A New Approach to Three- or Four-Layer Channel Routing

JINGSHENG CONG, D. F. WONG, AND C. L. LIU, FELLOW, IEEE

Abstract—**We present in this paper a new approach to the three- or four-layer channel routing problem. Since two-layer channel routing has been well studied, there are several two-layer routers which can produce optimal or near optimal solutions for almost all the practical problems. We develop a general technique which transforms a two-layer routing solution systematically into a three-layer routing solution. This solution transformation approach is different from previous approaches for three-layer and multilayer channel routing. Our router performs well in comparison with other three-layer channel routers proposed thus far. In particular, it provides a ten-track optimal solution for the famous Deutsch's difficult example, whereas other well known three-layer channel routers required 11 or more tracks. We extend our approach to four-layer channel routing. Given any two-layer channel routing solution without an unrestricted dogleg that uses w tracks, our router can provably obtain a four-layer routing solution using no more than $\lceil w/2 \rceil$ tracks. We also give a new theoretical upper bound $\lceil d/2 \rceil + 2$ for arbitrary four-layer channel routing problems.**

I. Introduction

A KEY PROBLEM in VLSI layout design and implementation is the channel routing problem. The two-layer channel routing problem has been studied extensively in the past ten years [4], [9], [20]–[22]. There are several two-layer channel routers which can produce channel routing solutions using at most one or two tracks more than channel density for most practical problems. With the advance in VLSI technology, utilization of more than two layers for signal routing has become feasible. As mentioned in [3], such a possibility has been exploited in the design and implementation of a number of gate arrays. The Motorola 2900ETL macrocell array is a bipolar gate array which uses three metal layers for routing. As many as four metal layers are used by the masterslice array in the IBM 4331 system. In MOS technology the one megabit DRAM designed by Taguchi *et al.* uses four routing layers, three layers of polysilicon and one layer of metal. Thus, the design and implementation of channel routing algorithms using a small number of layers (usually three or four layers) are not only practical, but also are becoming more and more important.

The multilayer channel routing problem has been studied in the literature. Chen and Liu [5] presented a three-layer channel router based on the net merging method used by Yoshimura and Kuh [22] for two-layer channel routing. Bruell and Sun [3] designed a "greedy" router for three-layer channel routing and obtained the first 11-track solution for Deutsch's difficult example. Braun *et al.* [2] implemented a multilayer channel router which divides layers into several groups. Each group contains two or three layers and routing for each group is done by the extended two-layer router YACR2 [20]. Enbody and Du [11] developed a multilayer router using leading column heuristics and limited backtracking. As for theoretical results, Hambrusch [15] obtained some near-optimal upper bounds for the case of two terminal nets allowing mixed wiring on the same layer. Brady and Brown [1] proposed an algorithm which produces asymptotically optimal results when the number of layers is large. However, their algorithm does not include the case of three-layer routing. Also, their bounds on four-layer routing are quite loose for most known problems.

In this paper, we present a new approach to the three-layer channel routing problem which can also be easily extended to the four-layer channel routing problem. Instead of trying to build a three-layer solution directly, as in previous approaches, we take advantage of existing, well-designed two-layer routers, and develop a method to transform a two-layer routing solution systematically into a three-layer routing solution. Our three-layer router performs well in comparison with other known three-layer routers or multilayer routers for all frequently quoted examples. In particular, we obtain a ten-track optimal three-layer solution for the famous Deutsch's difficult example. For four-layer channel routing, we can prove that our approach is guaranteed to produce a solution using no more than $\lceil w/2 \rceil$ tracks, where w is the number of tracks used in a two-layer routing solution without unrestricted dogleg. Consequently, we are able to obtain quite easily optimal four-layer routing solutions for most examples in the literature, including Deutsch's difficult example.

Manuscript received February 5, 1988; revised June 8, 1988. This work was partially supported by the National Science Foundation under Grant MIP 87-03273, by the Semiconductor Research Corporation under Contract 87-DP-109, by a grant from the General Electric Company, and by the Texas Advanced Research Program. The review of this paper was arranged by Associate Editor Alfred E. Dunlop.

This is an expanded version of the work originally presented at the IC-CAD-87.

J. Cong and C. L. Liu are with the Department of Computer Science, University of Illinois, Urbana, IL 61801.

D. F. Wong is with the Department of Computer Science, University of Texas, Austin, TX 78712.

IEEE Log Number 8822862.

II. Description of the Problem

A *channel* is a layered rectangular routing area with pins placed at the top and bottom edges. We assume that there is a grid superimposed over all the layers of the channel and that all terminals are on the grid points along the top and bottom edges of the channel. The *channel routing problem* is to connect pins in each net using a minimum number of tracks.

A *valid routing solution* must comply with the following design rules: 1) Wires may be routed only on grid edges. No two wires of different nets can share a common grid edge or a grid point. We say that there is a *horizontal wiring violation* if two horizontal wire segments share a grid edge or grid point. Similarly, we say that there is a *vertical wiring violation* if two vertical wires share a grid edge or a grid point. 2) Each layer is reserved exclusively for horizontal or vertical wires. We call a layer reserved for horizontal wires a *horizontal layer*, which will be denoted H. Similarly, we call a layer reserved for vertical wires a *vertical layer*, which will be denoted V. Fig. 1 shows an example of a valid two-layer channel routing solution.

A *via* is used to connect two wire segments on two adjacent layers. Two vias are said to be *adjacent* if they belong to two different nets and are on two adjacent tracks at the same column. The number of nets crossing a column is called the *local density* at that column. The maximum of all local densities is called the *channel density*, and is denoted d. In this paper, *channel width* is measured by the number of tracks used in a channel routing solution. Obviously, in two-layer channel routing, channel density is a lower bound of the channel width. In multilayer channel routing, if the number of horizontal layers is L, a lower bound of the channel width is $\lceil d/L \rceil$.

In general, introduction of doglegs may reduce the channel width [9]. If a dogleg of a net occurs at some column where a terminal of the net is located, we call the dogleg a *restricted dogleg*; otherwise, we call it an *unrestricted dogleg*.

III. A New Approach to Three-Layer Channel Routing

There are two possible ways to assign layer types in three-layer channel routing: one is VHV, and the other is HVH. In the VHV model, we can always use the left-edge algorithm [16] to obtain a solution using d tracks, which is the best possible solution since we have only one horizontal layer. But such a solution is usually not the optimal solution for three-layer channel routing. In the HVH model we have two horizontal layers; the lower bound on the number of tracks needed is $\lceil d/2 \rceil$. Although we can not always obtain a $\lceil d/2 \rceil$ track solution due to the vertical wiring constraints, for most practical problems the HVH model uses fewer tracks than the VHV model. Because the solution for the VHV model is trivial and the HVH model is more economical, all previous studies, as well as this paper, concentrate on the HVH model.

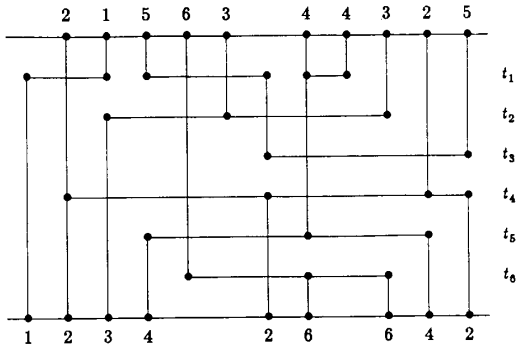


Fig. 1. An example of a two-layer channel routing solution.

A. Overview of the Algorithm

Our basic approach is to transform a known two-layer solution into a three-layer solution. Let S be a two-layer solution using w tracks $t_1, t_2, \cdots, t_w$. The general idea of our approach is to distribute the tracks evenly on the two horizontal layers. We assign t_1 to track one on layer one, and t_2 to track one on layer three, then t_3 to track two on layer one, and t_4 to track two on layer three, and so on. In general, we assign t_{2k-1} to track k on layer one and t_{2k} to track k on layer three. Two tracks in S that are assigned to the same track on the two horizontal layers are referred to as a *folded pair* of tracks. Because there is no horizontal wiring violation in the two-layer solution S, there is no horizontal wiring violation among the horizontal tracks on layers one and three. Then we try to use the vertical layer (layer two) to connect horizontal wire segments as required in the two-layer solution S column by column. However, there is no guarantee that we would not have introduced vertical violations on layer two. For example, given the two-layer solution shown in Fig. 1, we assign t_1, t_3, and t_5 to tracks one, two, and three on layer one, respectively, and assign t_2, t_4, and t_6 to tracks one, two, and three on layer three, respectively. For vertical connections, we can carry out the connections at column one and column two successfully, as shown in Fig. 2(a) and (b), where the vertical cutting surface shows how the vertical connections are made at each column; i.e., we cut the channel at a column, then turn the cutting face around such that it faces the reader. In a vertical cutting surface, horizontal lines represent connecting wires in layer two, and short vertical lines represent vias introduced between layers.

For column three, we want to carry out the connection as shown in Fig. 2(c). However, we note that there is a vertical violation. The two vertical wires at this column need to share a common grid point at track one on layer two, which is not allowed according to our design rules. Such vertical violation is caused by adjacent vias in the two-layer solution S between the two tracks in a folded pair. Because the two-layer solution S does not have any vertical violation, it is easy to see that this is the only type of violation that might occur when we assign tracks to the two horizontal layers as described above. Without adja-

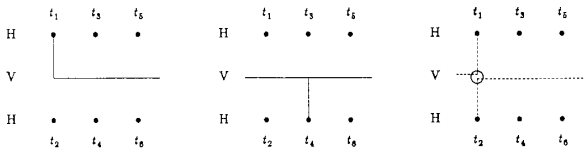


Fig. 2. Vertical cutting faces and connections at (a) column 1, (b) column 2, and (c) column 3.

cent vias between the two tracks in each folded pair in the two-layer solution, we can successfully complete the vertical connection column by column. Thus, we introduce the following definition:

Definition 1: Given a two-layer channel solution S using tracks $t_1, t_2, \cdots, t_w$, if there are no adjacent vias between tracks t_{2k-1} and t_{2k}, $k = 1, 2, \cdots, \lfloor w/2 \rfloor$, we say the solution S has a *perfect pairing*.

From the above discussion, we have the following claim: If S is a two-layer solution using w tracks with a perfect pairing, we can obtain a three-layer solution using $\lceil w/2 \rceil$ tracks. When adjacent vias exist between two tracks in a folded pair in the two-layer solution, such adjacent vias must be eliminated in order to complete the vertical connection. A simple yet effective way is to insert an empty track between the two tracks. Given a two-layer solution S, we begin from the top and pair-up tracks two by two. When we encounter adjacent vias between two tracks to be paired, we insert an empty track between these two tracks. Then we pair the upper track with the empty track, and try to pair the lower track with the track below. We repeat the procedure until we obtain a perfect pairing. For example, given the two-layer solution shown in Fig. 1, we insert four empty tracks and obtain a perfect pairing, as shown in Fig. 3.

For a given two-layer solution S, we use $e(S)$ to denote the number of empty tracks we need to insert to obtain a perfect pairing. We call $e(S)$ the *deficiency number* of solution S. Thus, the deficiency number of the example shown in Fig. 1 is four. Given a two-layer solution S using w tracks, we can obtain a three-layer solution using $\lceil (w + e(S))/2 \rceil$ tracks.

The key problem of our solution transformation technique is to modify a two-layer solution S to obtain another two-layer solution S' such that S' and S use the same number of tracks but $e(S') < e(S)$. There are two effective ways to modify a two-layer solution S to reduce $e(S)$: One is track permutation, and the other is local rerouting. For a given two-layer channel routing solution S, we may change the order of the tracks to obtain another valid two-layer solution and eliminate some adjacent vias so that $e(S)$ decreases. In other words, we try a different order of track distribution to avoid or reduce vertical wiring violation in the resulting three-layer solution. For example, given the solution shown in Fig. 1, if we exchange t_2 and t_3 we eliminate the adjacent vias both between t_1 and t_2 and between t_3 and t_4 (Fig. 4). Now we need only to insert two empty tracks to obtain a perfect pairing (Fig. 5). An-

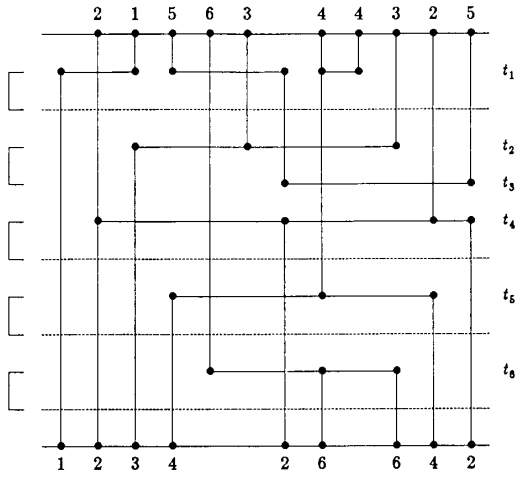


Fig. 3. A perfect track pairing of the example in Fig. 1 (dashed lines are the empty tracks inserted).

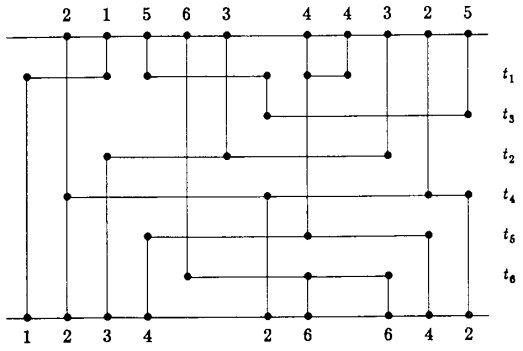


Fig. 4. After exchanging tracks 2 and 3 in Fig. 1.

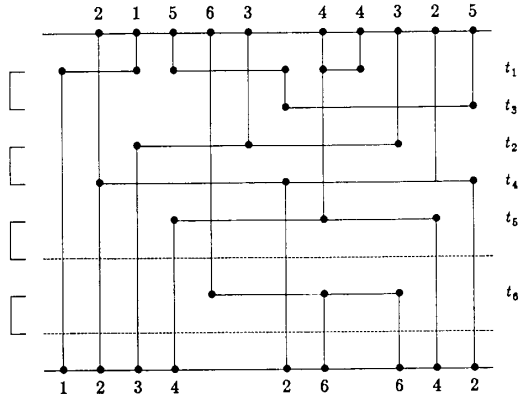


Fig. 5. The perfect track pairing of the solution in Fig. 4.

other effective way to reduce $e(S)$ is to do local rerouting to eliminate some of the adjacent vias. For the two-layer solution in Fig. 4, we can reroute net four at columns eight and nine to obtain another two-layer solution without adjacent vias between tracks five and six (Fig. 6) so that the resulting solution has a perfect pairing.

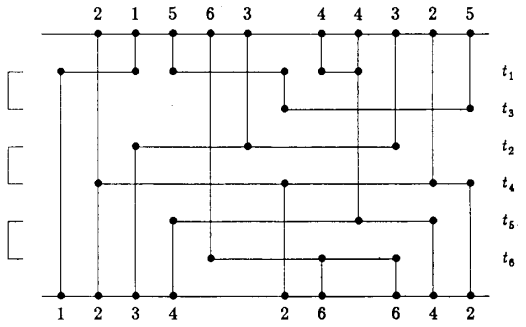


Fig. 6. The solution in Fig. 4 after rerouting.

B. Track Permutation

First, we shall discuss how the tracks in a valid two-layer routing solution can be permuted to yield another valid two-layer routing solution. Let S be a two-layer solution using tracks $t_1, t_2, \cdots, t_w$. Let π be a permutation on $\{1, 2, \cdots, w\}$. We use $\pi(S)$ to denote a two-layer wiring configuration where the $\pi(i)$th track of $\pi(S)$ is track t_i in S, and at each column there is a vertical wire segment connecting the $\pi(i)$th track and the $\pi(j)$th track in $\pi(S)$ if and only if there is a vertical wire segment connecting t_i and t_j in S. We call π *a valid track permutation* if $\pi(S)$ is a valid two-layer solution. Since there is no horizontal violation in S, for any permutation π there will be no horizontal violation in $\pi(S)$. Thus, Definition 3 is equivalent to the definition that π is a valid track permutation if and only if $\pi(S)$ has no vertical violation. To characterize all valid track permutations, we introduce the notion of a track ordering graph.

Definition 2: Let $t_1, t_2, \cdots, t_w$ denote the tracks in a two-layer solution S. The *track ordering graph* of S, denoted $TOG(S)$, is a directed graph in which each vertex v_i corresponds to the track t_i ($1 \leq i \leq w$) and there is a directed edge (v_i, v_j) if at some column there is a via on track t_i above a via on track t_j.

For the solution shown in Fig. 1, its track ordering graph is shown in Fig. 7. $TOG(S)$ is different from the vertical constraint graph (VCG) [22] since each vertex in $TOG(S)$ represents a track while each vertex in VCG represents a net. It is easy to see that for any routing solution S, $TOG(S)$ contains no directed cycle. The following two lemmas characterize valid track permutations:

Lemma 1: For a two-layer solution S using w tracks $t_1, t_2, \cdots, t_w$, if l is a topological labeling on $TOG(S)$, then the permutation $\pi = (^{1}_{l(1)}\ ^{2}_{l(2)} \cdots\ ^{w}_{l(w)})$ is a valid track permutation.

Lemma 2: Let S be a two-layer routing solution with only restricted doglegs using tracks $t_1, t_2, \cdots, t_w$. If π is a valid track permutation, then $\pi(i) = l(v_i)$ is a topological labeling on $TOG(S)$.

For the proofs of these two lemmas, see [8]. The key problem to be discussed in this section is to obtain a valid track permutation π such that $e(\pi(S))$ is minimized for a given S. A valid track permutation π for a two-layer solution S is said to be *optimal* if for any other valid track permutation π', we have $e(\pi(S)) \leq e(\pi'(S))$. In general, a given directed acyclic graph may have an exponential number of topological labelings. It seems that we might have to examine an exponential number of valid permutations in order to obtain an optimal one. However, it turns out that we can reduce the problem of finding an optimal track permutation of a given two-layer solution to the two-processor scheduling problem. Since the two-processor scheduling problem can be solved in linear time, we can obtain an optimal track permutation in linear time. The n-processor scheduling problem can be defined as follows:

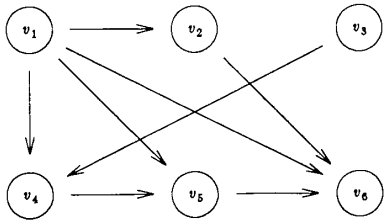


Fig. 7. The track ordering graph of the example in Fig. 1.

Input: A tuple $\Omega = \langle P, U, TPG(U) \rangle$, where $P = \{p_1, p_2, \cdots, p_n\}$ is a set of n processors, $U = \{u_1, u_2, \cdots, u_m\}$ is a finite collection of tasks, and $TPG(U)$ is a directed acyclic graph defined on U, called the task precedence graph.

Question: Find two functions τ and σ: τ is a function from U to $\{1, 2, \cdots\}$, and σ is a function from U to $\{1, 2, \cdots, n\}$, such that 1) if (u_i, u_j) is a edge in $TPG(S)$, then $\tau(u_i) < \tau(u_j)$; 2) if $\tau(u_i) = \tau(u_j)$, then $\sigma(u_i) \neq \sigma(t_j)$; 3) $T = \max_{u \in U} \tau(u)$ is minimized.

An intuitive explanation is that we have a set of tasks U, each of which can be executed on any one of the processors in P in unit time. Furthermore, task u_i must be executed before task u_j if there is a directed edge (u_i, u_j) in $TPG(S)$. A schedule $Q = (\tau, \sigma)$ specifies that task u_i would be executed at time $\tau(u_i)$ on processor $\sigma(u_i)$. We want to minimize the completion time for all the tasks. The following theorem relates the problem of finding an optimal track permutation to the two-processor scheduling problem:

Theorem 1: Let S be a two-layer channel routing solution using tracks $t_1, t_2, \cdots, t_w$. We construct an instance of the two-processor scheduling problem $\Omega(S) = \langle P, U, TPG(U) \rangle$ with $P = \{p_1, p_2\}$, $U = \{t_1, t_2, \cdots, t_w\}$, and $TPG(U) = TOG(S)$. Then S has a valid track permutation π if and only if $\Omega(S)$ has a schedule $Q = \{\tau, \sigma\}$ such that the completion time of Q equals the number of pairs in a perfect track pairing of $\pi(S)$.

Proof: If π is a valid track permutation it corresponds to a topological labeling of $TOG(S)$. Suppose that after the insertion of empty tracks we obtain a perfect pairing M of $\pi(S)$ with K pairs. We construct a schedule

$Q = \langle \tau, \sigma \rangle$ based on M as follows:

$\tau(t_i) = j$ if t_i is in the jth folded pair in M

$\sigma(t_i) = 1$ if t_i is the first track in a folded pair in M

$\sigma(t_i) = 2$ if t_i is the second track in a folded pair in M.

Since $TPG(U) = TOG(S)$, for any edge (t_i, t_j) in $TPG(U)$, it is also an edge in $TOG(S)$. Thus, we have $\pi(i) < \pi(j)$. Note that t_i is the $\pi(i)$th track in $\pi(S)$; thus, $\pi(i) < \pi(j)$ implies that $\tau(t_i) \leq \tau(t_j)$. If $\tau(t_i) = \tau(t_j)$, t_i and t_j are in the same pair in M; thus we have two tracks with adjacent vias in a same folded pair, which contradicts the fact that M is a perfect pairing. It follows that $\tau(t_i) < \tau(t_j)$. Therefore, Q thus constructed is indeed a schedule for $\Omega(S)$. Obviously, the completion time of Q is equal to K.

On the other hand, if $Q = \langle \tau, \sigma \rangle$ is a valid schedule for $\langle P, U, TPG(U) \rangle$, we sort U according to the pair $(\tau(t_i), \sigma(t_i))$. Let $\pi(t_i)$ be the index that t_i appears in the sequence. It is easy to verify that π thus defined is a topological labeling on $TOG(S)$. It follows that $\pi(S)$ is a valid track permutation. Now we construct a perfect pairing M of $\pi(S)$ as follows: Put t_i in the $\sigma(t_i)$th position in the $\tau(t_i)$th track pair, and put an empty track on the ith position in the jth pair if processor i is idle at time j ($i = 1, 2, j \geq 1$). Because t_i and t_j are in the same pair only if $\tau(t_i) = \tau(t_j)$, there is no edge (t_i, t_j) in $TOG(S)$. It follows that M is a perfect track pairing of $\pi(S)$. Obviously, the number of pairs in M equals to the completion time of Q. □

For the two-layer solution shown in Fig. 1, Fig. 8 shows the task precedence graph in the corresponding two-processor scheduling problem $\Omega(S)$, Fig. 9 shows a solution Q for $\Omega(S)$, and Fig. 10 shows the track permutation π and the perfect pairing of $\pi(S)$ induced by Q.

From Theorem 1 we conclude that finding an optimal track permutation of a given two-layer solution is equivalent to finding an optimal solution of a two-processor scheduling problem. For arbitrary n, the n-processor scheduling problem is NP-complete [14]. For fixed n ($n \geq 3$), it is still open whether the n-processor scheduling problem is NP-complete or polynomial time solvable. Fortunately, it has been known for more than a decade that the two-processor scheduling problem is polynomial time solvable [6], [12], [13]. According to the result by Gabow [13], the two-processor scheduling problem can be solved in linear time. Thus, we can obtain an optimal track permutation in time that is linearly proportional to the number of tracks in a given solution.

We note that the relative order of two tracks in the same folded pair may be changed to obtain another two-layer solution with the same deficiency number. In terms of two-processor scheduling, if two tasks can be executed at the same time, it does not matter which is executed by processor one and which is executed by processor two. We will decide the order of two tracks within each folded pair in a later stage so that local rerouting can be carried out more easily. We will discuss this in detail in subsection D after we explain our local rerouting procedure so that readers may see the motivation for deciding the relative order for the two tracks in a folded pair.

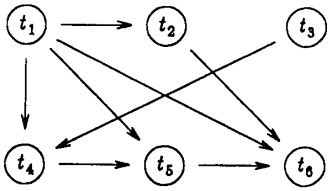


Fig. 8. The task precedence graph of $\Omega(S)$ for the example in Fig. 1.

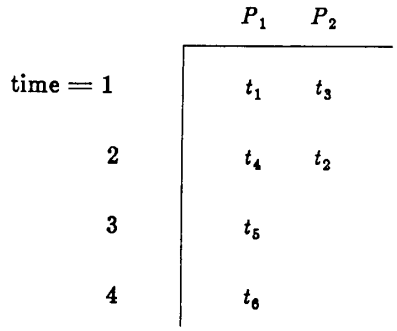


Fig. 9. An optimal solution for $\Omega(S)$.

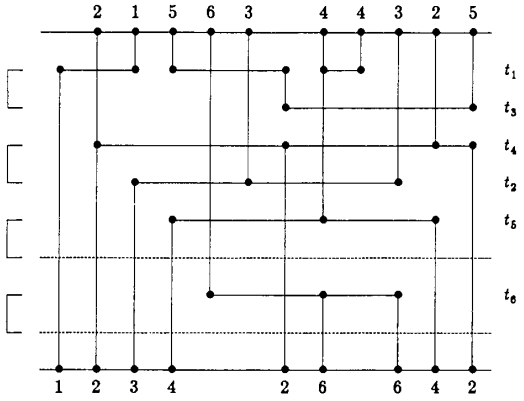


Fig. 10. The induced perfect track pairing from the optimal schedule in Fig. 9.

C. Local Rerouting and Empty Track Insertion

We call a track paired with an empty track a *singular track*. Obviously, if we can remove all the adjacent vias between two adjacent singular tracks, we can pair-up these two tracks and save two empty tracks. For the example shown in Fig. 10, obtained from the optimal solution of the two-processor scheduling problem, t_5 and t_6 are two singular tracks. If we can remove the adjacent vias at column eight (it turns out that we can), we shall obtain a perfect pairing with only three pairs.

We remove adjacent vias by carrying out local rerout-

ing. The basic idea is that given a pair of adjacent vias between two singular tracks, we tentatively remove a via in the pair and try to reconnect the two separated parts of the same net by a maze router, avoiding the deleted via and without introducing new adjacent vias between folded pairs of tracks. If we complete the connection, we succeed. If not, we try the same procedure for the other via before giving up. To be more illustrative, for the partial routing solution shown in Fig. 11(a), we remove via v_1 and try to connect the part of the net containing x with the part of the net containing the horizontal segment h_1 using a maze router (Fig. 11(b)). Our maze router is based on the classical wave-front algorithm of Lee [18]. We begin at x and expand in all possible directions at each step along unoccupied grid edges until we find a path to the part of the net containing h_1 (as shown in Fig. 11(c), if we can). For the solution shown in Fig. 10, our local rerouting procedure yields the solution shown in Fig. 12(a), which has a perfect pairing without inserted empty tracks.

In our maze router, we sometimes allow a short vertical wire in the horizontal layers. The same technique has been used in [2] and [20]. But we have a more restricted usage of such vertical wires. We only allow unit-length vertical wire to be routed on the horizontal layer to connect a terminal on the top edge to the first track or a terminal on the bottom edge to the last track. These restrictions ensure that the vertical segments in the horizontal layers will not block any horizontal tracks. Moreover, because these vertical wires are all connected to terminals and are all of unit length, there is no overlap of vertical wires in different layers. Thus, capacitive coupling will be negligible. Fig. 12(b) shows an alternative way of removing the adjacent vias between t_5 and t_6 in Fig. 10 using such short wires. Allowing to put short vertical wires on the horizontal layers, we also assume that terminals in a channel can be accessed directly from any layer. If this assumption causes a problem in some design technologies we can restrict our maze router, not allowing such displacement of the short vertical wires.

After eliminating adjacent vias between singular tracks as much as possible, if there still are singular tracks left, we insert empty tracks to obtain a perfect pairing. An important observation is that, after the insertion of an empty track, it is possible to do local rerouting again using an inserted empty track to remove more adjacent vias between other singular tracks. Thus, to exploit the presence of an inserted empty track, we combine the steps of local rerouting and empty track insertion in one iteration. Whenever we insert an empty track, we try to do more local rerouting.

D. Singular Track Shifting

As was pointed out above, if we can remove all the adjacent vias between two adjacent singular tracks we can pair them up and save two empty tracks. However, if we have two singular tracks separated by one or more track pairs as shown in Fig. 13(a), the local rerouting procedure described above would not help. In this case, we shall try

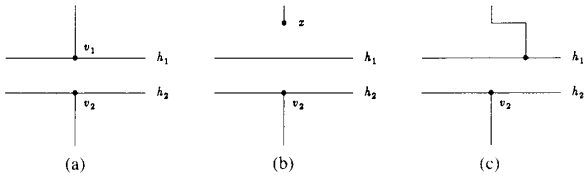


Fig. 11. Local rerouting.

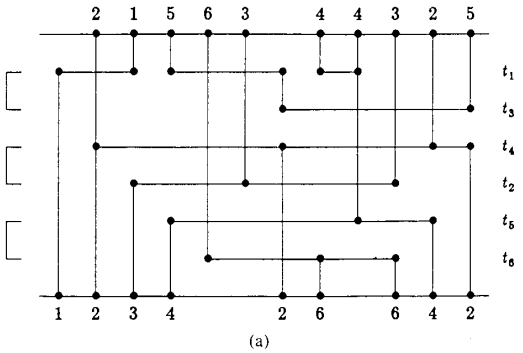


Fig. 12. (a) The solution in Fig. 10 after rerouting. (b) The solution in Fig. 10 after rerouting.

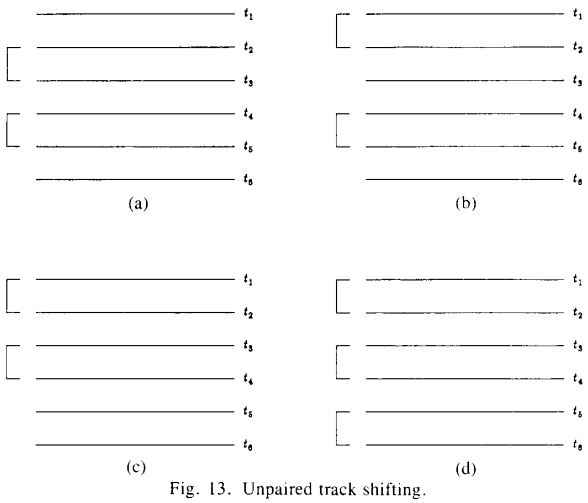


Fig. 13. Unpaired track shifting.

to use the maze router to remove all the adjacent vias between t_1 and t_2. If we succeed, we pair t_1 and t_2 together and let t_3 be a singular track, as shown in Fig. 13(b). Although we have not reduced the number of singular tracks by doing this, we move two singular tracks "closer" to each other. We call this procedure *singular track shifting*. In the next step, we try to remove all the adjacent vias between t_3 and t_4 and pair them up. If we succeed, we have two adjacent singular tracks t_5 and t_6, as shown in Fig. 13(c). If we succeed in eliminating adjacent vias between them, we obtain a perfect pairing without inserting any empty tracks (Fig. 13(d)). Because we never increase the number of singular tracks (which equals the number of empty tracks we need to insert) when we carry out singular track shifting, we are guaranteed to obtain a solution at least as good as the original one.

In the following discussion, we use the term *track group* to refer to either a singular track or a folded pair of tracks. When we do singular track shifting, we need to eliminate adjacent vias between tracks in two adjacent groups. For example, in Fig. 13(a) we need to remove, eventually, all the adjacent vias between t_1 and t_2, t_3 and t_4, and t_5 and t_6. Thus, it is desirable to obtain an optimal track permutation using an optimal two-processor scheduling algorithm such that the total number of adjacent vias between tracks in two adjacent groups is minimized.

An important observation is that as far as the number of singular tracks is concerned, the relative ordering of two tracks in the same pair is immaterial, as mentioned at the end of subsection B. Thus, we have complete freedom to switch the positions of two tracks in the same folded pair. For the solution shown in Fig. 13(a), we can switch the positions of either t_2 and t_3 or t_4 and t_5 to obtain another solution with the same number of singular tracks. In general, if we have k pairs of tracks in a two-layer solution S, we can obtain 2^k two-layer solutions with the same number of singular tracks by switching the positions of two tracks in each folded pair. We call each induced permutation of tracks a *relative ordering arrangement of* S. For each relative ordering arrangement δ, we define the *cost of* δ, denoted by $c(\delta)$, to be the total number of adjacent vias between every two adjacent tracks in two adjacent groups. We want to select a relative ordering arrangement with the minimum cost. Again, a straightforward computation has to examine all, possibly exponential in number, of the relative ordering arrangements. But we can solve this problem efficiently by reducing it to the shortest path problem of a directed graph in polynomial time.

For a given two-layer solution S obtained from the two-processor scheduling problem, we construct a directed weighted graph as follows: For each track group, if it contains a singular track t_k, we create a vertex labeled k; if it contains a pair of tracks t_i and t_j, we create two vertices, one corresponding to placing t_i above t_j and labeled ij, the other corresponding to placing t_j above t_i and labeled ji. There is a directed edge between vertex u and v if their corresponding track groups are adjacent and the track group corresponding to u is above the track group corresponding to v, and the weight on the edge $w(u, v)$ equals the number of adjacent vias between the corresponding groups when the track arrangement in each group is consistent with its corresponding vertex's labeling. We call the resulting graph the *track group order graph of* S, because each vertex in fact corresponds to a possible order of a track group. For convenience in later discussion we introduce two artificial vertices s and t, such that s is connected to all the vertices corresponding to the first track group with zero weighted edges, and t is connected to all the vertices corresponding to the last track group with zero weighted edges. For the example shown in Fig. 13(a), its track-group order graph is shown in Fig. 14, where $a(i, j)$ stands for the number of adjacent vias between t_i and t_j if they are adjacent.

Theorem 2: Each relative ordering arrangement δ of S corresponds to a path P from s to t in the track-group order graph such that $c(\delta) = w(P)$, where $w(P)$ is the sum of the weights of all the edges in the path P.

Proof: It is easy to see that each path P from s to t passes exactly one vertex for every track group. Thus, it induces a relative ordering arrangement δ such that the relative ordering in each track pair is specified by the corresponding vertex appearing in the path P. By the definition of $c(\delta)$ and the definition of the weight of the edges in the graph, we have $c(\delta) = w(P)$.

On the other hand, given any relative ordering arrangement δ, for each track group we choose the vertex for that track group such that the labeling of the vertex is consistent with the relative ordering arrangement δ. Obviously, all these vertices together with vertices s and t form a simple path from s to t. Again, we have $c(\delta) = w(P)$ by definition. □

Using the well-known shortest path algorithm by Dijkstra [10], we have the following claim: We can find a relative ordering assignment of a track permutation in quadratic time in terms of the number of tracks.

E. Overall Complexity

Fig. 15 shows a summary of our three-layer channel routing algorithm. Let c be the number of columns in a given channel routing problem; let w be the number of tracks used in the two-layer solution S obtained from an existing channel router (usually $w \leq d + 2$ for most problems). We have the following theorem on the complexity of our algorithm:

Theorem 3: The three-layer channel algorithm in Fig. 15 runs in time $O(w^3c^2)$ if the two-layer solution allows unrestricted doglegs; and runs in time $O(w^2c^2)$ if the two-layer solution allows only restricted dogleg.

Proof: At step one we call an existing two-layer router, which is not part of our implementation. Thus, its complexity is not included in our complexity analysis. In other words, we may assume that the two-layer solution has already been constructed, and therefore we need only

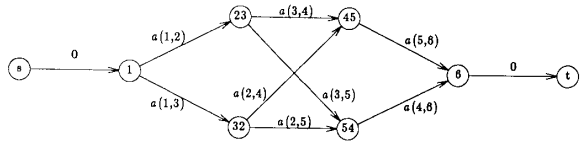


Fig. 14. The track group order graph for the example in Fig. 13(a).

Algorithm Three-Layer-Channel-Routing;
Input Two lists of terminals on the top and the bottom edge of the channel;
Output A three layer channel routing solution ;
Begin
1. Call a two layer channel router to obtain a two layer solution S;
2. Obtain optimal track permutation on S according to two processor scheduling algorithm;
3. Obtain a relative ordering arrangement with minimum cost by the shortest path algorithm;
4. *While not* (S has a perfect pairing) *do*
4.1. *If* S has two adjacent singular tracks *Then*
For each such track pair *do*
4.1.1 Remove all adjacent vias between them;
4.1.2 Pair them up;
End-for
4.2. *If* S has no adjacent singular tracks
Then Do singular track shifting;
4.3. *If not* (S has a perfect pairing)
4.3.1. Choose a singular track;
4.3.2. Pair it up with an empty track;
End-if;
End-while
5. Transform S into a three layer solution.
End.

Fig. 15. The three-layer channel routing algorithm.

to read in the solution in time proportional to the size of the solution, which is $O(wc)$. Step two takes $O(w)$ time, and step three takes $O(w^2)$ time. For step four, we note that each execution of the while-loop will decrease the number of singular tracks by at least one. Thus, the number of iterations of the while-loop is bounded by $O(w)$. Because there are at most wc adjacent vias, and to eliminate a pair of adjacent vias' maze router takes $O(wc)$ time in the worst case, the execution time of both steps 4.1 and 4.2 is bounded by $O(w^2c^2)$. Also, step 4.3 takes $O(w)$ time. It follows that the total time of step four is bounded by $O(w^3c^2)$. Obviously step five takes $O(wc)$ time. Thus, the whole algorithm runs in $O(w^3c^2)$ time. If the two-layer solution we are given contains only restricted doglegs, the total number of adjacent vias is bounded by $O(c)$. Thus, the operation time for steps 4.1 and 4.2 is bounded by $O(wc^2)$. Then the running time of the whole algorithm is bounded by $O(w^2c^2)$. □

IV. Extension to Four-Layer Channel Routing

A. Generalized Algorithm for Four-Layer Channel Routing

Our approach to three-layer channel routing can be generalized to four-layer channel routing. In choosing the layer type, it is reasonable to assume that each horizontal layer must be adjacent to a vertical layer, and that each vertical layer must be adjacent to a horizontal layer. Thus, for four-layer channel routing, we have four possible layer-type assignments: HVHV, HVVH, HVVH, and VHHV. In the following discussion we choose HVVH for our routing algorithm. We use H_1 and H_2 to denote the top and bottom horizontal layer, respectively, and we use V_1 and V_2 to denote the second and the third vertical layer, respectively. We assume that the two terminals at each column are accessible from both vertical layers. Also, we allow two vias at the same grid point on two different layers.

Again, we are going to transform a two-layer routing solution into a four-layer routing solution. Given a two-layer solution S using tracks $t_1, t_2, \cdots, t_w$, similar to our transformation for three-layer routing, we distribute the tracks of S evenly to H_1 and H_2; i.e., we assign t_1 to H_1, t_2 to H_2, t_3 to H_1, t_4 to H_2, and so on. In general, we assign t_{2k-1} to H_1 and t_{2k} to H_2. Then, we make the vertical connections column by column. At a particular column c, if there is a vertical wire connecting track t_i and t_j $(i < j)$, we make the vertical connection for the corresponding four-layer solution in four possible ways as shown in Fig. 16, depending on which layer t_i and t_j are assigned to.

If S has only restricted doglegs, at each column there are at most two vertical wires. Thus, we can complete all the vertical connections by the above wiring patterns without vertical violation. It follows that:

Theorem 4: Given a two-layer routing solution using w tracks with only restricted doglegs, we can obtain a four-layer routing solution using no more than $\lceil w/2 \rceil$ tracks. □

Applying this theorem, we can easily obtain an optimal four-layer channel routing solution for the famous Deutsch's difficult example using ten tracks from the 20 track two-layer solution produced by Yoshimura and Kuh [22], which contains only restricted doglegs. If the original two-layer solution S has unrestricted doglegs, the vertical connection method presented above still works well in most of the cases. The only exception is that in some columns S has four adjacent vias such that the first one is on t_{2k-1}, the second one is on t_{2k}, the third one is on t_{2k+1}, and the fourth one is on t_{2k+2} for some k. This situation is extremely rare in the two-layer routing solutions. If it does occur, we can do local rerouting and insert empty tracks to remove it. Thus, our transformation method can always guarantee the generation of a four-layer solution. We are not going to elaborate the algorithm because the structure of the algorithm and the techniques used are very similar to those for three-layer channel routing. It is worthwhile to mention that although we have two adjacent vertical layers v_1 and v_2, the final solution will contain no overlapping vertical wires on the two vertical layers according to our transformation.

B. An Improved Theoretical Upper Bound

Our choice of layer-type assignment HVVH may seem a bit counterintuitive because all previous works were based on the HVHV assignment. The advantage of the HVVH assignment is that when we switch from one vertical layer to the other vertical layer we do not block a track in the horizontal layers. Using the HVVH assignment, we can improve the theoretical bound in [1] for

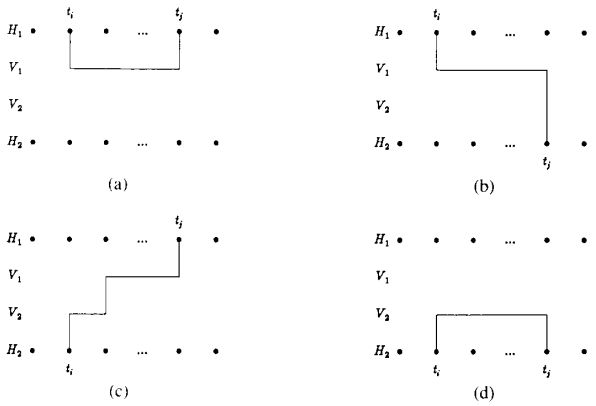


Fig. 16. Vertical connection patterns for four-layer channel routing.

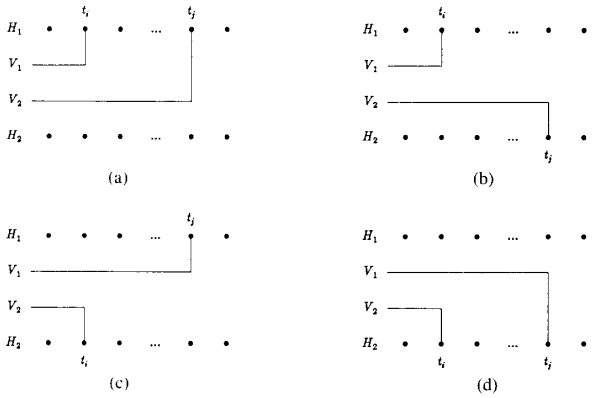


Fig. 17. Routing patterns for two vertical wires going to the top edge.

four-layer channel routing as shown in the following theorem:

Theorem 5: For a channel routing problem with density d, using the HVVH assignment we can obtain a four-layer routing solution using at most $\lceil d/2 \rceil + 2$ tracks. The running time is in $O(T)$, where T = total number of terminals occupied by the nets in the problem.

Proof: Using the method in [1] by pairing-up columns two by two and using two extra tracks at the top and bottom of the channel for jogging, we can modify the problem such that two vertical wires at each column go in the same direction without increasing the density. Then we assign evenly the horizontal interval of each net to the tracks on two horizontal layers according to the left edge algorithm [16], which will use at most $\lceil d/2 \rceil$ tracks in each layer. Then we do the vertical connection column by column. Suppose at column c two vertical wires are connected to the t_i and t_j ($i < j$) and both go to the top edge. We may route them in the four possible ways shown in Fig. 17, depending upon which layer t_i and t_j lie (again, the vertical connections are shown by the vertical cutting surface at column c).

Similar routing patterns can be designed for the two vertical wires going to the bottom edge at the same column. Thus, we can complete vertical connections successfully. The algorithm uses two tracks for jogging and $\lceil d/2 \rceil$ tracks for the horizontal wires of all the nets. Thus, the total number of tracks used is bounded by $\lceil d/2 \rceil + 2$. According to [1] the modification stage takes $O(T)$ time. Obviously, both the horizontal track assignment and the vertical wire connection take $O(T)$ time. Thus, the algorithm runs in $O(T)$ time. □

TABLE I
EXPERIMENTAL RESULTS OF OUR THREE-LAYER CHANNEL ROUTER

Ex.	d	# of tracks in the 2L solution used	# of unpaired tracks after permutation	# of adjacent vias removed	# of empty tracks inserted	# of tracks in our 3L solution	Lower bound for 3L solution
YK3a	15	15[22]	1	0	0	8	8
YK3b	17	17[22]	1	0	0	9	9
YK3c	18	18[22]	2	1	0	9	9
Diff.	19	19[4]	11	8	1	10	10
D1	18	18[23]	1	0	0	9	9
D3	15	15[19]	1	0	0	8	8
D5	17	17[19]	1	0	0	9	9

For L layer channel routing problem ($L \geq 5$), we can use the algorithm presented in [1] to obtain near optimal solutions. This algorithm uses $\lceil d/\lceil L/K \rceil \rceil$ tracks and runs in $O(T)$ time, when K is the minimum number of layers allowed between two overlapping wires.

V. EXPERIMENTAL RESULTS

We implemented our three-layer channel router in Pascal and ran our program on a Pyramid computer under UNIX 4.2BSD. Table I shows some of our experimental results. The examples labeled YK3a, YK3b, and YK3c are examples 3a, 3b, and 3c, respectively, in Yoshimura and Kuh [22]. The famous Deutsch's difficult example is labeled Diff. The examples labeled D1, D2, and D3 are from the GTE layout published in [19]. The third column of the table indicates the two-layer solution we started with and the number of tracks used in that two-layer solution.

For all the examples in Table I our results are optimal. In particular, we obtained an optimal ten-track three-layer solution for the well-known Deutsch's difficult example (Fig. 18) based on a 19-track two-layer solution [4]. Our track permutation and local rerouting techniques are very powerful, and in most of the cases we can complete our transformation without inserting an empty track. Also, the running time of our three-layer router is very short; for all the examples we tested, we obtained the results in less than 50 CPU seconds.

In general, our three-layer channel router performs better than other known three-layer channel routers. Table II shows a comparison with other routers in the commonly quoted examples. Our three-layer channel router consistently produced better solutions. C&L is Chen and Liu's three-layer channel router [5] based on Yoshimura and Kuh's net merging method for two-layer routing [22]. B&S is Bruell and Sun's three-layer "greedy" channel router [3]. "Chem" is the multilayer channel router Chameleon by Braun *et al.* [2]. E&D is the multilayer channel

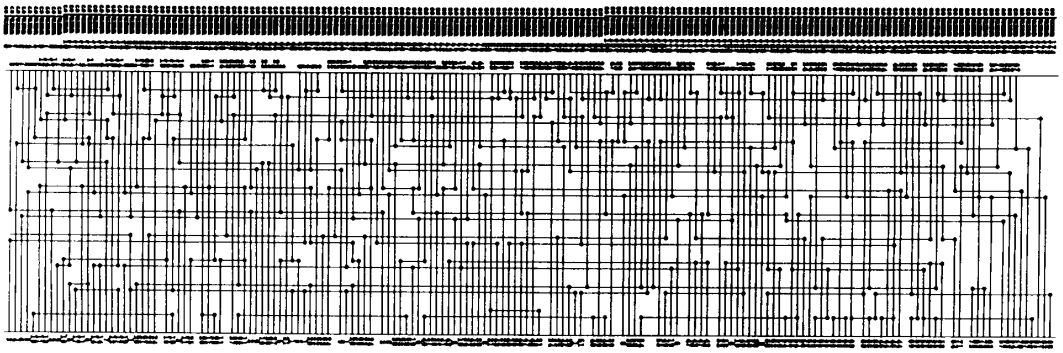


Fig. 18. Deutsch's difficult example.

TABLE II
COMPARISONS WITH OTHER THREE-LAYER CHANNEL ROUTERS

Ex.	d	C&L	B&S	Cham	E&D	Ours	Lower bound
YK3a	15	8	8	8	8	8	8
YK3b	17	10	10	10	10	9	9
YK3c	18	9	10	10	9	9	9
Diff.	19	14	11	11	13	10	10

router by Enbody and Du using limited backtracking [11]. For all test examples our algorithm never does worse than its competitors.

VI. Conclusion and Remarks

In this paper we developed a new approach to the three-layer channel routing problem based on the idea of transforming a two-layer solution into a three-layer solution. Our transformation consists of several steps, which can be formulated as two-processor scheduling, maze routing, and the shortest path problem, respectively. By using the best known algorithms for these well-studied optimization problems, we can solve each subproblem optimally in polynomial time. Indeed, our router performs very well on a variety of test examples and runs fast enough for any practical use. Most of the techniques can be generalized to four-layer channel routing.

Our maze router is quite effective in removing adjacent vias. Also, there is a certain degree of freedom in our algorithm for further refinement. For example, when there are several adjacent vias between two singular tracks, the removal of one via may block the rerouting for the elimination of other adjacent vias. In our implementation, we simply follow the natural order of appearance to remove the adjacent vias one by one. A possible refinement would be to set up some measurement of the difficulty of removing adjacent vias, and to eliminate adjacent vias according to the order of decreasing difficulties.

In this paper we showed an improved upper bound $\lceil d/2 \rceil + 2$ for four-layer channel routing, which differs from the lower bound $\lceil d/2 \rceil$ only by 2. It is worth mentioning that there is no satisfactory upper bound for the general three-layer channel routing problem. Any result along this line will be quite interesting.

References

[1] M. L. Brady and B. J. Brown, "Optimal multilayer channel routing with overlap," to be published.

[2] D. Braun *et al.*, "Chameleon: A new multi-layer channel router," in *Proc. 23rd Design Auto. Conf.*, 1986, pp. 495-502.

[3] P. Bruell and P. Sun, "A 'greedy' three layer channel router," in *Proc. ICCAD '85*, 1985, pp. 298-300.

[4] M. Burstein and R. Pelavin, "Hierarchical channel router," *Integration, J. VLSI*, vol. 1, pp. 21-38, 1983.

[5] Y. K. Chen and M. L. Liu, "Three layer channel routing," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, no. 2, pp. 156-163, 1984.

[6] E. G. Coffman, Jr. and R. L. Graham, "Optimal scheduling for two processor systems," *Acta Informat*, vol. 1, pp. 200-213, 1972.

[7] J. Cong, D. F. Wong, and C. L. Liu, "A new approach to the three layer channel routing," in *Proc. ICCAD '87.*

[8] J. Cong, "A new approach to three layer channel routing," M.S. thesis, Dept. Comp. Sci., Univ. of Illinois, Urbana, May 1987.

[9] D. N. Deutsch, "A dogleg channel router," in *Proc. 13th Des. Auto. Conf.*, 1976, pp. 425-433.

[10] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, p. 269-271, 1959.

[11] R. J. Enbody and H. C. Du, "Near-Optimal n-layer channel routing," in *Proc. 23th Design Auto. Conf.*, 1986, pp. 708-714.

[12] H. N. Gabow, "An almost linear time algorithm for two-processor scheduling," *J. Ass. Comput. Mach.*, vol. 29, no. 3, pp. 766-780, 1982.

[13] N. H. Gabow, "A linear time algorithm for a special case of disjoint set union," *J. Comput. Syst. Sci.*, vol. 30, pp. 223-225, 1985.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: Freeman, 1979.

[15] S. E. Hambrusch, "Channel routing algorithms for overlap models," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 23-30, Jan. 1985.

[16] A. Hashimoto and S. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 8th Design Auto. Workshop*, 1971, pp. 155-169.

[17] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 8th Design Auto. Workshop*, 1976, pp. 155-169.

[18] C. Y. Lee, "An algorithm for path connection and its application," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 346-365, 1961.

[19] D. F. Wong and C. L. Liu, "Compacted channel routing with via placement restriction," *Integration, J. VLSI*, vol. 4, pp. 267-307, 1986.

[20] J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro, "A new symbolic channel router: YACR2," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 208-219, 1985.

[21] R. L. Rivest and C. M. Fiduccia, "A 'greedy' channel router," in *Proc. 19th Design Auto. Conf.*, pp. 418-424, 1982.

[22] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 25-35, Jan. 1982.

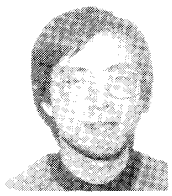
[23] D. N. Deutsch, private communication, 1987.

Jingsheng (Jason) Cong received the B.S. degree in computer science from Peking University in Peking, China, in 1985, and the M.S. degree in computer science from the University of Illinois at Urbana-Champaign in 1987. Currently, he is a Ph.D. candidate in computer science at the University of Illinois, Urbana-Champaign.

Since 1986 he has been a Research Assistant in Computer Science Department of the University of Illinois. He worked at the Xerox Palo Alto Research Center during the summer of 1987. His research interests include design and analysis of efficient combinatorial and geometric algorithms, computer-aided design of integrated circuits, and parallel algorithms.

Mr. Cong received the Best Graduate Award from Peking University in 1985. He is a member of the Association of Computing Machinery.

*



D. F. Wong received the B.Sc. degree in mathematics from the University of Toronto, Toronto, Ontario, Canada, and the M.S. degree in mathematics from the University of Illinois at Urbana-Champaign. He obtained the Ph.D. degree in computer science from the University of Illinois, Urbana-Champaign, in Jan. 1987.

He is currently an Assistant Professor of Computer Science at the University of Texas at Austin. His current research interests are in computer-aided design of integrated circuits and design and analysis of algorithms.

Dr. Wong received the Best Paper Award in the physical design category at the 1986 IEEE-ACM Design Automation Conference.

*

C. L. Liu (M'64-M'78-SM'82-F'86) obtained the B.Sc. degree at Cheng Kung University in Taiwan in 1956. He obtained the M.S. and E.E. degrees in 1980 and the Sc.D. degree in 1962, all in electrical engineering, at the Massachusetts Institute of Technology, Cambridge.

He is currently a Professor of Computer Science at the University of Illinois at Urbana-Champaign. His areas of research interest are design and analysis of algorithms, computer aided design of integrated circuits, and combinatorial mathematics.

Dr. Liu is a Fellow of the Institute of Electrical and Electronics Engineers. He currently serves on the editorial board of *IEEE Transactions on Electronic Computers, Information Sciences, Graphs & Combinatorics*, and *Algorithmica*, and is also an Editorial Advisor to the Computer Science Series published by the World Scientific Publishing Company.