

On the Crossing Distribution Problem in Two Regions*

T. K. YU⁺ AND D. T. LEE^{***}

⁺*Department of Computer Science and Information Engineering
National Taiwan University
Taipei, 106 Taiwan*

E-mail: tkyu@ntu.edu.tw

^{**}*Institute of Information Science
Academia Sinica
Taipei, 115 Taiwan*

E-mail: dtlee@iis.sinica.edu.tw

VLSI layout design is typically decomposed into four steps: placement, global routing, routing region definition and ordering, and detailed routing. The crossing distribution problem occurs prior to detailed routing, in which crossings of nets are distributed according to design constraints. In this paper, we propose an $O(n \log n)$ time algorithm for the crossing distribution problem for two-terminal nets in two regions, where n is the number of nets. This algorithm improves a previously known result that runs in $O(n^2)$ time [6]. We also give an $O(n^2)$ time algorithm for the crossing distribution problem for three-terminal nets under some assumption.

Keywords: VLSI, crossing distribution problem, crossing minimization problem, detailed routing, two-terminal net, multi-terminal net

1. INTRODUCTION

A VLSI circuit is usually modeled so as to be composed of *modules* and a set of *nets*. Each net specifies on the boundary of the modules a subset of points, called *terminals*, to be connected by wire. The *layout* problem is to interconnect the modules as specified by the nets in accordance with different technological design rules. Normally, the *layout* problem is solved in four steps [6]:

Placement: the modules are placed on the plane.

Global routing: the routing region is partitioned into simple subregions, each called an *elementary region*, and global assignment of the wiring paths is determined for each net.

Routing region definition and ordering: The routing region is usually decomposed into rectangular channels and/or L-shaped channels. Channels have to be ordered properly such that their size can be adjusted without rerouting the previously completed channels.

Detailed routing: the wiring of each of the individual routing regions is given.

A VLSI layout (Fig. 1) consists of some *modules*, *regions* and *nets*. In Fig. 1, a sketch of a global routing is shown, in which each net is specified by a list of regions

Received January 31, 2003; accepted July 4, 2003.

Communicated by Shih-Pyng Shieh.

* A preliminary version of this paper has been presented at the 2002 International Computer Symposium, 2002.

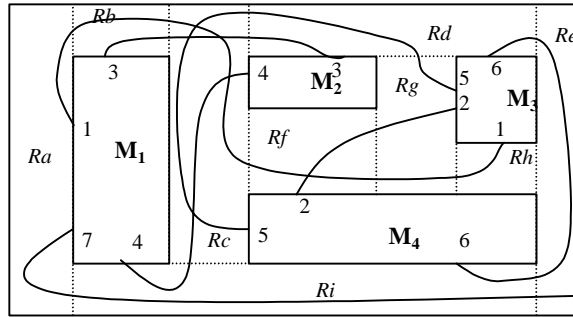


Fig. 1. An example of VLSI layout.

through which its wiring passes. For instance, for Net 5, the sequence of regions is (Rc, Rd, Rg) . Net 1 and Net 3 will have a crossing if we require that the sequence of regions passed through for each net, i.e., the homotopy, be fixed as specified. The positions crossed by the nets at the boundary of two adjacent regions are referred to as the junction terminals. Once the ordering of the junction terminals, say from left to right for each horizontal boundary edge and from bottom to top for each vertical boundary edge, is fixed, the number of crossings for each region is thus determined. Indeed, crossings in VLSI layouts imply the use of vias and require one or more layers in detailed routing [5]. As vias take up routing space, the number of vias allowed for each routing region may have to be restricted. The crossing distribution problem (CDP) calls for specification of the ordering of the junction terminals for all the nets such that every routing region has no more crossings than allowed. That is, we must appropriately distribute these crossings into regions in order not to violate the quota of vias in each region. The problem was recently studied by Groenveld [3], Marek-Sadowska and Sarrafzadeh [5], Wong and Shung [7], and Song and Wang [6]. In [5], Marek-Sadowska and Sarrafzadeh presented an $O(M \log M)$ algorithm for solving a CDP for two-terminal nets in two regions, where M is the number of non-redundant crossings of the global routing. Song and Wang [6] presented an $O(n^2)$ algorithm for CDP for two-terminal nets in two regions, where n is the number of two-terminal nets between two regions. In this paper, we study the crossing distribution problem for two-terminal nets in two regions and present an $O(n \log n)$ algorithm. The result can be generalized to some cases where there are multiple regions. In section 2, we define the problem. In sections 3, 4 and 5, we provide a detailed description of the algorithms for solving the CDP for two-terminal nets, and in section 6, we consider the problem for three-terminal nets and give an $O(n^2)$ time algorithm under some assumption.

2. PROBLEM DEFINITION

Without loss of generality, we will consider the case where the routing region can be modeled as a circle and terminals are located on the circumference, as shown, e.g., in Fig. 2. Unless otherwise specified, all nets are two-terminal nets in the following discussion. Imagine that we have a cut line that cuts across the circle and divides it into two regions. This routing region can be further represented as a two-shore channel routing region as shown on the right part of Fig. 2.

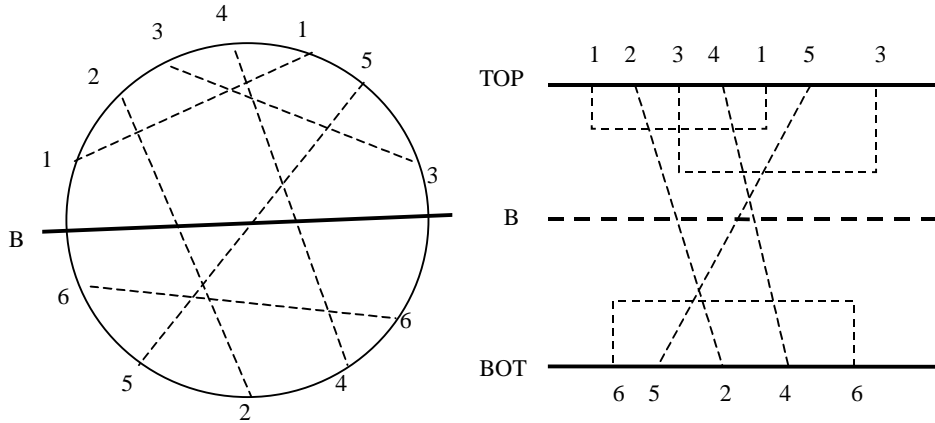


Fig. 2. The crossing distribution problem (CDP) for in two regions.

Let TOP and BOT be two horizontal lines on which terminals are placed. A two-terminal net $v = (p, q)$ is *two-sided* if v has a bottom terminal, p , on BOT and a top terminal, q , on TOP. A two-terminal net $u = (r, s)$ is *one-sided* if both r and s are on either BOT or TOP. A *crossing* is an intersection of two different nets. We distinguish between *inherent* (*necessary* or *forced*) crossings and *redundant* crossings [6]. Intuitively, an inherent crossing between two nets is one that cannot be removed by a connection homotopy [5]. Consider two nets, a and b ; three different types of inherent crossings between a and b are shown in Fig. 3. Some redundant crossings are shown in Fig. 4.

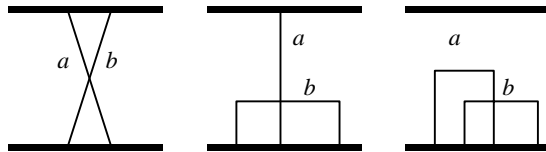


Fig. 3. Three types of inherent crossings between a and b .

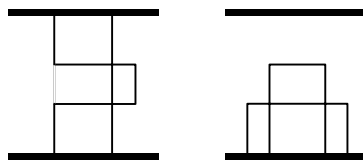


Fig. 4. Some redundant crossings.

In this paper, we eliminate redundant crossings whenever possible. It is trivial to eliminate redundant crossings for two-sided nets. For one-sided nets, this task is not difficult either, and it will be discussed in section 5.

Let R_1 and R_2 denote two routing regions sharing a *common boundary* B . Given a global routing of n two-terminal nets whose terminals are located on the outer boundary of the routing regions R_1 , and R_2 respectively, let C denote the total minimal number of crossings that exist in R_1 and R_2 , and let k denote an integer ($k \leq C$). The two-region crossing distribution problem is to find an ordering of the nets (junction terminals) at the boundary B such that exactly k crossings are located in R_1 , and $C - k$ crossings are located in R_2 . As mentioned previously, we assume that the outer boundary of the two adjacent routing regions, R_1 and R_2 , can be represented by a circle whose circumference contains the terminals. Furthermore, we assume that the top part of the circle refers to region R_1 and the bottom part to region R_2 . Fig. 2 gives an illustration. Let TOP and BOT represent the top and bottom boundaries, respectively, that contain an ordering of the nets whose global routes are dissected by B .

The crossing distribution problem in the two regions R_1 and R_2 is to determine the net orderings of the junction terminals at the boundary B such that no redundant crossings are introduced and crossings are “properly” distributed between these two regions. Formally, we define the problem as follows:

Problem I. Let $\Psi = \{N_1, N_2, \dots, N_n\}$ be a set of n ($n \geq 1$) nets. Let TOP = (a_1, a_2, \dots, a_u) and BOT = (b_1, b_2, \dots, b_v) be two sequences of terminals of the nets on the top and bottom lines, respectively, where a_i (or b_j) refers to a net number representing a net terminal and i denotes the terminal position ($1 \leq i \leq u$, $1 \leq j \leq v$) on TOP or BOT. Given a boundary B and an integer quota k ($k \leq C$, where C is the total number of crossings), distribute exactly k crossings to R_1 and $C - k$ crossings to R_2 .

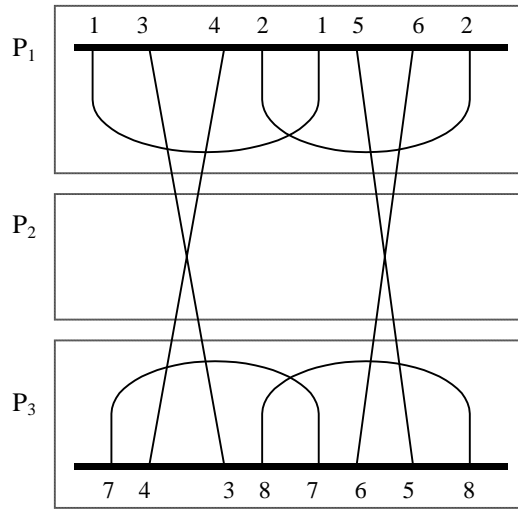
Let P denote the set of crossings in the global routing in two regions. Let X , Y , and Z denote the set of one-sided nets on TOP, two-sided nets, and one-sided nets on BOT, respectively. The set Ψ of nets is, thus, partitioned into three pairwise disjoint subsets, X , Y , and Z . That is, we have $\Psi = X \cup Y \cup Z$, $X \cap Y = \emptyset$, $X \cap Z = \emptyset$, and $Y \cap Z = \emptyset$. Let (a, b) denote the crossing between nets a and b . Define the following three sets of crossings [6]:

$$\begin{aligned} P_1 &= \{(a, b) \mid ((a \in X) \wedge (b \in X)) \vee ((a \in Y) \wedge (b \in X)) \vee ((a \in X) \wedge (b \in Y))\}; \\ P_2 &= \{(a, b) \mid (a \in Y) \wedge (b \in Y)\}; \\ P_3 &= \{(a, b) \mid ((a \in Z) \wedge (b \in Z)) \vee ((a \in Y) \wedge (b \in Z)) \vee ((a \in Z) \wedge (b \in Y))\}. \end{aligned}$$

Fig. 5 shows an example; in this case $P_1 = \{(1, 2), (1, 3), (1, 4), (2, 5), (2, 6)\}$, $P_2 = \{(3, 4), (5, 6)\}$, $P_3 = \{(7, 8), (7, 3), (7, 4), (8, 5), (8, 6)\}$. Note that $P = P_1 \cup P_2 \cup P_3$, and that these three cases are mutually exclusive. We therefore divide our original problem into three sub-problems, one for two-sided nets (P_2) and two for one-sided nets (P_1 and P_3).

With the definition above, we can divide problem I into problem II and problem III described below:

Problem II. Crossing distribution for two-sided nets only (P_2). Let $\Psi = \{N_1, N_2, \dots, N_n\}$ be a set of n ($n \geq 1$) two-sided nets. Let TOP = $\{a_1, a_2, \dots, a_n\}$ and BOT = $\{b_1, b_2, \dots, b_n\}$ be two sequences of terminals of the nets of Ψ on the top and bottom lines, respectively,

Fig. 5. Three types of crossings, P_1 , P_2 , and P_3 .

where a_i (or b_i) is a net number representing a net terminal and i denotes the terminal position ($1 \leq i \leq n$). Given a boundary B and an integer quota k ($k \leq C$, the total number of crossings), find a permutation of net terminals at the boundary B such that exactly k crossings are located at R_1 and $C - k$ crossings are located at R_2 .

Problem III. Crossing distribution for two-sided nets in the presence of one-sided nets (Fig. 5 P_1 , P_3). Let $\Psi = (N_1, N_2, \dots, N_n)$ be a set of n ($n > 1$) nets such that N_i ($i = 1, \dots, n$) is either a one-sided net on a line G or a two-sided net having a terminal on G . Let $L = (t_1, t_2, \dots, t_r)$, $r \leq 2n$, be the sequence of terminals of the nets in Ψ on G . Given a boundary B and an integer quota k ($k \leq C$, where C is the total number of crossings), distribute exactly k crossings above B .

3. CROSSING DISTRIBUTION FOR TWO-SIDED NETS

Without loss of generality, we assume that $TOP = (1, 2, 3, \dots, n)$, and that BOT is a permutation of $\{1, 2, 3, \dots, n\}$. The two-sided nets $N_i = (p, q)$, $N_j = (r, s)$ form a crossing iff $(p < r$ and $q > s)$ or $(p > r$ and $q < s)$. In [6], Song and Wang noted that the number of crossings is equal to the number of inversions [2, 4] of the permutation. For instance, permutation $(4, 2, 1, 3)$ has inversions $(4, 2)$, $(4, 1)$, $(4, 3)$, $(2, 1)$. When we place $(1, 2, 3, 4)$ on TOP and $(4, 2, 1, 3)$ on BOT , and then connect the same numbers with straight lines, we will find 4 crossings, and they are exactly $(4, 1)$, $(4, 2)$, $(4, 3)$, and $(2, 1)$.

Note that given a sequence of numbers, we know that number i at position s_i and number j at position s_j have an inversion if $j < i$ and $s_j > s_i$. For each integer $i \in (1, 2, \dots, n)$, the number of inversions induced by i can be obtained as follows. Since there are $i - 1$ numbers less than i , if we know the number m of such integers lying to the left side of i in the permutation, then the number of inversions induced by integer i is $i - 1 - m$. We call the number m the *magic number* of i . We will discuss this in more detail in section 4.

As shown in Fig. 6, the magic numbers for the integers in the sequence on BOT are respectively 0, 0, 1, 3, 4, 3, and 1 respectively. The numbers shown in the rectangular box in Fig. 6 are the inversion numbers induced by the integer immediately above.

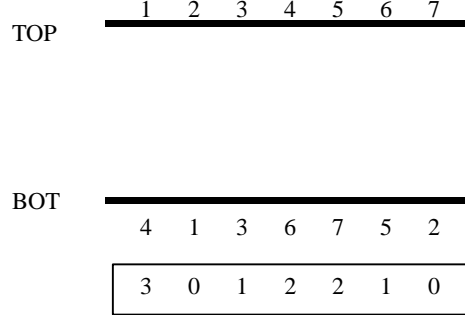


Fig. 6. Inversion numbers induced by each integer.

How to compute the magic numbers efficiently is described in the next section. The following lemma is obvious:

Lemma 1 If we have all the magic numbers of a permutation of $\{1, 2, \dots, n\}$ we can find the number of inversions of the permutation. \square

As will be shown later, CDP for two-sided nets can be solved easily, once we have computed the magic number for each integer. We now give an algorithm to compute a permutation of $\{1, 2, \dots, n\}$ whose inversion or crossing number is $k \leq C$, where C is the total number of inversions of a given permutation BOT of $\{1, 2, \dots, n\}$. We build a list called B-list to record the permutation on boundary B. Initially the permutation is the identity permutation, i.e., the same as TOP. In other words, all the crossings are located at R_2 (in the lower region). If we output a node from B-list, we delete it at the same time.

Algorithm 1. Crossing distribution for two-sided nets.

Input: k an integer; $BOT = (b_1, \dots, b_n)$

Output: $B = (c_1, c_2, \dots, c_n)$ a permutation on B

1. Build *B-list* as $(1, 2, \dots, n)$
2. Scan BOT from b_1 to b_n
 - 2.1. if $k = 0$ then break
 - 2.2. find current node b_i 's magic number m and its inversion $r = b_i - 1 - m$
 - 2.3. if $(r > k)$ then output first $r - k$ nodes of *B-list*
output b_i ; $k = 0$
 - 2.4. else output b_i ; $k = k - r$
3. Output the rest of nodes of *B-list* in linear order.

Let us illustrate this algorithm with an example. In Fig. 7, the initial B-list is the same as TOP, and we assume that $k = 5$, and that the list of inversion numbers induced

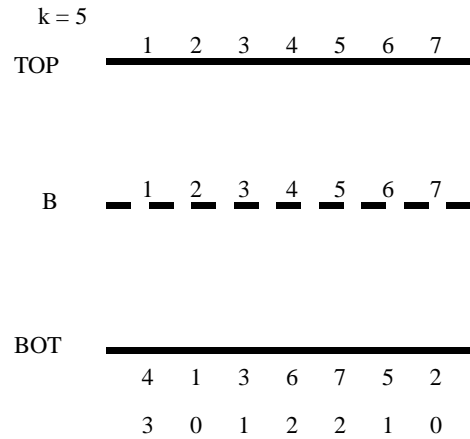


Fig. 7. Initial situation, assuming inversion numbers are given as shown.

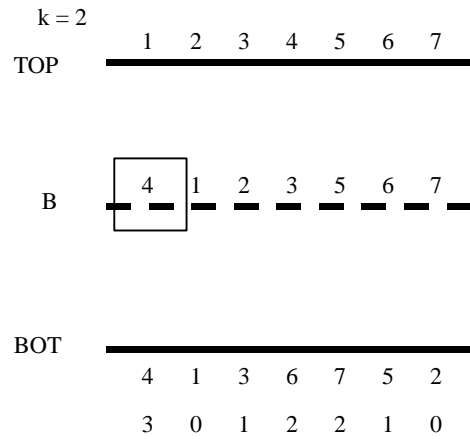


Fig. 8. After an iteration, 4 has been outputted, and k becomes 2.

by each integer in BOT has been pre-computed. The first number of BOT is 4, and its inversion number is 3. Since $k = 5 > 3 = r$, we output 4 (and delete 4 from B-list) and k becomes to $k - 3 = 2$ (see Fig. 8).

In Fig. 8, the square containing 4 on B is the current output sequence. Let us pause for a moment here. The initial sequence routing of the first four nets, as shown in Fig. 9, causes 4 crossings in R_2 . In Fig. 10, we move 4 to the head of the permutation. This movement will cause 3 crossings to relocate in region R_1 . This is due to the fact that each time we interchange two adjacent elements in a permutation, we increase or decrease the total number of inversions by one [4]. Now, the next element is 1, and its inversion number is 0. We do nothing except output this number (Fig. 11). Next, we find 3 and its inversion number 1 (Fig. 12). Since $k = 2 > 1 = r$, we output 3 and update k to be 1. Now, we have four crossings in R_1 .

When we find 6, $k = 1 < 2 = r$, according to Step 2.3, we start to output B-list $r - k$ nodes. This means that we only need to make k exchanges of adjacent positions to meet

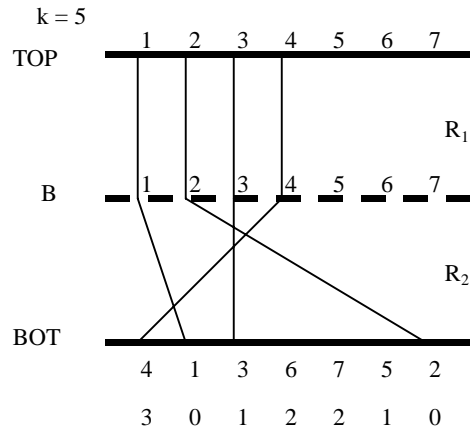


Fig. 9. Four crossings in R_2 .

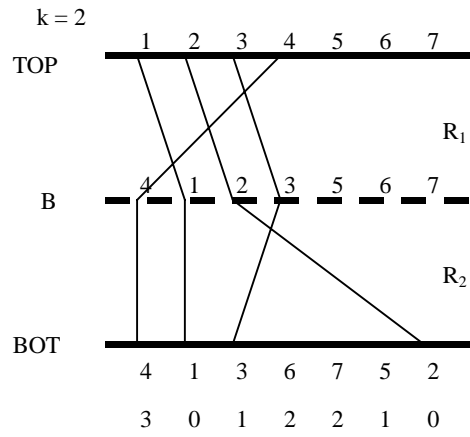


Fig. 10. Three crossings in R_1 and one crossing in R_2 .

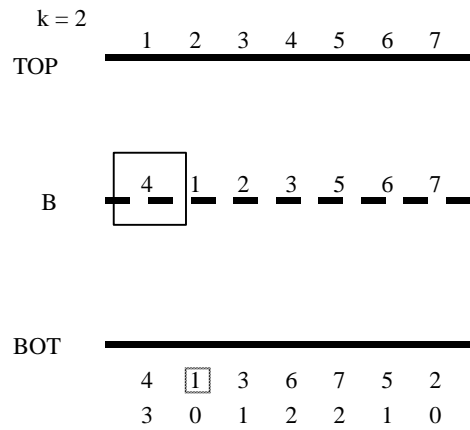


Fig. 11. 1 is scanned next.

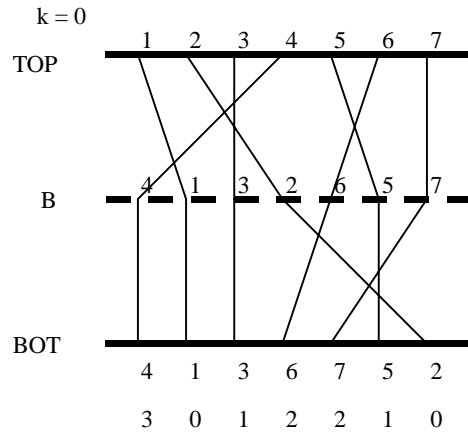


Fig. 15. Totally 9 crossings. 5 in R_1 , 4 in R_2 .

the quota requirement. So we output $r - k$ nodes, then output 6, and set k to 0. We will break this iteration and do step 3 to output other nodes of B-list (see Figs. 13 and 14). Fig. 15 is the final routing result, which satisfies the quota requirement, and we can see that there is no redundant crossing.

Theorem 1 The two-sided CDP problem for n nets can be solved in $O(n \log n)$ time.

Proof: When $k = 0$ or after we have scanned the whole BOT list, the algorithm would terminate. Since the algorithm does not create redundant crossings (two-sided nets by default have no redundant crossing), it satisfies the quota requirement [6]. Therefore, it is correct. Let us now analyze the time complexity of the algorithm. It is obvious that other than the time needed to compute the magic (inversion) number for each element in BOT, the total time needed is linear. As we will show below the time needed to compute the magic numbers and hence the inversion numbers is $O(n \log n)$. This completes the proof. \square

4. FINDING MAGIC NUMBERS EFFICIENTLY

To find the magic number of some b_i , we should find out how many numbers less than b_i which occur before b_i in the permutation. If we record all the scanned elements, we can easily compute the number of scanned elements that are smaller than the current element. The inversion number induced by the current element is obtained by simply subtracting the magic number plus 1 from b_i . Therefore, if we could find the magic number in $O(\log n)$ time, the total time complexity would be $O(n \log n)$.

Since this problem is rather straightforward, we will illustrate it by using an example. The underlying data structure used is a height-balanced binary search tree. Basically, for each node, we store a key plus an additional field, called *left-number*, containing the total number of elements stored in its left subtree plus one. That is, for each key, we store the total number of elements in the current binary search tree less than or equal to itself.

When an element is scanned, it is inserted into the tree in an appropriate position. In the meantime, its magic number is computed as follows. Initially, it is 0. Each time a right subtree of a node v is followed, the magic number is incremented by $\text{left-number}(v)$. When a left subtree of a node v is followed, $\text{left-number}(v)$ is incremented by 1 (indicating that the newly inserted number is less than $\text{key}(v)$). Since insertions may result in height imbalance, rotations to restore height balance will be needed. All these operations are known to take $O(\log n)$ time per insertion for height balanced binary search trees.

In Fig. 16, 5's left-number is 3 since it has 2 nodes in its left subtree. Similarly, 9 has "7" in its left subtree, so its left-number is 2.

See Fig. 17. When we insert 8 into this tree, since $8 > 5$, we follow the right subtree, and the magic number is now 3. Going down the tree, we meet 9. Since $8 < 9$, we follow the left subtree, so the left-number of node 9 is incremented by 1. Finally, we insert node 8 into the correct place and add its left-number 1 to obtain its final magic number, which is 4.

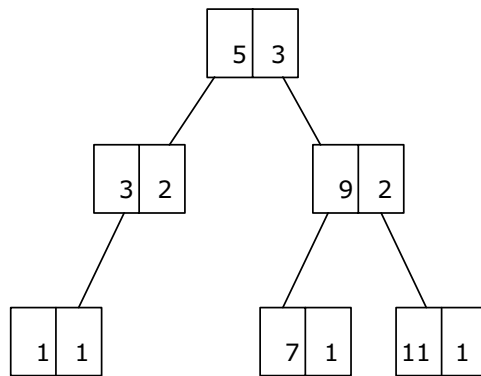


Fig. 16. An example of left-numbers in a balanced tree.

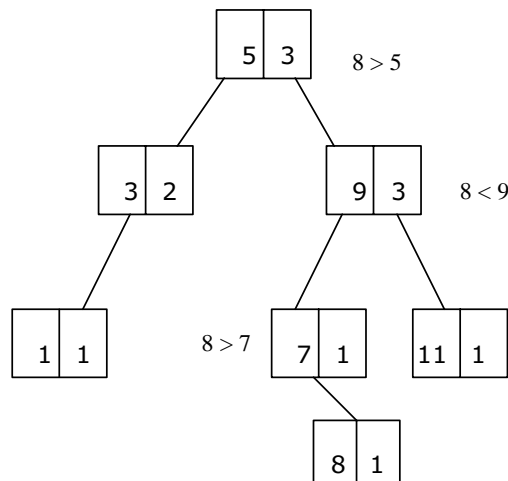


Fig. 17. After inserting 8 to balanced tree.

Algorithm 2. Insert a node into a left-numbered tree and find its magic number.

Input: a node p ; a left-numbered height-balanced tree T

Output: magic number m of p

1. $m = 0$; $u = T.root$; $p.left-number = 1$

2. if $p.key > u.key$

 then $m = m + u.left-number$

$u = u.right$

 else $u.left-number = u.left-number + 1$

$u = u.left$

3. if $u = null$

 then insert p into u

 return m

 else goto 2

5. CROSSING DISTRIBUTION PROBLEM IN THE PRESENCE OF ONE-SIDED NETS

We will only discuss P_3 here (a one-sided net on BOT). The method for P_1 is similar. In [6], Song and Wang enumerated all the crossings and used a topological sort to solve this problem. The crossing number is bounded by n^2 , so their algorithm takes $O(n^2)$ time in the worst case. We show below that we need not enumerate all the crossings to solve this problem.

In this section, we will also use the left-numbered height-balanced tree to solve this problem. We make some observations to help us understand this problem better. Fig. 18 shows an example of one-sided net routing, where N_3 , N_5 , and N_7 are two-sided nets, and the others are one-sided nets. Let each one-sided net N be denoted by $(Begin(N), End(N))$, where $Begin(N)$ and $End(N)$ denote the beginning and ending terminals of the net.

Definition Net N_i is said to *contain* Net N_j if and only if $Begin(N_i) < Begin(N_j) < End(N_j) < End(N_i)$ (in Fig. 18, $Begin(N_1)$ is 1, $End(N_1)$ is 7, and N_1 contains N_2).

In Fig. 19, N_1 contains N_2 . If we route N_2 in the upper region and N_1 in the lower region, two redundant crossings will occur. Therefore, if N_i contains N_j , we should always route N_i “above” N_j to prevent redundant crossings. This “above” routing constraint is implicitly maintained in our algorithm described below when N_i is routed before N_j .

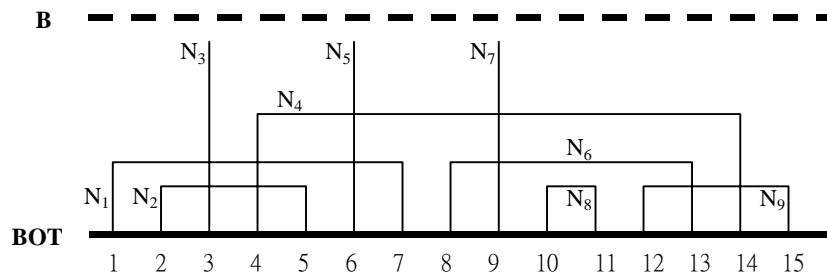


Fig. 18. An example of one-sided net routing.

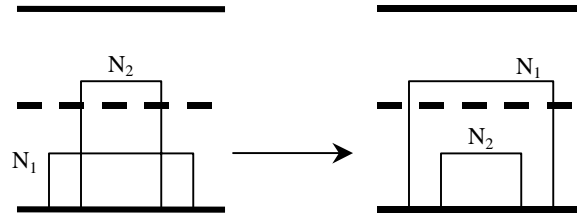


Fig. 19. Prevent redundant crossings.

Lemma 2 If $\text{Begin}(N_i) < \text{Begin}(N_j)$, N_j cannot contain N_i .

Proof: Immediate. □

With lemma 2, we can use the order of the beginning terminal to solve this problem. We scan the BOT list once to find some information: (1) the beginning positions of all one-sided nets, (2) the ending positions of all one-sided nets, and (3) the positions of all two-sided nets. This information can be found in linear time.

After finding this information, we initialize a left-numbered height-balanced tree by inserting all two-sided nets with positions as keys. This tree records the current order of net lines piercing boundary B. In other words, it records the current permutation of two-sided net on boundary B.

Algorithm 3. Crossing distribution in the presence of one-sided nets.

Input: $\text{BOT} = (b_1, b_2, \dots, b_n)$; k an integer.

Output: $B = (c_1, c_2, \dots, c_n)$ a permutation on B

1. Find the information of all the nets with regard to their positions.
2. Build a left-numbered height-balanced tree T by inserting two-sided nets in order.
3. Scan BOT from b_1 to b_n .
 - 3.1 Let the current node be b_i , and let its correspond net be N_j .
 - 3.2 if N_j has been processed (we encounter its right terminal) or it is a two-sided net
then continue.
 - 3.3 Find the left numbers of $\text{Begin}(N_j)$ and $\text{End}(N_j)$ in T, called m_1 and m_2 .
 - 3.4 if $m_2 - m_1 \geq k$ then
insert $\text{Begin}(N_j)$ to T; Break
else
insert $\text{Begin}(N_j)$ and $\text{End}(N_j)$ to T; $k = k - (m_2 - m_1)$; continue.
4. Output nodes in tree T by means of inorder traversal. After outputting $\text{Begin}(N_j)$, continue outputting k nodes, and then output $\text{End}(N_j)$. Finally, output the other nodes in tree T.

In Step 3.3, the number m_1 (respectively, m_2) denotes the number of nets piercing the boundary B and lying to the left of $\text{Begin}(N_j)$ (respectively $\text{End}(N_j)$). Therefore, $m_2 - m_1$ denotes the number of nets piercing the boundary and lying between the two terminals of the one-sided net N_j . For instance, Fig. 20 shows an initial state of a one-sided net routing, where the initial left-numbered tree contains N_3 , N_5 , and N_7 . At the first iteration

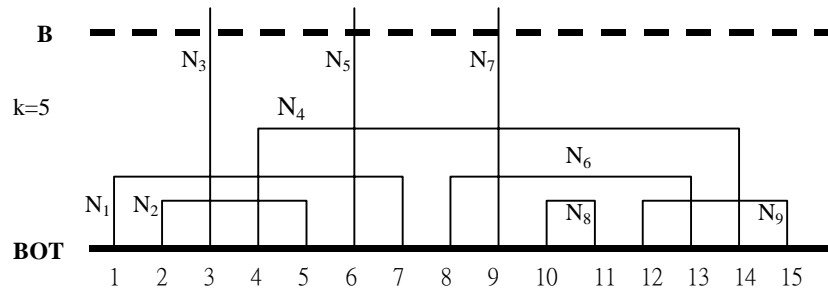


Fig. 20. Initial state.

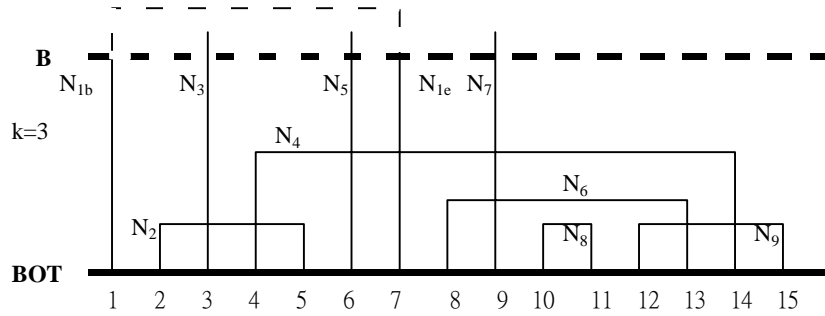


Fig. 21. After processing N_1 .

of Step 3, we find N_1 . We use $\text{Begin}(N_1)$ and $\text{End}(N_1)$ to find $m_1 = 0$ and $m_2 = 2$, respectively. Quota $k = 5 > 2 = m_2 - m_1$, so we insert $\text{Begin}(N_1)$ and $\text{End}(N_1)$ into the left-numbered tree and decrease k by 2.

This means if we route net N_1 in the upper region R_1 and we will cause an $m_2 - m_1$ crossings to occur in R_1 . Now, N_{1b} (beginning) and N_{1e} (ending) behave as if they were two-sided nets after this iteration, so we insert them into the left-numbered tree.

At the next iteration, N_2 is scanned, $m_1 = 1$, and $m_2 = 2$. Since $k = 3 > 1 = m_2 - m_1$, we insert $\text{Begin}(N_2)$ and $\text{End}(N_2)$ into the left-numbered tree and decrease k by 1 (Fig. 22).

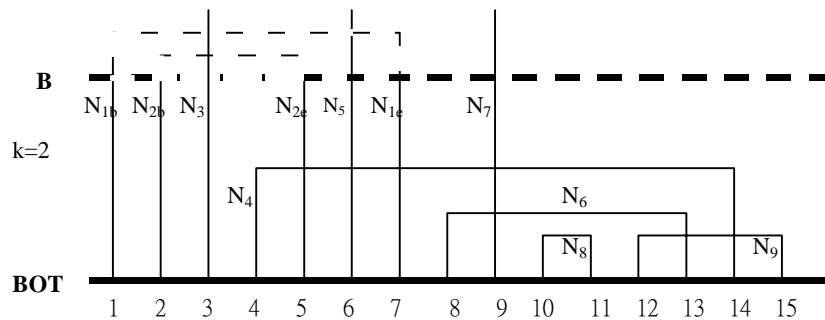


Fig. 22. After processing N_2 .

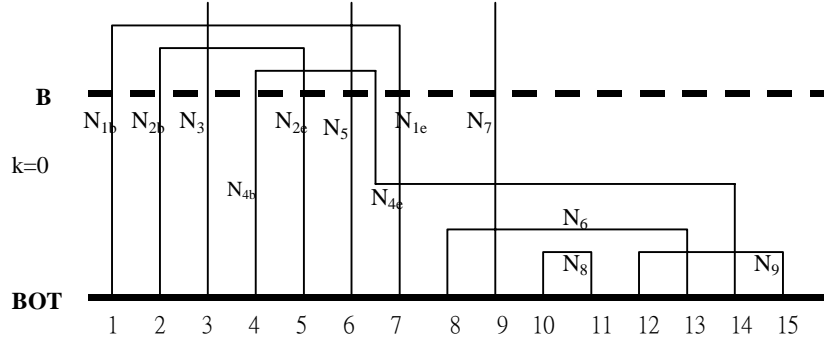


Fig. 23. The final result.

Next, we find N_4 , and we have $m_1 = 3$, $m_2 = 7$. $k = 2 < m_2 - m_1$, and we insert $\text{Begin}(N_4)$ into T to obtain the sequence $(N_{1b}, N_{2b}, N_3, N_{4b}, N_{2e}, N_5, N_{1e}, N_7)$ and then go to Step 4. We then output 4 nodes using inorder traversal of T until $\text{Begin}(N_4)$. Therefore, we output $k (= 2)$ nodes in T and then output $\text{End}(N_4)$. And then others in T are output. That is, the final permutation is $(N_{1b}, N_{2b}, N_3, N_{4b}, N_{2e}, N_5, N_{4e}, N_{1e}, N_7)$ and the routing is shown in Fig. 23. Initially (see Fig. 20), there are 10 crossings in R_2 . After running this algorithm, we have 5 crossings in R_1 and 5 crossings in R_2 .

Theorem 2 The CDP in the presence of one-sided nets with n nets can be solved in $O(n \log n)$ time.

Proof: The algorithm terminates when $k \leq m_2 - m_1$ or the entire list BOT is scanned. The quota is not exceeded when $k > m_2 - m_1$. When $k \leq m_2 - m_1$, we output $\text{Begin}(N_i)$ at k positions to the left of $\text{End}(N_i)$ to create k crossings in the upper region to meet the quota requirement. As for the time complexity, Step 2 takes time $O(t \log t)$, where t is the number of two-sided nets. Steps 1 and 4 each need linear time, and Step 3 takes $O(n \log n)$ time to complete the operations needed for left-numbered tree searching and insertions. Thus, the time complexity is totally $O(n \log n)$. \square

Having solved these sub-problems, we will now describe how we can solve the original problem. Given a CDP with as inputs the permutations of TOP and BOT, and an integer k , we solve P_1 . Let $C(P_i)$ denote the crossing number in P_i , $i = 1, 2, 3$. Initially, the permutation on boundary B is just like TOP and all the crossings (including P_1, P_2, P_3) are assumed to be in the lower region. Note that $C(P_i)$ for $i = 1, 2, 3$ can be obtained by scanning the permutations on TOP and BOT. If $C(P_1) \geq k$, we can find the solution after processing P_1 . If $C(P_1) < k$, then there remain $k - C(P_1)$ crossings to be moved into the upper region. We process P_2 next. If $C(P_2) < k - C(P_1)$, then there remain $k - C(P_1) - C(P_2)$ crossings. Note that after P_2 has been processed (all crossings have been moved to the upper region), the permutation of two-sided nets on B is just like BOT. Therefore, we can process P_3 in the final step.

6. CDP FOR THREE-TERMINAL NETS IN TWO REGIONS

In [5] Marek-Sadowska and Sarrafzadeh introduced the *multi-terminal net problem*. They gave a deterministic algorithm for three-terminal nets in time $O(mn^2 + m\xi^{3/2})$, where m is the number of modules, n is the number of nets, ξ is the number of crossings, and the algorithm is a heuristic algorithm for multi-terminal nets in multi-regions. In this paper we will study the CDP of three-terminal nets in two regions. We will give an $O(n^2)$ algorithm to solve the problem.

We first distinguish all possible configurations of a three-terminal net containing 2 or 3 terminals. There are four types as shown in Fig. 24, referred to as I-type, U-type, Y-type and M-type respectively according to their appearance.

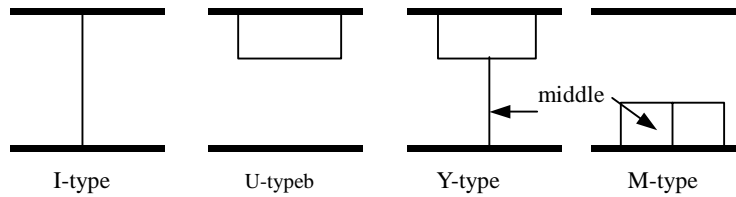


Fig. 24. Four types of nets.

We first need to guarantee that there are no redundant crossings. The only difference between a two-terminal net and a three-terminal net is that a three-terminal net may have an extra terminal, a *middle terminal* denoted by $\text{Middle}(N_i)$, in addition to beginning and ending terminals (Fig. 24).

In the two-terminal version, there are three types of crossings (Fig. 3), but in the three-terminal version, there are many more crossing types. Thus, we list all the types in Appendix A and here will choose some representative examples. The CDP of three-terminal nets can be divided into three parts just like the two-terminal version (see Fig. 5). Let Q_1 , Q_2 , and Q_3 represent the three parts of problems corresponding to P_1 , P_2 , and P_3 of the two-terminal version, respectively.

The same as before, we start with Q_2 , which contains many types of crossings (Fig. 25): (1) I-type with I-type, (2) I-type with Y-type, and (3) Y-type with Y-type. The Y-type net, in theory, has a special feature: the middle terminal can be connected to any position of the U-type base defining the beginning and ending terminals (Fig. 26 (a)). As shown in Fig. 26 (b), by moving the position of the middle net, we can prevent a redundant crossing from being created. How to determine the positions of the middle terminal for all Y-type nets so as to minimize the number of crossings is another problem, called the *crossing minimization problem (CMP)* [1]. This is a non-trivial problem, and we shall leave it for future work. From now on, we shall assume that the position of the middle terminal for each Y-type net is fixed and given in advance. Once this information is given, Q_2 can be solved using the method given in section 3 within $O(n \log n)$ time.

Lemma 3 The problem Q_2 of CDP can be solved in $O(n \log n)$ time, when the middle terminal positions of Y-type nets are given. \square

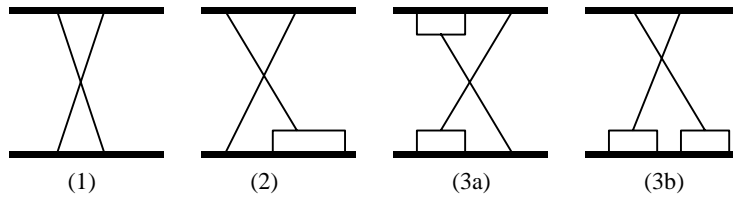


Fig. 25. Crossings in Q_2 : (1) I-type and I-type, (2) I-type and Y-type, (3a) (3b) Y-type and Y-type.

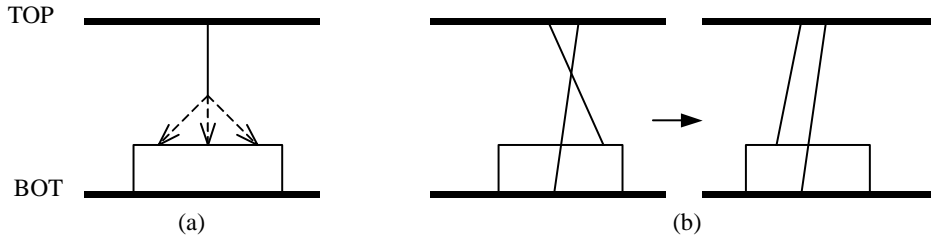


Fig. 26. (a) The middle terminal can connect to an arbitrary position between beginning and ending terminals. (b) Move middle terminal to remove redundant crossings.

Now, we want to consider Q_3 (Q_1 is similar). We first treat an M-type net like two U-type subnets in succession and treat the base (the two-terminal end) of a Y-type net like a U-type net.

Fig. 27 presents two examples: in example A, the method described in section 5 is applied, that is; we treat the M-type net as if there were two U-type subnets; but in example B, if we apply this method, a redundant crossing will be created, unless the right U-type subnet (of the M-type) is also routed in the upper region along with the left U-type subnet.

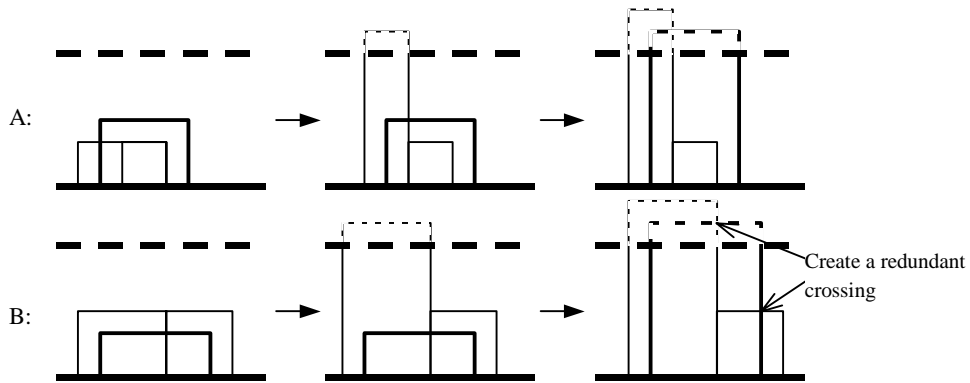


Fig. 27. A: the method of section 5 is applied; B: the method described in section 5 creates a redundant crossing.

When a U-type or an M-type net is contained in another M-type net, as shown in Fig. 28, where the inner net crosses the middle terminal of the outer net, we call this an *ill combination*. Routing for ill combination pairs needs extra treatment, for otherwise, redundant crossings may result. Indeed, these two cases are the same. The left-subnet of the inner M-type net shown in Fig. 28 (b) has no effect on the combination. Fig. 28 (b) shows a symmetric case where the left-subnet route crosses the middle terminal of the outer net, but this is similar. Therefore, we will always take Fig. 28 (a) as an example when presenting this problem.

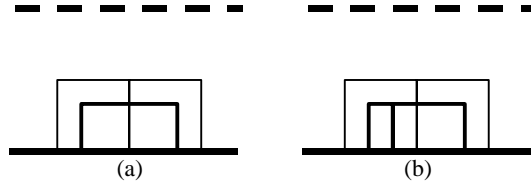


Fig. 28. Two situations that break the method in section 5.

Fig. 29 lists all the possible interactions between U-type nets and M-type nets, and between two M-type nets, excluding ill combinations. By treating M-type nets as two consecutive U-type subnets and using the method described in section 5, we can easily verify the correctness of the algorithm for Q_3 in the case where no ill combinations exist. The following lemma is obvious.

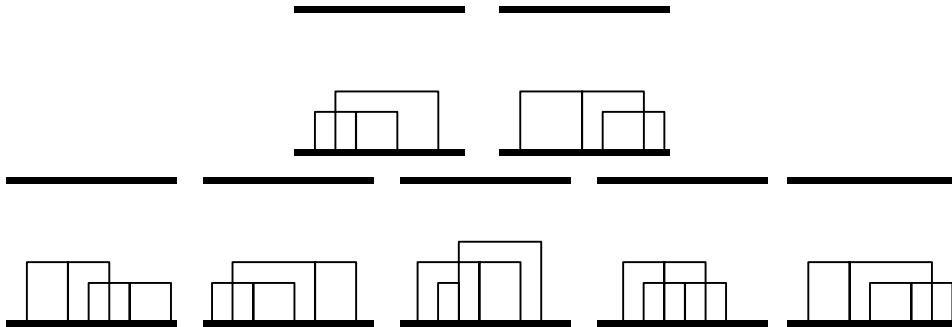


Fig. 29. All possible crossings between a U-type and an M-type and between two M-types except ill combination.

Lemma 4 If there are only I-type, U-type, M-type nets, and if there exist no ill combinations, then we can solve Q_3 of CDP in $O(n \log n)$ time. \square

We shall now address the problem of redundant crossings that may result due to ill combinations (Fig. 28). Recall that when a U-type net N_i contains another U-type net N_j , net N_i should be routed above N_j to avoid creating redundant crossings. In our algorithm given in section 5, N_i is processed before N_j , and N_i implicitly is routed above N_j . A U-type net U_i and an M-type net M_j form an ill combination when the M-type net M_j contains the U-type net U_i and the middle terminal of M_j crosses the U-type net U_i , caus-

ing one inherent crossing (Fig. 28 (a)). When the left U-type subnet of M_j and the U-type net U_i have been processed and, thus, have both been routed into the upper region, it is possible that the algorithm will terminate because the quota requirement has been met. In this case, there will be a redundant crossing caused by the pair composed of the right U-type subnet of M_j and U-type net U_i as shown in Fig. 27 case B. Thus, we need to ensure that no redundant crossing is created in any of our processing steps. In other words, when the algorithm terminates, we should guarantee that there exists no redundant crossing.

We adopt the strategy called *delay processing* to solve this problem. Consider Fig. 30 for example. Since routing the whole inner U-type net into the upper region may lead to redundant crossings, we can route just the initial part of the U-type net into the upper region (Fig. 30 picture 2) by creating a “jog” or a bend at the leg associated with the end-terminal of the U-type net. After the right U-type subnet of the outer M-type net has been routed into the upper region, the inner U-type net will be completely routed into upper region (see Fig. 30, picture 3) if necessary. This inner U-type net will be referred to as a *floating* net, and the leg associated with the ending terminal of the floating net as a *floating end*. If an inner net has many outer M-type nets with which to form ill combinations, its floating end might float to each of the middle terminals of every outer M-type net. Fig. 31 shows an example. Fig. 31 (a) shows the initial condition, indicating that N and M_1 form an ill combination, and indicating that N and M_2 also form an ill combination. Consider Fig. 31 (b), in which both $\text{Begin}(M_1)$ and $\text{Begin}(M_2)$, i.e., the left U-type subnets, have been processed. To prevent redundant crossings from being created, we must process the floating net N in an appropriate manner. In Fig. 31 (c), $\text{Middle}(M_1)$ lies to the left of $\text{Middle}(M_2)$, so the floating end of N will float to $\text{Middle}(M_1)$ at the beginning. In Fig. 31 (d), after $\text{Middle}(M_1)$ has been processed at which point, M_1 and N will not cause a redundant crossing, the initial part of N can be extended further, floating to $\text{Middle}(M_2)$. In other words, the “bend” of the ending terminal of the floating net is considered “stretchable” and will become “straight” only when the scanning process reaches the position of the ending terminal.

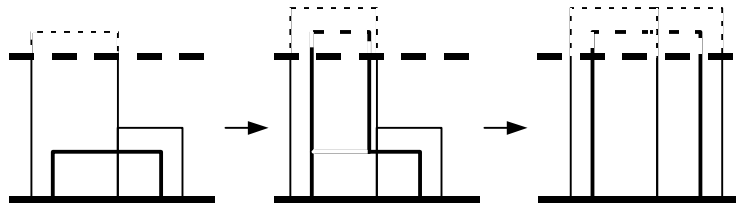


Fig. 30. The strategy for preventing redundant crossings.

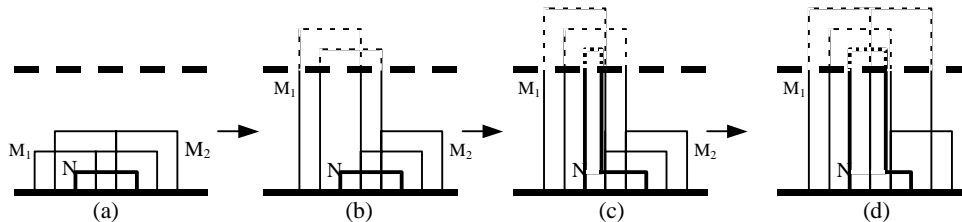


Fig. 31. (a) Initial condition. (b) $\text{Begin}(M_1)$ and $\text{Begin}(M_2)$ have been processed. (c) N has been processed, but floats to middle(M_1). (d) $\text{Middle}(M_2)$ has been processed, and N floats to $\text{Middle}(M_2)$.

We will now describe our algorithm more formally. We shall first identify the pairs of nets that form an ill combination and give the routing constraint relation “above.” Net A is “above” net B if and only if no redundant crossing will result when Net A is routed above Net B. Net B is a floating net if it is a U-type net totally contained in an M-type net M or a U-type subnet of an M-type net totally contained in another M-type net M, and if they form an ill combination pair. We use “Net A \rightarrow Net B” to denote the “above” routing constraint. In Fig. 31, we have $M_1 \rightarrow N$ and $M_2 \rightarrow N$. Note that this directed graph (for the relation “above”), called an *ill-combination graph*, has no cycles. For all nets M_i such that $M_i \rightarrow N$, we shall maintain for floating net N a *sorted list* of the middle terminals of M_i to which the floating end of net N will float in that order. The task to be performed is to maintain the “above” routing constraint imposed by the ill-combination graph. To ensure that the floating ends of the floating nets float to appropriate middle terminal positions correctly, we use the corresponding sorted lists and process the floating ends in an order-preserving manner. Suppose there is another floating U-type net N' that contains U-type net N, and M-type net $M \rightarrow N$, $M \rightarrow N'$. Note that since net N' contains net N, net N' is processed before net N and, hence, is routed above net N. When we float net N to Middle(M), we must ensure that floating end of net N' 's must be processed before net N. That is, all those floating nets should be processed in an appropriate order so that the “above” routing relation can be preserved. Fortunately those floating nets have a natural ordering which is the same as the order of beginning terminals, or the order in which these nets are scanned.

Step 2 initializes the left-numbered height-balanced tree T by inserting all of the I-type nets.

Step 3 is the main part of the algorithm. Steps 3.3 and 3.4 perform the delay process as shown in Figs. 30 and 31. The floating terminal is also inserted into the left-numbered tree. It ensures that the algorithm is correct whenever it terminates. We use the *ill combination sorted list* to handle floating terminals. Step 3.5 handles middle terminals. When a middle terminal of an M-type net is scanned, we process the right subnet of the M-type net. After we process it, some ill combination pairs involving the M-type net may no longer exist. We then proceed to finish processing the floating terminals, if any exist, attached on it. We will discuss the time complexity of this part later.

Algorithm 4. CDP of Q_3 containing I-type, U-type, and M-type nets.

Input: BOT = (b_1, b_2, \dots, b_n) ; connecting information of Y-type nets; k an integer

Output: $B = (c_1, c_2, \dots, c_n)$ a permutation on B .

1. Find all the ill combination pairs. For each floating U-type net (or subnet), construct a sorted list of the nets of those M-type nets with which it forms an ill combination.
2. Build a left-numbered height-balanced tree T by inserting I-type nets in order.
3. Scan BOT from b_1 to b_n .
 - 3.1 Let the current node be b_i , and let its corresponding net be called N_j .
 - 3.2 if b_i is an end terminal or N_j is an I-type net then continue (step 3).
 - 3.3 if N_j is a U-type net then
 - 3.3.1 Check and delete the first element E of its ill combination sorted list. Use b_i and the middle terminal position e of E (if no element exists in the list, use $\text{End}(N_j)$) as keys to find magic numbers m_1 and m_2 from T.

- 3.3.2 Insert $\text{Begin}(N_j)$ with key b_i into T
if $m_2 - m_1 \geq k$ *then* insert $\text{End}(N_j)$ into the desired position; break;
else
if N_j is floating *then* attach it to the middle terminal to which it is floating
else insert $\text{End}(N_j)$ into T
 $k = k - (m_2 - m_1)$; continue (step 3)
- 3.4 *if* N_j is an M-type and b_i is a starting terminal *then*
3.4.1 Check and delete the first element E of its ill combination sorted list. Use b_i and the middle terminal position e of E (if no element exists in the list, use $\text{Middle}(N_j)$) as keys to find magic numbers m_1 and m_2 from T.
3.4.2 Insert $\text{Begin}(N_j)$ with key b_i into T
if $m_2 - m_1 \geq k$ *then* insert $\text{Middle}(N_j)$ into the desired position; break;
else
if N_j is floating *then* attach it to the middle terminal to which it is floating.
else insert $\text{End}(N_j)$ into T.
 $k = k - (m_2 - m_1)$; continue (step 3).
- 3.5 *else* N_j is an M-type and b_i is an middle terminal
3.5.1 Check and delete the first element E of its ill combination sorted list. Use b_i and the middle terminal position e of E (if no element exists in the list, use $\text{End}(N_j)$) as keys to find magic numbers m_1 and m_2 from T.
3.5.2 Insert $\text{Middle}(N_j)$ with key b_i into T;
if $m_2 - m_1 \geq k$ *then* insert $\text{End}(N_j)$ into the desired position; break;
else
if N_j is floating *then* attach it to the middle terminal to which it is floating.
else insert $\text{End}(N_j)$ into T;
 $k = k - (m_2 - m_1)$;
- 3.5.3 Check all the elements attaching to N_j
3.5.3.1 Now we consider net N_k with terminal t attached to N_j . Check and delete the first element E of its ill combination sorted list. Find the number of terminals m in T between $[b_i, e]$ where e is the middle position of E (if no such E exists in the list, use the right terminal of the net (subnet)).
3.5.3.2 update its floating relation.
3.5.3.3 *if* $m \geq k$ *then* insert t into the desired position; break;
else
if N_k is floating *then* attach it to the middle terminal to which it is floating.
else insert t into T;
 $k = k - m$;
4. Output nodes in tree T by inorder traversal. When we reach a middle terminal, we output its attached terminals of nets in reverse order.

Now, we will investigate the correctness of this strategy. We need to prove three things: (1) it will stop; (2) it prevents the creation of redundant crossings; (3) it distributes the desired number of crossings into the upper region.

Note that the size of the ill combination graph, the total size of the ill-combination sorted lists are at most $O(n^2)$. So after at most $O(n^2)$ steps, it will stop. Within all the op-

erations in the algorithm, we have prevented the creation of redundant crossings, and the crossing number is, obviously, satisfied.

Time complexity analysis is presented in the following. Step 1 needs at most $O(n^2)$ time to construct the ill combination graph and also at most $O(n^2)$ time to construct the entire ill combination sorted lists. Step 2 needs $O(n \log n)$ time to construct the tree. Steps 3.3 and 3.4 have time complexity $O(n \log n)$. Step 3.5.3 seems a bottleneck of the algorithm. We can use a B⁺tree-like data structure to implement the left-numbered tree, thus we can find successor in constant time. With the data structure and the position of b_i , we can use sequential search to find position of e . Summing all these steps up the total time spent is proportional to the number of crossings, which is bounded by $O(n^2)$. For every terminal, we insert it into left-numbered tree at most once and each takes $O(\log n)$ time. Therefore, this step takes, in total, $O(n^2)$ time for Step 3.5.3 and $O(n \log n)$ time for the other operations. Step 4 takes linear time. Thus, the whole algorithm has a time complexity of $O(n^2)$.

Lemma 5 The CDP on $Q_3(Q_1)$ can be solved in $O(n^2)$ time when only I-type, U-type, and M-type nets are present. \square

Note that the position of the middle terminal of a Y-type net must be placed in such a way that no redundant crossing can be created. For instance, if the position of the middle terminal of a Y-type net is fixed as shown in Fig. 32, a straightforward routing may create a redundant crossing. We can adopt the strategy (delay processing) mentioned previously to obtain the routing shown in Fig. 32 (c) in $O(n^2)$ time.

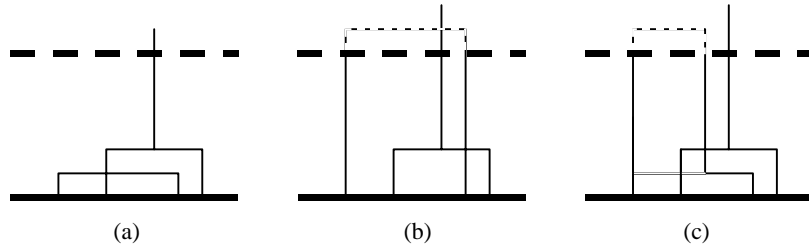


Fig. 32. (a) A situation that might cause the redundant crossings. (b) An example of a redundant crossing. (c) A solution to prevent the problem.

Lemma 6 The CDP on $Q_3(Q_1)$ can be solved in $O(n^2)$ time when the middle terminal positions of Y-type nets are fixed. \square

Theorem 3 The three-terminal CDP problem for n nets with two regions can be solved in $O(n^2)$ time, when the middle terminal positions of Y-type nets are fixed.

Proof: By Lemmas 3, 4, 5, and 6, the time complexity is $O(n^2)$. \square

7. CONCLUSION AND FUTURE WORKS

We have considered the crossing distribution problem (CDP) with n two-terminal nets in two regions and presented an $O(n \log n)$ time algorithm to solve this problem, improving upon a previously known result [6] that takes $O(n^2)$ time. We have also solved the CDP problem for three-terminal nets with two regions in $O(n^2)$ time, when the positions of middle terminals of Y-type nets are given. The CDP problem for multi-terminal nets with more than three terminals remains to be solved.

By introducing more cutting lines, B_1, B_2, \dots, B_t , we can solve the CDP by distributing all C crossings in regions R_1, R_2, \dots, R_{t+1} each containing k_1, k_2, \dots, k_{t+1} crossings, respectively, in $O(t * n \log n)$ time by repeatedly solving the two-region CDP problem t times if the *region adjacent graph* [5] is a path. When the routing region is not a simply connected region, i.e., it has holes, whose boundaries also contain terminals, the CDP problem becomes much harder to solve. Whether one can find a more efficient algorithm than the one previously proposed, which makes use of the Max-flow model [5], remains to be seen.

The following crossing minimization problem arising from the study of 3-terminal nets is of interest. Given n two-sided 3-terminal nets, find the positions of the middle terminal of these 3-terminal nets so that the total number of crossings is minimized.

ACKNOWLEDGMENT

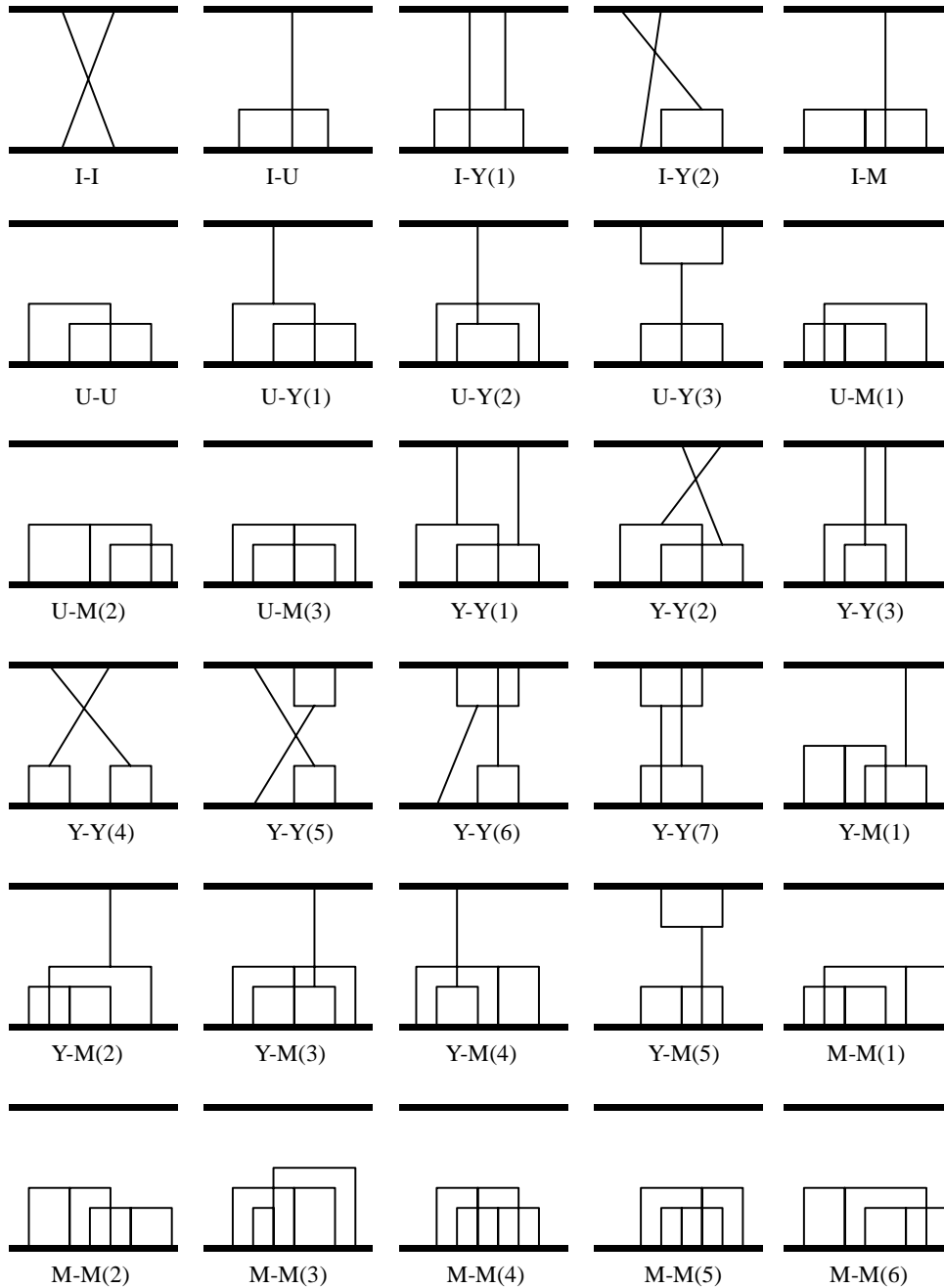
This paper was supported in part by the National Science Council under grants No. NSC91-2219-E-001-001 and No. NSC91-2219-E-001-002. A preliminary version of this paper was also presented at International Computer Symposium 2002, held in Taiwan [8].

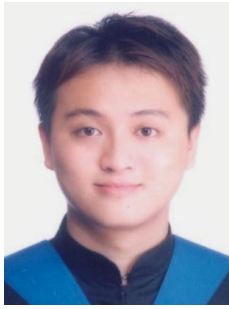
REFERENCES

1. H.-F. S. Chen and D. T. Lee, "On crossing minimization problem," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, 1998, pp. 406-418.
2. G. Cramer, *Introduction L'analyse des Lignes Courbes Algébriques*, Geneva, Switzerland, 1750.
3. P. Groenveld, "On global wire ordering for macro-cell routing," in *Proceedings of the 26th ACM/IEEE Conference on Design Automation*, 1989, pp. 155-160.
4. D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley Longman Publishing Company, Inc., Reading, 1973.
5. M. Marek-Sadowska and M. Sarrafzadeh, "The crossing distribution problem," *IEEE Transactions on Computer-Aided Design*, Vol. 14, 1995, pp. 423-433.
6. X. Song and Y. Wang, "On the crossing distribution problem," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 4, 1999, pp. 39-51.
7. D. C. Wang and C. B. Shung, "Crossing distribution," in *Proceedings of the European Conference on Design Automation*, 1992, pp. 354-361.
8. T. K. Yu and D. T. Lee, "An $O(n \log n)$ algorithm on the crossing distribution problem," in *Proceedings of International Computer Symposium 2002*, Hualian, Taiwan, 2002.

APPENDIX A

We have four types of nets, I, U, Y and M. Let the type I-Y refer to a crossing contributed by an I-type net and an Y-type net. All the crossing types of three-terminal nets are shown below (the symmetric cases are omitted):





T. K. Yu (游騰楷) received a B.S. and a M.S. in computer science and information engineering from National Taiwan University in 2001 and 2003 respectively. He joined the Institute of Information Science of Academia Sinica since July 2003 as a research assistant. His interests include computational geometry, combinatorial optimization, graph theory and bioinformatics.



D. T. Lee (李德財) received his B.S. degree in Electrical Engineering from the National Taiwan University in 1971, and the M.S. and Ph.D. degrees in Computer Science from the University of Illinois at Urbana-Champaign in 1976 and 1978 respectively. Dr. Lee has been with the Institute of Information Science, Academia Sinica, Taiwan, where he is Director and a Distinguished Research Fellow since July 1, 1998. Prior to joining the Institute, he was a Professor of the Department of Electrical and Computer Engineering, Northwestern University, where he has worked since 1978. He spent one year (August 1989 - August 1990) working as Program Director for Computer & Computation Theory Program, Division of Computer & Computation Research of the National Science Foundation. He was a Distinguished Visiting Researcher, Ministry of Education, Culture and Sciences of Japan, in July 1991. His research interests include design and analysis of algorithms, computational geometry, VLSI layout, web-based computing, algorithm visualization, software tools development, compliant controller for active suspension and vibration control, bio-informatics, digital libraries and advanced IT for intelligent transportation systems. He has published over 120 technical articles in scientific journals and conference proceedings, and he also holds three U.S. patents, and one R.O.C. patent. He is Editor of *Algorithmica*, *Computational Geometry: Theory & Applications*, *ACM Journal of Experimental Algorithmics*, Editor-in-Chief of *Journal of Information Science and Engineering*, and Co-Editor-in-Chief of *International Journal of Computational Geometry & Applications*. Dr. Lee is Fellow of IEEE, Fellow of ACM, and President of IICM.