function of vertex u is defined as follows.

$$P(u) = |c(u)| \times \text{val}(G_s^c) + \text{val}(G_s^c, u) \quad (6)$$

In this priority function, the term $\text{val}(G_s^c)$ means the complexity of cross over of wire segments in the routing, and the term $\text{val}(G_s^c, u)$ indicates the number of wire segments which are crossing with the vertex u or the degree of constraints of the vertex u . Definition 8 shows that the inactive vertices may be classified in some kinds based on the $|c(u)|$. Each inactive vertex is going to be one of 0C, 1C, 2C and 3C kinds when there are three colours needed for the three-layer routing. Because of $\text{val}(G_s^c) \geq \text{val}(G_s^c, u)$, the highest priority vertex is 3C vertex and lowest one is 0C vertex. If some inactive vertices belong to the same kind, one with highest degree of constraints will be selected first from them. Initially, all vertices in G_s are 0C vertices.

In Step 3, we have to decide the active colour assigned to the vertex selected in Step 2. Let $L = \{0, 1, 2\}$ be a set of colours corresponding to the three layers used in the routing. The colours which can be assigned to the vertex u must be in the set of $L - c(u)$. However, the feasible colour of $a(u)$ must be in the set of $e(u)$. Therefore, if $a(u) \notin e(u)$ or $e(u) = \emptyset$ (where $\emptyset$ is null set), some vias will be created. For Fig. 14, both via candidate (u, x) and via

Fig. 14 Example of colour assignment

candidate (u, y) cannot be eliminated because $e(u) = \emptyset$ and $a(u) = 2$. Let $|r_\tau(u)|$ be the number of colour τ in $r(u)$ and $\tau \in L$. For eliminating the maximum number of via candidates, the colour assigned to an inactive vertex u will be $a(u)$ satisfying the feasible condition that is defined below.

$$a(u) = \tau, \tau \in (u) \text{ and } |r_\tau(u)| \text{ is a maximum} \quad (7)$$

For the example as shown in Fig. 13, the $a(u)$ will be 2 to eliminate two via candidates because of $|r_1(u)| = 1$ and $|r_2(u)| = 2$.

The determination of $a(u)$ is different from each kind of inactive vertex u . First, we list the formulae to determine the $a(u)$ of different kinds of vertex u . We explain in detail later.

A: 0C vertex u ($|c(u)| = 0$)

$$\begin{aligned} \text{A1: } r(u) &= \emptyset \\ a(u) &= \tau, \tau \in L \text{ with highest priority} \end{aligned} \quad (8)$$

$$\begin{aligned} \text{A2: } r(u) &\neq \emptyset \\ a(u) &= \tau, \tau \in r(u) \text{ and } |r_\tau(u)| \text{ is maximum} \end{aligned} \quad (9)$$

B: 1C vertex u ($|c(u)| = 1$)

$$\begin{aligned} \text{B1: } r(u) \cap c(u) &= \emptyset, \text{ or } e(u) = r(u) \\ a(u) &= \tau, \tau \in r(u) \text{ and } |r_\tau(u)| \text{ is maximum} \end{aligned} \quad (10)$$

$$\begin{aligned} \text{B2: } r(u) \cap c(u) &\neq \emptyset \\ \text{B2a: } r(u) &\subset c(u), e(u) = \emptyset \\ a(u) &= \tau, \tau \in (L - c(u)) \text{ with highest priority} \\ \text{B2b: } r(u) &\not\subset c(u), \text{ or } e(u) = r(u) - (r(u) \cap c(u)) \\ a(u) &= \tau, \tau \in e(u) \text{ and } |r_\tau(u)| \text{ is a maximum} \end{aligned} \quad (11)$$

C: 2C vertex u ($|c(u)| = 2$)

$$\begin{aligned} \text{C1: } e(u) & \\ a(u) &= e(u) \end{aligned} \quad (13)$$

$$\begin{aligned} \text{C2: } e(u) &= \emptyset \\ a(u) &= L - c(u) \end{aligned} \quad (14)$$

D: 3C vertex u ($|c(u)| = 3$)

$$a(u) = \phi, \text{ where } \phi \text{ is null element} \quad (15)$$

It is easy to show that the formulae in eqns. 9, 10, 12, and 13 satisfy the feasible condition eqn. 7 of $a(u)$. Therefore, there are $|r_\tau(u)|$ via candidates to be eliminated in the cases associated with these formulae. In other formulae, the via candidates cannot be eliminated since the corresponding formulae do not follow the feasible condition. In the case of 3C vertex, the $a(u) = \phi$ indicates that it is in contradiction to 3-colourable assumption. In order to hold the 3-colourable requirement, the colour reassignments of some active vertices are needed during the colour assignment of 3C vertex. This backtracking to reassign the active vertices with new colours will be discussed in the next section.

In eqn. 8 or 11, it derives a question: 'Which colour in L or $(L - c(u))$ will be selected first?' This is a priority problem of layers in routing. If one objective of the layer assignment is to maximise the total amount of metal, the priority of layers can be set to this order which is

$0 > 1 > 2$ when both layer 0 and layer 1 are assumed to be two metal layers. However, by trying to maximise the total length of metal wires may incur more vias. At the aim of via minimisation, the priority order of layers is $1 > 0 > 2$ since the layer 1 is the middle layer which has more flexible to be assigned than layers 0 and 2. The priority of layers is determined to maximise the use of preferred routing layers and minimise the via usage.

Now, we present the heuristic algorithm in the following.

ALGORITHM LAYER_ASSIGNMENT

```

1. input the layout and construct  $G_s$ ;
2. repeat
3.   select an inactive vertex  $u$  based on the priority
      function;
4.   if ( $|c(u)| = 3$ ) then call BACKTRACK( $u$ );
      /* 3C vertex */
5.   else if ( $|c(u)| = 2$ ) then /* 2C vertex */
6.      $a(u) = L - c(u)$ ; /* an unique layer */
7.     else if ( $|c(u)| = 1$ ) then /* 1C vertex */
8.       if ( $r(u) \cap c(u) = \emptyset$ ) then  $a(u) = (10)$ ;
9.       else if ( $r(u) \subseteq c(u)$ ) then  $a(u) = (11)$ ;
10.      else  $a(u) = (12)$ ;
11.      else if ( $r(u) = \emptyset$ ) then /* 0C vertex */
12.         $a(u) = (8)$ ;
13.      else  $a(u) = (9)$ ;
14.   for each vertex  $i$  in  $A(u)$  do
15.      $r(i) = r(i) \cup a(u)$ ;
16.   end
17.   for each vertex  $i$  in  $C(u)$  do
18.      $c(i) = c(i) \cup a(u)$ ;
19.   end
20. until (all vertices in  $G_s$  are active)
END ALGORITHM LAYER_ASSIGNMENT

```

6 Backtracking in the layer assignment

The backtracking will be treated to reassign some of the active vertices with new colours so that the 3C inactive vertex can be assigned to an active colour. Since all colours in L have been assigned to the vertices in the cluster of the 3C vertex u , a colour, which is assigned to some of the vertices in $C(u)$, is selected to be an active colour of u . The vertex in $C(u)$ will change to be inactive after its colour is used to be an active colour of the 3C vertex u . Each vertex in $C(u)$ may be one of the 0C, 1C or 2C vertices. If a colour assigned to the 2C vertex in $C(u)$ is selected for the colour assignment of the 3C vertex u , this 2C vertex will change to be a 3C inactive vertex such that the work of backtracking is continued. In the other case of a 0C or 1C vertex in $C(u)$, the backtracking will be terminated because the vertex is not going to be a 3C vertex after its colour is assigned to the 3C vertex u . The requirement of selecting a colour is to minimise the number of active vertices which will become 3C vertices after this colour is assigned to the current 3C vertex. For the example as shown in Fig. 15, all wire segments except

Fig. 15 Example for backtracking

s_u are active and $C(s_u) = \{s_i, s_j, s_k\}$. The wire segments s_i , s_j and s_k are corresponding to the 0C, 1C and 2C vertices, respectively, and the wire segment s_u corresponds to the 3C inactive vertex because of $a(s_i) = 2$, $a(s_j) = 0$, and $a(s_k) = 1$. The colour which is assigned to the 0C vertex s_i will be selected as a colour assignment of the vertex s_u , since the vertex s_i will become a 1C inactive vertex which has more flexibility to reassign. But it is not reasonable for selection of the colour corresponding to the vertex s_k , because the vertex s_k will be a 3C inactive vertex after its colour is assigned to s_u . The algorithm of backtracking is presented in the following:

ALGORITHM BACKTRACK (u)

```

/* the  $u$  is 3C vertex */
1. do
2.   calculate the  $k_\tau(u)$  and  $|c_\tau(u)|$  for  $\tau = 0, 1$  and  $2$ ;
3.    $a(u) = \tau$ ,  $k_\tau(u)$  and  $|c_\tau(u)|$  are minimum;
4.   for all  $i$  in  $C(u)$  do
5.      $c(i) = c(i) \cup a(u)$ ;
6.     if ( $a(i) = a(u)$ ) then [ set vertex  $i$  to be inactive;
7.       for all  $j$  in  $A(i)$  do
8.          $r(j) = r(j) - a(i)$ ;
9.       end
10.      for all  $j$  in  $C(i)$  do
11.         $c(j) = c(j) - a(i)$ ;
12.      end]
13.     if ( $|c(i)| = 3$ ) and ( $i$  does not be marked)
14.       then  $S \leftarrow i$ ; /* push  $i$  into  $S$  */
15.   end
16.   for all  $i$  in  $A(u)$  do
17.      $r(i) = r(i) \cup a(u)$ ;
18.   end
19.   vertex  $u$  is marked;
20.   if ( $S$  is empty) then return;
21.   else  $u \leftarrow S$ ; /* pop from  $S$  to  $u$  */
22. forever
END ALGORITHM BACKTRACK

```

Two parameters are needed in the algorithm BACKTRACK. We have to define the parameters for understanding this algorithm. Note that only one of three colours in the layer assignment will be selected for an active colour of the 3C inactive vertex. Both parameters of $k_\tau(u)$ and $|c_\tau(u)|$ are used to determine the active colour assigned to the 3C vertex. The $k_\tau(u)$ is defined as follows.

$$k_\tau(u) = \max (|c(i)|, \text{ for all vertices } i \in C(u) \text{ and } \tau = a(i)) \quad (16)$$

The $k_\tau(u) = p$ means some of the vertices in $C(u)$, which have been assigned to a colour τ , are going to be $(p + 1)$ constrained vertices after the τ colour was selected for an active colour of the 3C vertex u . For the example as shown in Fig. 16, the inactive wire segment u corresponds to the 3C vertex and the $k_0(u) = 2$, $k_1(u) = 1$, $k_2(u) = 1$. If the colour 0 is assigned to the 3C vertex u , the wire segment s_2 will become a 3C inactive vertex, meanwhile, the backtracking will be continued. Therefore, the assigned colour for a 3C vertex u may be 1 or 2.

Fig. 16 Example for termination in backtracking

The other parameter $|c_\tau(u)|$ is defined to be the number of active vertices which will become the inactive vertices after colour τ is assigned to the 3C vertex u . The $|c_\tau(u)|$ is calculated below:

$$|c_\tau(u)| = \text{The count of vertices } i \text{ for } a(i) = \tau \text{ and } i \in C(u) \quad (17)$$

From the example of Fig. 16, one of the colours 1 and 2 can be used as an active colour of the 3C vertex u . However, the colour 2 is better to be an active colour of the 3C vertex than the colour 1 since $|c_1(u)| = 2$ and $|c_2(u)| = 1$. The active colour $\tau = \min (|c_\tau(u)|)$, (for $\tau = 0, 1$ and 2) means that the number of active vertices which become inactive vertices are minimum or the resulted iterations in the backtracking process is minimum after colour τ was assigned to the 3C vertex. Therefore, the active colour is determined in the following formula:

$$a(u) = \tau, k_\tau(u) \text{ and } |c_\tau(u)| \text{ are minimum for } \tau = 0, 1 \text{ and } 2 \quad (18)$$

Some vertices, which are going to be 3C vertices after a colour is assigned to the 3C inactive vertex, will be stored in the stack S . The 3C vertex which is assigned to an active colour in the backtracking will be marked. The marked vertex indicates it is not going to be inactive again during the iterations of backtracking to avoid the oscillation in the iterations. For the cases as shown in Fig. 16, the backtracking will be terminated since there are not new 3C vertices to be generated. However, in the case as shown in Fig. 17, the backtracking will be continued because of $k_\tau(u) = 2$ for $\tau = 0, 1$ and 2 .

Fig. 17 Worst case in backtracking

7 Results and conclusions

Both algorithms for track assignment and layer assignment were coded in C language and implemented on the SUN 3/110 workstation. The algorithms are efficient and have been evaluated on some examples including examples 1, 3a, 3b, 3c, 4b and 5 presented in Reference 5 and Deutsch's difficult example [11]. The solution quality with respect to tracks used is comparable to the solution quality presented in Chen and Liu [5] and the performance of via usage is better than them. The performance of solutions is given in Table 1. The number of tracks used for all examples except 3b are consistently the minimum. In an average case, the number of vias used is reduced by 30 percent or so. Fig. 18–23 are the final layout for examples 1, 3a, 3b, 3c, 4b and 5, respectively. The final layout of Deutsch's difficult example is presented in Fig. 24.

In the theoretical discussion of the layer assignment problem, we can assume that the terminal is available on all layers. On the basis of this assumption, the best results shown in Table 1 are reached. However, on the practical manufacturing of IC processing, some applications restrict the availability of terminals to particular layers. If terminals are available for one particular layer, the layer assignments of wire segments connected to terminals will be preassigned to this particular layer and their corresponding vertices in SCG are set to be active. This pre-

Table 1

Example	# Net	# Column	Low Bd.	Router in [5]		Delayed layering router		
				#Track	#vias	#Track	#vias	reducing rate
1	21	35	7	7	57	7	36	36.8
3a	30	45	8	8	68	8	51	25
3b	47	61	9	10	107	10	79	26.2
3c	54	79	9	9	125	9	96	23.2
4b	57	119	13	13	179	13	106	40.8
5	63	119	10	10	150	10	102	32
diff.	72	174	23	23	290	23	204	29.7

Fig. 18 Final layout of example 1

Fig. 21 Final layout of example 3c

Fig. 19 Final layout of example 3a

Fig. 22 Final layout of example 4b

Fig. 20 Final layout of example 3b

Fig. 23 Final layout of example

Fig. 24 Final layout of Deutsch's example

assignment tasks of particular wire segments can be done in the preprocessor before the algorithm *LAYER_ASSIGNMENT* is started. Table 2 gives the results of all examples with terminal restriction. The experiments show that the terminal restriction may

Table 2

Examples	Case 1* # via	Case 2** # via
1	55	44
3a	67	56
3b	95	84
3c	120	106
4b	163	122
5	144	113
diff.	273	227

* All terminals are restricted at only one layer

** All terminals can be at any one of two proper layers

degrade the performance of layer assignment although the solution quality with respect to via usage was better than for the previous solution.

We have described a new channel router that uses the delayed layering to reduce the via usage. This router has been run successfully on several examples. In all instances, the via usages in their solutions were better than for the previous routers, and the solution quality with respect to channel width was comparable to the previous solutions. The delayed layering router is also able to maximise the use of a preferred layer. The priority of layers can be $1 > 0 > 2$ to maximise the total length of metal wires and minimise the via usage.

8 References

- 1 YOSHIMURA, T., and KUH, E.S.: 'Efficient algorithms for channel routing', *IEEE Trans.*, 1982, **CAD-1**, (1), pp. 25-35
- 2 RIVEST, R.L., and FIDUCCIA, C.M.: 'A greedy channel router'. *Proc. ACM-IEEE 19th Design Automat. Conf.*, June 1982, pp. 418-424
- 3 BURSTEIN, M., and PELAVIN, R.: 'A hierarchical channel router'. *IEEE Trans.*, 1983, **CAD-2**, pp. 223-234
- 4 LEONG, H.W., and LIU, C.L.: 'Discretionary channel routing', *IEEE Proc.*, 1988, **135**, pt. G, (2), pp. 45-57
- 5 CHEN, Y.K., and LIU, M.L.: 'Three-layer channel routing', *IEEE Trans.*, 1984, **CAD-3**, (2), pp. 156-163
- 6 PITCHUMANI, V., and ZHANG, Q.: 'A mixed HVH-VHV algorithm for three-layer channel routing', *IEEE Trans.*, 1987, **CAD-6**, (4), pp. 497-502
- 7 HAMBRUSCH, S.E.: 'Channel routing algorithm for overlap models', *IEEE Trans.*, 1985, **CAD-4**, (1), pp. 23-30
- 8 PREPARATA, F.P., and LIPSKI, W.: 'Optimal three-layer channel routing', *IEEE Trans.*, 1984, **C-33**, (5), pp. 427-437
- 9 BRUELL, P., and SUN, P.: 'A "greedy" three-layer channel router'. *Proc. ICCAD-85*, Nov. 1985, pp. 298-300
- 10 DEUTSCH, D.N.: 'A dogleg channel router'. *ACM-IEEE 20th Design Automat. Conf.*, June 1983, pp. 591-597
- 11 GAREY, M.R., and JOHNSON, D.S.: 'Computers and intractability' (W.H. Freeman, San Francisco, 1979)
- 12 GOLUMBIC, M.C.: 'Algorithmic graph theory and perfect graphs' (Academic Press, New York, 1980)
- 13 SZYMANSKI, T.G.: 'Dogleg channel routing is NP-complete', *IEEE Trans.*, 1985, **CAD-4**, (1), pp. 31-40