Bubble-sort approach to channel routing

S.-S.Chen, C.-H.Yang and S.-J.Chen

Abstract: An efficient bubble-sort technique for solving the two-layer non-Manhattan channel-routing problem is presented. The time and space complexities of our algorithm are O(kn) and O(n), respectively, where k is the number of sorting passes required and n is the total number of two-terminal nets in a routing channel. The algorithm is easily extended to handle the cases with multiterminal nets distributed in a channel. Various tests verify the efficiency of the bubble-sort based router. Experimental results indicate that the router is time-efficient for routing. A three-layer algorithm having O(kn) time based on an identical problem formulation is proposed for solving the non-Manhattan channel routing.

1 Introduction

Channel routing plays an important role in minimising the routing area at the physical design level of VLSI circuits. In the last two decades various channel routing results have been reported based on the Manhattan model [1–7], which restricts the routing wires to be either vertical or horizontal. An example of the Manhattan routing is shown in Fig. 1, which solution needs some extra columns. Nowadays, the VLSI fabrication process does not preclude a layout style to be in a non-Manhattan routing model. As shown in Fig. 2, the vertical constraints which occurred in the Manhattan routing model no longer exist in the non-Manhattan case.

The first non-Manhattan diagonal channel router, introduced by Lodi, *et al.* [8], realises a layout of two-terminal nets on a two-layer channel using a diagonal channel routing model (DCRM), where only wires of 45° and −45° can be used. Later, Wang and Kuh [9, 10] proposed a mini-swap sorting method to solve the non-Manhattan channel routing problem. Fig. 3 shows a four-track solution for the example of Fig. 1 using Wang's mini-swap router.

A heuristic algorithm for the non-Manhattan channel routing based on the bubble-sort technique has been proposed by Chaudhary and Robinson [11]. The basic concept of the algorithm was derived from the inversion table introduced by Knuth [12]. Two types of inversion tables were used in Chaudhary's algorithm: left and right inversion tables. The criteria of sorting direction depends on the numbers of nonzero elements in the left and the right inversion tables. The sorting direction is from left to right (*right-step*) if the number of nonzero entries in the left inversion table is greater than that in the right inversion table and *vice versa*. But the algorithm hides a shortcoming, the sorting direction is set as from right to left

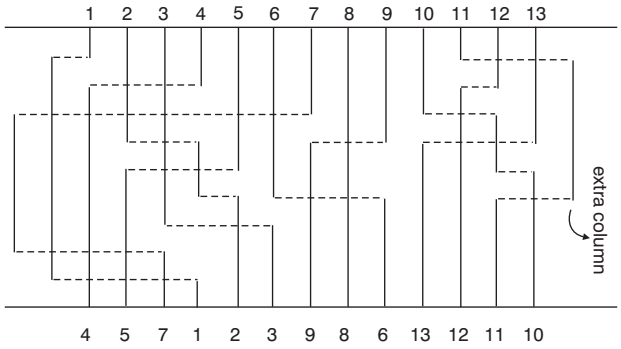

Fig. 1 *Nine-track channel routing using Manhattan model*

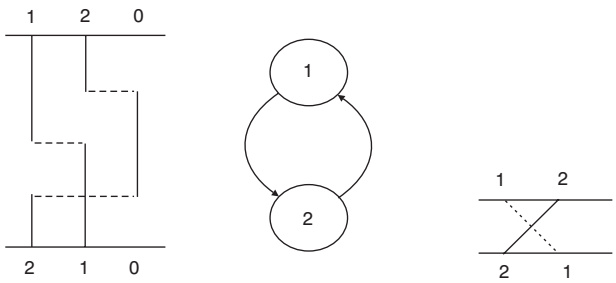

Fig. 2 *Vertical constraints to not list in non-manhattan model*

a Manhattan model
b Cyclic constraint graph
c Diagonal model

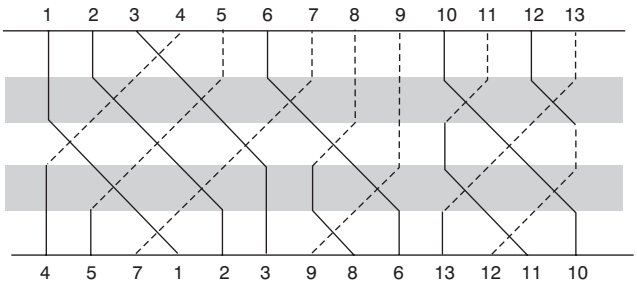

Fig. 3 *Four-track solution for Fig. 1 using Wang's mini-swap model*

IEE Proceedings online no. 20000810

DOI: 10.1049/ip-cdt:20000810

Paper first received 1st July 1999 and in revised form 28th July 2000

S.-S. Chen and S.-J. Chen are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

C.-H. Yang is with the Department of Information Management, Kun Shan University of Technology, Tainan, Taiwan, Republic of China

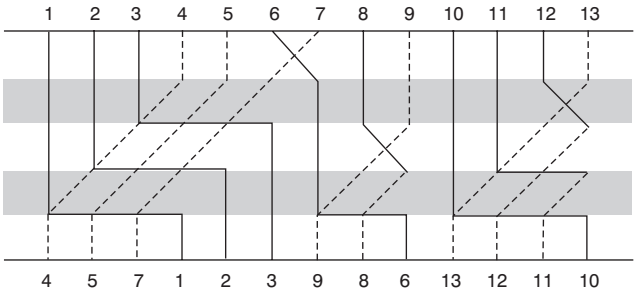

Fig. 4 *Four-track solution for Fig. 1 using Chaudhary's routing model*

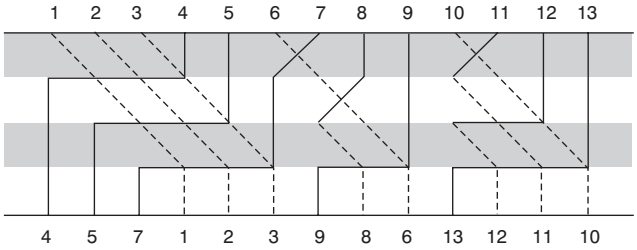

Fig. 5 *Optimal three-track solution is obtained using our approach*

(*left-step*) when the numbers of nonzero entries in the left and in the right inversion tables are equal. Therefore it cannot guarantee an optimal solution in terms of routing tracks required. Intuitively, the example shown in Fig. 4 is nonoptimal because a truly optimal solution needs only three routing tracks. In addition, the time and space complexities of the algorithm are $O(kn^2)$ and $O(n)$, respectively, where k is the number of sorting passes required and n is the total number of two-terminal nets in a routing channel.

Recently, an optimal bubble-sort algorithm in terms of routing tracks for the non-Manhattan channel routing has been proposed by Chen, *et al.* [13]. The optimal sorting sequence is generated by applying a *left-step* sorting and a *right-step* sorting to the nonequivalent sorting vectors in each pass of the bubble-sort. The procedure is iterated until some completely sorted vectors are found. This kind of enumerative algorithm can achieve an optimal solution with minimum number of sorting passes required. But the time and space complexities of the algorithm are not optimal because they still preserve an order of $O(k^2n)$ and $O(kn)$, respectively. We present in this paper a bubble-sort based algorithm under the same problem formulation as in [9, 11, 13] and this algorithm can be extended easily to handle the cases of multiterminal nets. The time and space complexities of our algorithm are $O(kn)$ and $O(n)$, respectively. Obviously, our algorithm has a significant improvement in time and space complexities over the existing algorithms [11, 13]. An optimal routing solution for the example in Fig. 1 using our algorithm is shown in Fig. 5.

2 Problem formulation

Our research on non-Manhattan channel routing carries on with previous literature, such as Wang [9], Chaudhary [11], and Chen [13]. A non-Manhattan router has the following advantages over its Manhattan counterpart: The number of routing tracks and vias required is less than the Manhattan router (see Fig. 8 in Wang [9] and Fig. 1 in Chen [13]); the vertical constraints are not necessary; the total wire length is shorter than the Manhattan router; and no need of using

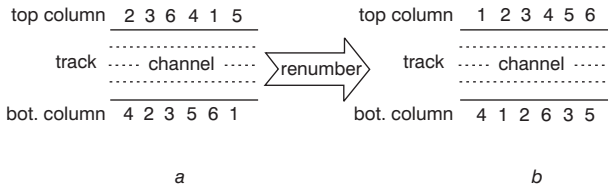

Fig. 6 *Two identical channel routing problems*

a Original channel
b Renumbered channel

extra columns outside the width of a channel to have the routing task done. For simplicity and making the concept clear, we assume that each net has two terminals over a channel, one terminal is from the top column vector and the other from the bottom column vector. The handling of multiterminal nets is described later. At first the terminals located in these column vectors are randomly labelled as integers. But we renumber the terminal labels in the top column vector to be 1 through n and corresponding changes are made to the bottom terminal labels. For instance, assume that the top column vector = (2, 3, 6, 4, 1, 5) and the bottom column vector = (4, 2, 3, 5, 6, 1). If the top column vector is renumbered as (1, 2, 3, 4, 5, 6), the bottom column vector will be changed to (4, 1, 2, 6, 3, 5), as shown in Fig. 6.

The objective of channel routing is to interconnect nets having the same terminal labels and to minimise the number of routing tracks required, the time, and the space complexities as much as possible. In this paper we apply the bubble-sort technique to accomplish channel routing. Before describing our algorithm for the non-Manhattan channel routing, we first define some terminologies as follows.

Definition 1: Left major table. $(b_1, b_2, \ldots, b_n)$ is said to be the *left major table* of a permutation $V = (v_1, v_2, \ldots, v_n)$ if b_{v_j} is set to 1, when there exists in V at least one element to the left of v_j and greater than v_j, for $j = 1$ to n and $v_j \in V$; otherwise, b_{v_j} is set to 0.

For example, given a permutation $V = (5, 1, 9, 2, 8, 6, 4, 7, 3)$, we have the following *left major table* (1, 1, 1, 1, 0, 1, 1, 1, 0).

Definition 2: Right minor table. $(c_1, c_2, \ldots, c_n)$ is said to be the *right minor table* of a permutation $V = (v_1, v_2, \ldots, v_n)$ if c_{v_j} is set to 1, when there exists in V at least one element to the right of v_j and less than v_j, for $j = n$ to 1 and $v_j \in V$; otherwise, c_{v_j} is set to 0.

For example, given the permutation $V = (5, 1, 9, 2, 8, 6, 4, 7, 3)$ we have the following *right minor table* (0, 0, 0, 1, 1, 1, 1, 1, 1).

Definition 3: Lexicographical ordering. Given two tables $B = (b_1, b_2, \ldots, b_n)$ and $C = (c_1, c_2, \ldots, c_n)$ which belong to a *left major table* and a *right minor table* or *vice versa*. Their *lexicographical order* is determined as follows:

(i) Initially, let j be equal to 1.
(ii) If $b_j > c_j$ (or $c_j > b_j$) we say that B is greater (less) than C and can immediately terminate the comparison between B and C.
(iii) Otherwise, if $b_j = c_j$ we increase the index j by 1 and continue steps 2 and 3 to compare the next pair of elements in the tables until $j = n$. Besides, we say $B = C$ if $b_j = c_j$ for all $j = 1$ to n.

For instance, given two tables $B = (1, 0, 1, 1, 0, 1, 1, 1, 0)$ and $C = (1, 0, 1, 0, 0, 1, 1, 1, 1)$, we say that table B is greater than table C.

3 Two-layer non-Manhattan channel routing

3.1 Non-Manhattan routing model

Traditionally, the most common channel routing model, known as a Manhattan model, uses vertical and horizontal wires for routing (also known as an H–V routing model) on two or more layers over a channel. In a non-Manhattan channel routing model three wiring types are used: diagonal (run at 45° or −45° direction), horizontal and vertical wires as shown in Fig. 7. The routing model we use is a non-Manhattan model with two available layers for the routing of two-terminal nets over a channel. We reserve one layer for metal-1 wires and the other layer for metal-2 wires and a contact cut is required while making connection between these two layers.

Chaudhary [11] made an interesting observation about the bubble-sort as follows. Given a vector $V = (v_1, v_2, \ldots, v_n)$, if the sorting direction for V either traversing from left to right (denoted as *right-step*) or from right to left (denoted as *left-step*) in each bubble sorting pass is fixed, it will yield worse results in terms of total number of routing tracks in a channel. Intuitively, one can obtain an optimal result if the sorting direction has been properly determined such that elements in V can be moved more steps (distances) in each pass of the bubble-sort. The effects of using different types of sorting directions on the final routing tracks are shown in Fig. 7. Therefore we have to further explain in the following Section how to determine the sorting direction by using the *left major* and the *right minor tables* in each of the sorting passes.

3.2 Algorithm for bubble-sort router

We propose an efficient bubble-sort algorithm for the non-Manhattan channel routing. First, assume that the unsorted and bottom column-vector is the input sorting vector. The algorithm has two important phases. The purpose of first phase is to generate both the *left major table* and the *right minor table* for a sorting vector. Let the sorting vector $V = (v_1, v_2, \ldots, v_n)$ be a permutation of the natural sequence $(1, 2, \ldots, n)$. Entries in the *left major table* and the *right minor table* are generated by applying the following lemma.

Lemma 1: For a sorting vector $V = (v_1, v_2, \ldots, v_n)$, the *left major table* and the *right minor table* of V in each of sorting passes can be constructed in $T_a(n) = O(n)$ time each, where n is the number of elements in V.

Proof: Let $(b_1, b_2, \ldots, b_n)$ and $(c_1, c_2, \ldots, c_n)$ be the *left major table* and the *right minor table* of this vector V, respectively. These two tables are constructed as follows. If $v_j > Max\_val$, then set b_{v_j} to a value of zero and replace Max_val with v_j. Otherwise set b_{v_j} to a value of one; for $j = 1, 2, \ldots, n$ (from left to right). Obviously, the *left major table* construction takes $O(n)$ time. Similarly, if $v_j < Min\_val$, then set c_{v_j} to a value of zero and replace Min_val with v_j. Otherwise set c_{v_j} to a value of one; for $j = n, n-1, \ldots, 2, 1$ (from right to left). This *right minor table* construction also takes $O(n)$ time. Here the initial values of Max_val and Min_val are zero and a large positive number, respectively. □

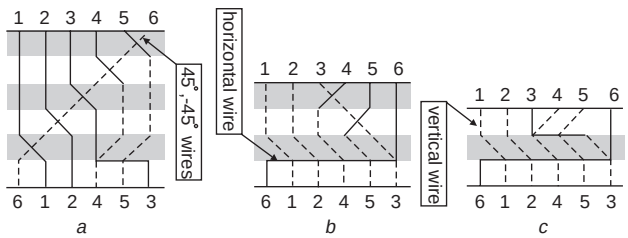

Fig. 7 *Effects of different sorting directions*

a Left-step only
b Right-step only
c Left- & *right-steps*.

The second phase purpose is to properly choose a sorting direction for each pass of the bubble-sort, since a good sorting direction will reduce the number of sorting passes required by the bubble-sort. From the previous phase we have constructed two sets of tables: the *left major table* and the *right minor table*. We choose the sorting direction to be *left-step* (respectively, *right-step*) if the number of nonzero entries in the *left major table* is less (respectively, greater) than that in the *right minor table*. If the numbers of nonzero entries in the *left major table* and in the *right minor table* are equal, we must compare their *lexicographical order*. We choose the sorting direction to be *right-step* if the *left major table* is greater than the *right minor table* referring to the lexicographical ordering definition in Section 2; and *left-step* otherwise. Once the sorting direction for the sorting vector is determined a pass of bubble-sort is performed. These phases are repeated until the completely sorted vector is found.

To make the principle of our algorithm clear we describe the sorting passes of a non-Manhattan channel routing as follows. Given an example with a top column vector = (1, 2, 3, 4, 5, 6, 7, 8, 9) and a bottom column vector = (2, 3, 9, 4, 5, 6, 7, 8, 1). The bubble sorting process begins with the bottom sorting vector. At each intermediate step of the sorting, a new sorting vector is generated as the result of a permutation of the natural sequence $(1, 2, \ldots, n)$ and this vector will require one routing track in a channel. Initially, the *left major table* and the *right minor table* can be constructed by scanning the input sorting vector. For this example, we have the first *left major table* and the first *right minor table* as (1, 0, 0, 1, 1, 1, 1, 1, 0) and (0, 1, 1, 1, 1, 1, 1, 1, 1), respectively. Obviously, the numbers of nonzero entries in the *left major table* and the *right minor table* are 6 and 8, respectively. Hence the sorting direction will be chosen as *left-step*. After having performed this *left-step* bubble-sort, a new sorting vector is produced as (1, 2, 3, 9, 4, 5, 6, 7, 8). Similarly, in the second pass, the *left major table* and the *right minor table* constructed are (0, 0, 0, 1, 1, 1, 1, 1, 0) and (0, 0, 0, 0, 0, 0, 0, 0, 1), respectively. As the numbers of nonzero entries in the *left major table* and the *right minor table* are 5 and 1, respectively, the sorting direction will be chosen as *right-step* and the current sorting vector is changed to (1, 2, 3, 4, 5, 6, 7, 8, 9) which is already in order. Therefore two passes of bubble-sort are required to complete the sorting work. The illustration for the example is shown in Fig. 8 and its corresponding non-Manhattan channel routing solution is plotted in Fig. 9. The description of our bubble-sort router is depicted in algorithm 1.

Theorem 1: The time and space complexities of using our bubble-sort algorithm to route a two-layer non-Manhattan channel take $T_b(n) = O(kn)$ time and $O(n)$ space, respectively, where k is the number of sorting passes required and n is the number of two-terminal nets in a channel.

Proof: In each of the bubble-sorting passes the sorting task for a sorting vector $V = (v_1, v_2, \ldots, v_n)$ will be performed by choosing a *left-step* or a *right-step* direction, which runs in $O(n)$ time. Referring to lemma 1, we need $T_a(n) = O(n)$ time to generate a table in each sorting pass. Assume that the sorting task is completely sorted after the kth bubble-

Pass no.	Sorting vector Left major table	*Max_val*	Sorting vector Right minor table	*Min_val*
1	①②③④⑤⑥⑦⑧⑨ (***2*** 3 9 4 5 6 7 8 ***1***)	2	①②③④⑤⑥⑦⑧⑨ (2 3 9 4 5 6 7 8 ***1***)	1
	(■ 0 ■ ■ ■ ■ ■ ■ ■)		(0 ■ ■ ■ ■ ■ ■ ■ ■)	
	(***2 3*** 9 4 5 6 7 8 1)	3	(2 3 9 4 5 6 7 ***8 1***)	1
	(■ 0 0 ■ ■ ■ ■ ■)		(0 ■ ■ ■ ■ ■ ■ 1 ■)	
	(***2 3 9*** 4 5 6 7 8 1)	9	(2 3 9 4 5 6 ***7 8 1***)	1
	(■ 0 0 ■ ■ ■ ■ ■ 0)		(0 ■ ■ ■ ■ ■ 1 1 ■)	
	the operation continues . . .			
	(***2 3 9 4 5 6 7 8*** 1)	9	(2 ***3 9 4 5 6 7 8 1***)	1
	(■ 0 0 1 1 1 1 1 1 0)		(0 ■ 1 1 1 1 1 1 1)	
	(***2 3 9 4 5 6 7 8 1***)	9	(***2 3 9 4 5 6 7 8 1***)	1
	(1 0 0 1 1 1 1 1 0)		(0 1 1 1 1 1 1 1 1)	
	New sorting vector (1 2 3 9 4 5 6 7 8) is generated by applying a pass of *left-step* bubble-sort to current sorting vector.			
2	(***1 2*** 3 9 4 5 6 7 8)	1	(1 2 3 9 4 5 6 7 ***8***)	8
	(0 ■ ■ ■ ■ ■ ■ ■ ■)		(■ ■ ■ ■ ■ ■ ■ 0 ■)	
	the operation continues . . .			
	(***1 2 3 9 4 5 6 7 8***)	9	(1 ***2 3 9 4 5 6 7 8***)	2
	(0 0 0 1 1 1 1 ■ 0)		(■ 0 0 0 0 0 0 0 0 1)	
	(***1 2 3 9 4 5 6 7 8***)	9	(***1 2 3 9 4 5 6 7 8***)	1
	(0 0 0 1 1 1 1 1 0)		(0 0 0 0 0 0 0 0 1)	
	Numbers of nonzero entries in *left major* and *right minor tables* are 5 and 1, respectively. Thus a pass of *right-step* bubble-sort is applied to this sorting vector and then the completely sorted vector is found at pass two.			

Fig. 8 *Example to illustrate sorting process of our algorithm*

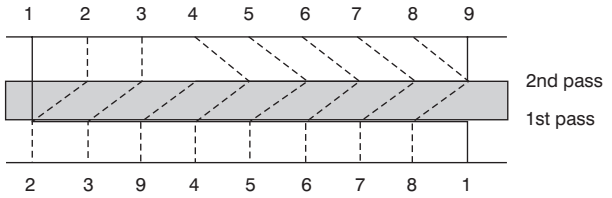

Fig. 9 *Non-Manhattan channel routing solution corresponding to Fig. 8*

sorting pass. As a result the overall time complexity $T_b(n)$ of our algorithm is formulated as

$$T_b(n) = \sum_{i=1}^{k}[n + 2T_a(n)] = \sum_{i=1}^{k}[n + 2n]$$
$$= 3\sum_{i=1}^{k} n = 3kn \cong O(kn)$$

In the worst case the time complexity takes $O(n^2)$ when all elements in a sorting vector V are in reverse order and then k is asymptotic to n. On the space complexity side, for storing the information needed in sorting vectors, the *left major table* and the *right minor table* used in all of the sorting passes need n space each. Obviously, the total space complexity of our algorithm is preserved in $O(n)$ because the required space for the sorting vectors and tables is reusable in each sorting pass of the bubble-sort. □

3.3 Handling of multiple terminal nets

In general, multiterminal nets usually exist in a routing channel. We first transform these multiterminal nets into sets of two-terminal nets before the bubble-sort router. For simplicity, we classify these multiterminal nets into type-I, type-II and type-III nets. A net is called type-I net if their terminals are distributed only on the upper or lower side of a channel. For example, the terminals (b_1, b_2) of net b in Fig. 10*a* can be directly routed together and be dropped from consideration in later sorting phase. A type-II m-terminals net has m-1 terminals ($m \geq 2$) on one side and only one terminal on the opposite side of a channel. If the m-1 terminals are on the upper side, we first connect them together and then one of the terminals on this side and the only terminal on the lower side are chosen to form a two-terminal net to be routed in a later sorting phase.

Algorithm 1: Two-layer bubble-sort channel router

```
Algorithm TwoLayer_BubbleSortRouter()
{
  PASS = 0;
  Sorting_Vector = Input_Vector ;
  write( Sorting_Vector);
  while( unsorted for Sorting_Vector ) {
      Compute_Tables( Sorting_Vector );
      Set_Step_Type( Sorting_Direction );
      switch( Sorting_Direction ) {
          case L_to_R : /* the sorting direction is right-step */
                  for(j = 1; j < n; j ++)
                    if( Sorting_Vector[j] > Sorting_Vector[j + 1])
                        swap( Sorting_Vector[j], Sorting_Vector[j + 1]);
                  break;
          case R_to_L : /* the sorting direction is left-step */
                  for( j = n; j > 1; j--)
                    if ( Sorting_Vector[j] < Sorting_Vector[j − 1])
                        swap( Sorting_Vector[j], Sorting_Vector[j − 1]);
       }
      write( Sorting_Vector ); /* output the sorting vector to a file */
      PASS ++; /* increase the number of sorting passes by one*/
  }
} /* end of the TwoLayer_BubbleSortRouter() Algorithm */


Compute_Tables( Sorting_Vector )
{ /* compute the left major table and the right minor table */
  Initialize Left_table, Right_table;
  Left_val = Right_val = 0;
  Max_val = 0;
  for( j = 1; j <= n; j ++) /* construct the left major table */
      if( Sorting_Vector[j] > Max_val)
          Left_table[ Sorting_Vector[j]] = 0;
          Max_val = Sorting_Vector[j];
      else
          Left_table[ Sorting_Vector[j]] = 1;
          Left_val ++;
  Min_val = ∞;
  for(j = n; j >= l; j--) /* construct the right minor table */
      if ( Sorting_Vector[j] < Min_val)
          Right_table[ Sorting_Vector[j]] = 0;
          Min_val = Sorting_Vector[j];
      else
          Right_table[ Sorting_Vector[j]] = 1;
          Right_val ++;
} /* end of the subroutine */

Set_Step_Type(Sorting_Direction)
{ /* decide the sorting direction for a sorting vector */
  switch(Left_val - Right_val) {
      case > 0 : Sorting_Direction = L_to_R;
                 break;
      case = 0 : if (Left_table > Right_table in terms of lexicographical order)
                   Sorting_Direction = L_to_R;
                 else  Sorting_Direction = R_to_L;
                 break;
      case < 0 : Sorting_Direction = R_to_L;
    }
}/* end of the subroutine */
```

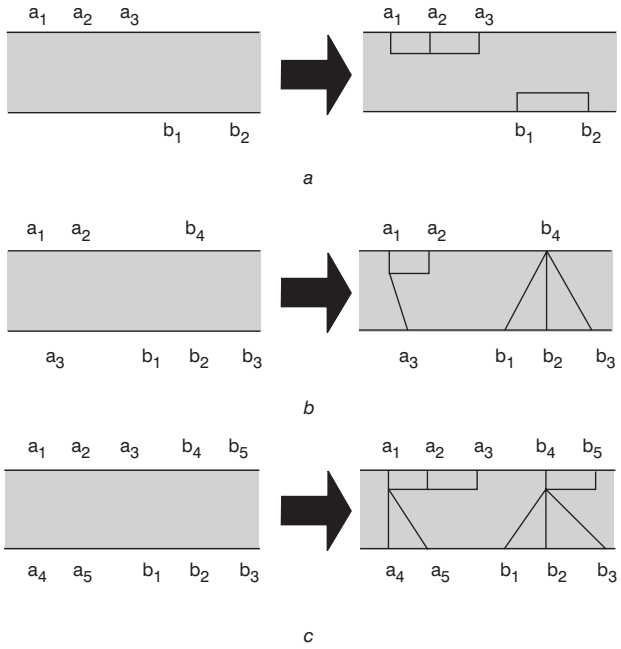

Fig. 10 *Classification of multiterminal nets in bubble-sort router*

a Type-I nets
b Type-II nets
c Type-III nets

Similarly, if the m-1 terminals are located on the lower side, they will be connected to the upper terminal at a single column in later sorting phase. This type-II case is plotted in Fig. 10*b*. A net is called type-III net if there are some x terminals distributed on the upper side and some (m-x) terminals on the lower side of a channel. This case is illustrated in Fig. 10*c*. For the upper x terminals, we first connect them together and choose a suitable terminal as the final representative terminal. Then this upper representative terminal is connected to all (m-x) terminals on the lower side as in the type-II case. After this process, all multiterminal nets can be transformed into two-terminal nets, and we can then apply the same bubble-sort router to complete the routing of multiterminal nets.

4 Three-layer non-Manhattan channel routing

Advances in VLSI technology mean that more than two layers are available for routing. We now extend the bubble-sort router to solve the same routing problem over a three-layer channel.

During three-layer routing, our two-layer bubble-sort router (algorithm 1) can first be invoked to generate a sorting sequence S for an unsorted vector $V=(v_1, v_2, \ldots, v_n)$. Each element in the sorting sequence is either a *left-step (L)* or a *right-step (R)*. The sorted vector can be represented as $SV=A_kA_{k-1}\ldots A_2A_1V$, where A_i is either a *left-step* pass or a *right-step* pass of bubble-sort and $1 \le i \le k$. Now we scan the S from right to left until a pass of *left-step* bubble-sort, A_j, is found with index j in S. In other words,

$$S = A_kA_{k-1}\ldots A_{j+1}L\underbrace{RR\ldots RR}_{j-1}.$$

Again, by repeatedly applying the theorem $LRV=RLV$ in [13], we have

$$SV = A_kA_{k-1}\ldots A_{j+1}L\underbrace{RR\ldots RR}_{j-1}V = \ldots$$
$$= A_kA_{k-1}\ldots A_{j+1}R\underbrace{R\ldots RRL}_{j-1}V.$$

This process is iterated for the finding of *left-steps* until all *left-step* sorting passes in S are moved to the right side of the sorting sequence S. Finally, the sorting sequence is represented as follows:

$$SV = A_kL\underbrace{RR\ldots R}_{c-1}LL\ldots LV$$
$$= \underbrace{RRR\ldots R}_{c}\underbrace{LLL\ldots L}_{k-c}V = R^cL^{k-c}V$$

where k is the number of sorting passes required for V and c is a constant, $0 \le c \le k$.

Lemma 2: The completely sorted sequence $S=R^cL^{k-c}$ of a sorting vector V generated by our bubble-sort router can be adjusted to be R (*right-step*) and L (*left-step*) alternatively in $T_c(n)=O(k)$ time, where k is the number of sorting passes required for V and $0 \le c \le k$.

Proof: Consider a completely sorted sequence $S=R^cL^{k-c}$ of a sorting vector V. Using the theorem $LRV=RLV$ as proved in [13], the sorting sequences of LR and RL in a bubble-sort solution to V are interchangeable. That is, the sorting result only depends on the number of *left-step* (L) or *right-step* (R) passes rather than their permutation positions in a bubble-sort solution. As a result,

$$SV = R^cL^{k-c}V = (L^x|R^x)\underbrace{RLRL\ldots RL}V$$

is obtained in $O(k)$ time, where $0 \le x \le k$. Here, the symbol $(L^x|R^x)$ represents the extra L^x or R^x when the numbers of Ls and Rs are not equal. □

Since three layers are available for routing in a channel, if the routings of a pass of R and a pass of L bubble-sort occupy different horizontal layers, they can be integrated into a single track. Therefore the number of routing tracks for an unsorted vector V is equal to the value of maximum $\{c, k-c\}$.

Following closely the foregoing discussion, the sorting result has been generated as a series of intermediate permutations. The criteria presented in [11] will be used for layer assignment of wire segments as follows. In an intermediate permutation the wire segments which have been moved toward to the right are assigned to layer 1, those moved toward the vertical are assigned to layer 2, and those moved toward the left are assigned to layer 3. This characteristic of layer assignment is similar to the conventional HVH Manhattan routing model. Finally, the routing is done by integrating a pair of adjacent intermediate permutations into one routing track in a channel. An outline of the three-layer routing algorithm is depicted in algorithm II. For example, given an unsorted vector

Algorithm II: Three-layer bubble-sort channel router

```
Algorithm ThreeLayer_BubbleSortRouter()
{
  /* k is the number of sorting passes */
  V = Input_Vector;
  Invoke TwoLayer_BubbleSortRouter() presented in Algorithm I to generate a
  sorting sequence, S for the unsorted vector V;
  Represent the completely sorted vector as SV = R^c L^{k-c} V, 0 ≤ c ≤ k;
  Adjust the sorted vector R^c L^{k-c} V into (L^x|R^x) RLRL...RL V such that any two
  adjacent sorting directions appear to be R and L alternatively, where 0 ≤ x ≤ k;
  Generate the sorting result as a series of intermediate permutations;
  while (routing is incomplete) {
      Wire segments jogging to the right, vertical, and left directions are assigned
      to layers 1, 2, and 3 in a channel, respectively;   /* layer assignment */
      Integrate a pair of adjacent intermediate permutations into a routing track;
    }
} /* end of the ThreeLayer_BubbleSortRouter() Algorithm */
```

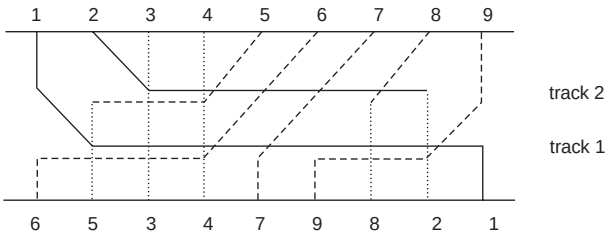

Fig. 11 *Example of three-layer non-Manhattan channel routing solution using our bubble-sort router*

$V = (6, 5, 3, 4, 7, 9, 8, 2, 1)$. A three-layer non-Manhattan channel routing solution $SV = RRLLV = RLRLV$ using our bubble-sort router is shown in Fig. 11.

Theorem 2: The time complexity $T_d(n)$ of using our bubble-sort algorithm to route a three-layer non-Manhattan channel takes $O(kn)$ time, where k and n stand for the number of sorting passes required and the number of two-terminal nets in a channel, respectively.

Proof: The time complexity of our three-layer bubble-sort algorithm is calculated as follows. Consider a completely sorted vector $R^cL^{k-c}V$ obtained by our two-layer bubble-sort based algorithm takes $T_b(n)$ time, where $V = (v_1, v_2, \ldots, v_n)$. By *lemma* 2, to adjust the completely sorted sequence R^cL^{k-c} such that any two adjacent sorting directions appear to be R and L alternatively takes $T_c(n)$ time. Formally stated, the overall time complexity $T_d(n)$ of our three-layer algorithm is

$$T_d(n) = T_b(n) + T_c(n) = O(kn) + O(k) \cong O(kn)$$

As a result, the time complexity is bounded in $O(kn)$ on the average. In the worst case, a time complexity $O(n^2)$ is obtained when all elements in a sorting vector V are in reverse order and k is asymptotic to n. □

5 Experimental results

Our bubble-sort channel router, Chaudhary's [11] router, and Chen's [13] router have been implemented in C language and tested on a Sun Ultra-SPARC-1 workstation running the Solaris 2.6 platform. Furthermore, we have conducted many testing examples collected from the literature [9, 11, 13] and some cases originated by ourselves to evaluate the effectiveness and correctness of our router. The experimental results are compared and analysed as follows.

First, the results of running different routers on several benchmarks are shown in Table 1. From the testing results of Lodi [8] in Table 1, many situations of 'no solution' occur under the constraint of no allowance of using extra column(s) in a channel. Next, the comparisons of complexity among two-layer algorithms are listed in Table 2. In Table 2 the time complexity of the diagonal router proposed in [8] runs $O(n)$. But it fails to complete the routing without magnifying the row spacing by $\sqrt{2}$ and without using extra column(s) in a channel for most of the cases. We next compare the experimental results with the mini-swap router [9] which has a time and a space complexities of $O(kn)$ and $O(n)$, respectively. But it could not generate optimal solutions in terms of routing tracks required. We also run the same instances using Chaud-

Table 1: Experimental results on running some benchmarks

Benchmarks	Order of nets terminals	nets	Routing tracks for different non-Manhattan routers Lodi [8]	Wang [9]	Chaudhary [11]	Chen [13]	ours
Circuit 1 Fig. 8 in [9]	14	7	no solution	3	3	3	3
Circuit 2 Fig. 3 in [13]	16	8	no solution	6	5	5	5
Circuit 3 Fig. 7 in [13]	16	8	6	6	4	4	4
Circuit 4 Fig. 5 in [11]	18	9	no solution	8	2	2	2
Circuit 5 Fig. 5 [our]	26	13	no solution	4	4	3	3
Circuit 6 Fig. 5 in [13]	32	16	no solution	5	5	3	3
Circuit 7 Fig. 9 in [9]	34	17	no solution	7	5	5	5

Table 2: Comparisons for two-layer routing algorithms

Complexity / Algorithms	Time complexity average	Time complexity worst	Space complexity average	Space complexity worst	Bubble sort approach?
Lodi [8]	$O(n)$	no solution	need extra columns		*NO*
Wang [9]	$O(kn)$	$O(n^2)$	$O(n)$	$O(n)$	*NO*
Chaudhary [11]	$O(kn^2)$	$O(n^3)$	$O(n)$	$O(n)$	*YES*
Chen [13]	$O(k^2n)$	$O(n^3)$	$O(kn)$	$O(n^2)$	*YES*
Our Router	$O(kn)$	$O(n^2)$	$O(n)$	$O(n)$	*YES*

Table 3: Comparison with other routers for randomly generated examples

Ex no.	300-net examples Chaudhary [11] tracks	time	Chen [13] tracks	time	Ours tracks	time	Ex no.	375-net examples Chaudhary [11] tracks	time	Chen [13] tracks	time	Ours tracks	time
1	142	3.47	142	1.37	142	0.07	1	185	7.02	184	2.82	185	0.10
2	157	3.83	156	1.63	157	0.08	2	197	7.48	195	3.17	195	0.13
3	162	3.97	162	1.75	162	0.08	3	203	7.70	200	3.32	203	0.12
4	157	3.83	155	1.62	156	0.07	4	185	7.02	185	2.85	185	0.10
5	150	3.65	147	1.45	150	0.07	5	177	6.70	176	2.60	177	0.10
avg	**153**	**3.75**	**152**	**1.56**	**153**	**0.12**	**avg**	**189**	**7.18**	**188**	**2.95**	**189**	**0.20**
	450-net examples							525-net examples					
1	228	12.43	228	5.07	228	0.15	1	264	19.48	257	7.50	262	0.20
2	240	13.10	239	5.62	240	0.17	2	266	19.67	266	8.05	266	0.20
3	238	12.95	236	5.45	238	0.15	3	272	20.08	270	8.37	272	0.20
4	236	12.80	235	5.38	236	0.17	4	266	19.70	265	8.02	266	0.20
5	236	12.82	235	5.45	236	0.15	5	268	19.77	268	8.20	268	0.20
avg	**235**	**12.82**	**234**	**5.39**	**235**	**0.28**	**avg**	**267**	**19.74**	**265**	**8.03**	**266**	**0.32**
	600-net examples							675-net examples					
1	298	28.72	295	11.38	297	0.25	1	334	40.75	334	16.33	334	0.32
2	311	30.08	311	12.58	311	0.27	2	338	41.18	337	16.67	338	0.32
3	314	30.33	313	12.80	314	0.28	3	332	40.57	332	16.08	332	0.32
4	308	29.80	306	12.25	308	0.25	4	370	45.10	368	19.70	370	0.35
5	297	28.62	296	11.45	297	0.25	5	355	43.22	351	17.88	355	0.33
avg	**305**	**29.51**	**304**	**12.09**	**305**	**0.46**	**avg**	**345**	**42.16**	**344**	**17.33**	**345**	**0.52**

hary's [11] router. Its time complexity not only takes $O(kn^2)$ on the average, even $O(n^3)$ in the worst case (i.e. the case where all elements in a sorting vector are in reverse order). In general this router cannot generate an optimal solution in terms of routing tracks required and time performance. Chen [13] has presented an optimal router in terms of routing tracks required for the non-Manhattan channel routing. But the time and space complexities of the router preserved a high order of $O(k^2n)$ and $O(kn)$ 'respectively' on the average, even $O(n^3)$ time in the worst case, where k is asymptotic to n. Our proposed algorithm spends $O(kn)$ on the average or $O(n^2)$ in the worst case for time complexity, and $O(n)$ for space complexity.

To explore the features of our proposed router, we tested the last three bubble-sort based algorithms on randomly generated examples as shown in Table 3. Experimentally, our CPU-time T (in seconds) performance is 97% better than that of [11] and 40% better than that of [13] as shown in Fig. 12. The time performance improvement of [11] is denoted as zero in Figure 12 because we compare both our and Chen's [13] routers to [11]. And the number of routing tracks required by our router as shown in Table 3 is only 1% more than that of the optimal router [13].

Finally, the complexity comparisons among our and other three-layer non-Manhattan routing algorithms are shown in Table 4. It is seen that the time complexity of our approach is better than the other algorithms [11, 13].

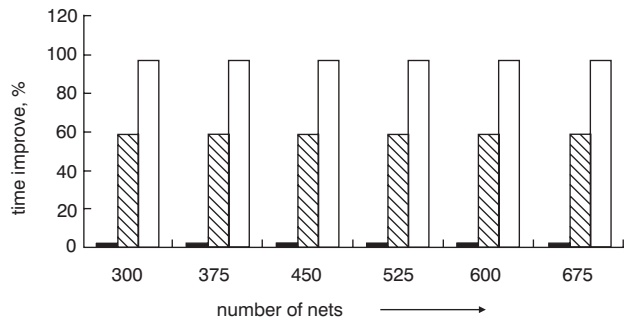

Fig. 12 *Comparison on performance improvement for examples in Table 3*

■ Chaudhary [11]
▨ Chen [13]
□ Our router

Table 4: Comparisons for three-layer routing algorithms

Algorithms \ Complexity	Time complexity average	Time complexity worst	Space complexity average	Space complexity worst
Chaudhary [11]	$O(kn^2)$	$O(n^3)$	$O(n)$	$O(n)$
Chen [13]	$O(k^2n)$	$O(n^3)$	$O(kn)$	$O(n^2)$
Our router	$O(kn)$	$O(n^2)$	$O(n)$	$O(n)$

6 Conclusions

We have described efficient bubble-sort-based algorithms for the two- and three-layer non-Manhattan channel routing problems. Based on the same routing model, the time complexities of our algorithm, two previous algorithms Chaudhary's [11], and Chen's [13] for the two-layer (and three-layer) non-Manhattan channel routings are $O(kn)$, $O(kn^2)$, and $O(k^2n)$, respectively, where k is the number

of sorting passes required and n is the number of two-terminal nets in a channel.

To further conduct the performance analysis of the three bubble-sort based algorithms, we tested them on a set of examples. Experimental results indicate that our algorithm requires only 1% more routing tracks than the optimal Chen's router [13] and the time improvement is over 40% of [13] on the average. Clearly, the time performance of our algorithm is better than previous algorithms [11, 13]. In future, we plan to integrate our bubble-sort router into an over-the-cell (OTC) channel router to reduce the final channel height in VLSI chip design.

7 Acknowledgments

The authors thank the anonymous referees for their constructive and helpful comments on this article. This work was supported by the National Science Council, Taipei, Taiwan, Republic of China, under grant NSC 86-2221-E-002-066.

8 References

1 HASHIMOTO, A., and STEVENS, J.: 'Wire routing by optimizing channel assignment within large apertures'. Proceedings of the 8th *Design automation* workshop, 1971, pp. 155–169
2 DEUTSCH, D.N.: 'A dogleg channel'. Proceedings of the 13th *Design automation* conference, 1976, pp. 425–433
3 YOSHIMURA, T., and KUH, E.S.: 'Efficient algorithms for channel routing', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1982, **1**, pp. 25–35
4 RIVEST, R.L., and FIDUCCIA, C.M.: 'A greedy channel router'. Proceedings of the 19th *Design automation* conference, 1982, pp. 418–424
5 BURSTEIN, M., and PELAVIN, R.: 'Hierarchical channel router', *VLSI J.*, 1983, **1**, pp. 21–38
6 CONG, J., WONG, D.F., and LIU, C.L.: 'A new approach to three- and four-layer channel routing', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1988, **7**, pp. 1094–1104
7 SAIT, S.M., and YOUSSEF, H.: 'VLSI physical design automation' (McGraw-Hill, 1995), pp. 289–332
8 LODI, E., LUCCIO, F., and PAGLI, L.: 'A preliminary study of a diagonal channel-routing model', *Algorithmica*, 1989, 4, pp. 585–597
9 WANG, D.C.: 'Novel routing schemes for IC layout part I: two-layer channel routing'. Proceedings of the 28th *Design automation* conference, 1991, pp. 49–53
10 WANG, D.C., and KUH, E.S.: 'New algorithms for two- and three-layer channel routing', *Int. J. Circuit Theory Appl.*, 1991, **19**, pp. 525–549
11 CHAUDHARY, K., and ROBINSON, P.: 'Channel routing by sorting', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1991, **10**, pp. 754–760
12 KNUTH, D.E.: 'Sorting and searching, 3' (Addison-Wesley, Reading, MA, 1973)
13 CHEN, C.Y.R., HOU, C.Y., and SINGH, U.: 'Optimal algorithms for bubble sort based non-Manhattan channel routing', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1994, **13**, pp. 603–609