# A Sub-Quadratic Algorithm on the Crossing Distribution Problem

T.K. Yu*  D.T. Lee**

## Abstract

VLSI layout design is typically decomposed into four steps: *placement*, *global routing*, *routing region definition*, and *detailed routing*. The crossing distribution problem occurs prior to detailed routing[1][2][3][4]. It is much more difficult to rout nets with crossings than not. We will use at least two layers to implement and require some *vias.* So the crossing distribution problem (CDP) is a very important problem of VLSI layout. In this paper we will firstly solve the simple case: the crossing distribution problem of two regions. In previous research, the best result is an $O(n^2)$ algorithm[4]. The previous approach is bounded by crossing numbers. This paper presents an $O(n\log n)$ algorithm. We don't put our focus on the crossings but on permutation. So the time complexity may get better.

## 1. Introduction

A VLSI circuit is usually modeled to be composed of *modules* and a set of *nets*. Each net specifies a subset of points, called *terminal*s, on the boundary of the modules. The *layout* problem is to interconnect the modules as specified by the nets in terms of different technological design rules. Normally the *layout* problem is solved in four steps[4]:

***Placement****:* the modules are placed on the plane.

***Global routing****:* the routing region is partitioned into simple subregions, each called *an elementary regio*n, 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-shape channels. Channels have to be ordered properly such that their size can be adjusted without rerouting the previous completed channels.

* Department of Computer Science and Information Engineering, Graduate School of National Taiwan University, Taipei, Taiwan. E-mail: r90091@csie.ntu.edu.tw

**Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan.

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

***Detail routing:*** detailed wirings of the individual routing regions are given.

An arbitrary VLSI layout (see figure 1-1) consists of some *modules*, *regions* and *nets*. In Figure 1-1 a sketch of a global routing is shown, in which each net is specified by a list of regions through which its wiring passes. For instance for net 5, the sequence of regions is (*Rg, Rc, Rd*). 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, is 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 left to right for each horizontal boundary edge and bottom to top for each vertical boundary edge, is fixed, the number of crossings for each region is thus determined. Indeed, crossings in VLSI layout imply the usage of vias and require one or more layers in detailed routing [2]. As vias take up routing area, the number of vias allowed for each routing region may have to be restricted. The crossing distribution problem (CDP) calls for a 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[1]; Marek-Sadowska and Sarrafzadeh [2]; Wong and Shung [3]; and by Song and Wang[4]. In [2] Marek-Sadowska and Sarrafzadeh presented an O(MlogM) algorithm for solving a CDP, where M is the number of non-redundant crossings of the global routing. Song and Wang[4] presented an $O(n^2)$ algorithm, where n is the number of nets for CDP of two-terminal nets between two regions. In this paper we study the crossing distribution problem of two-terminal nets between two regions and present an O(n log n) algorithm. The result can be generalized to cases where there are multiple regions.

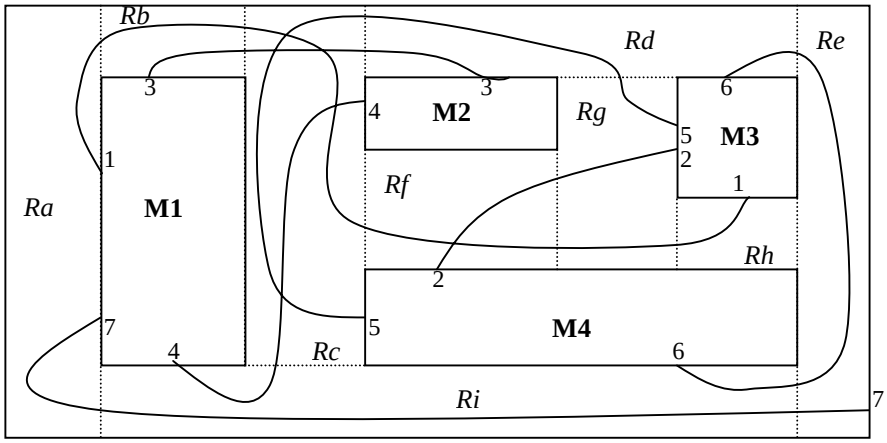


Figure 1-1 An example of VLSI layout.

## 2. Problem Definition

Without loss of generality we 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 figure 2-1. Imagine 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 of figure 2-1.

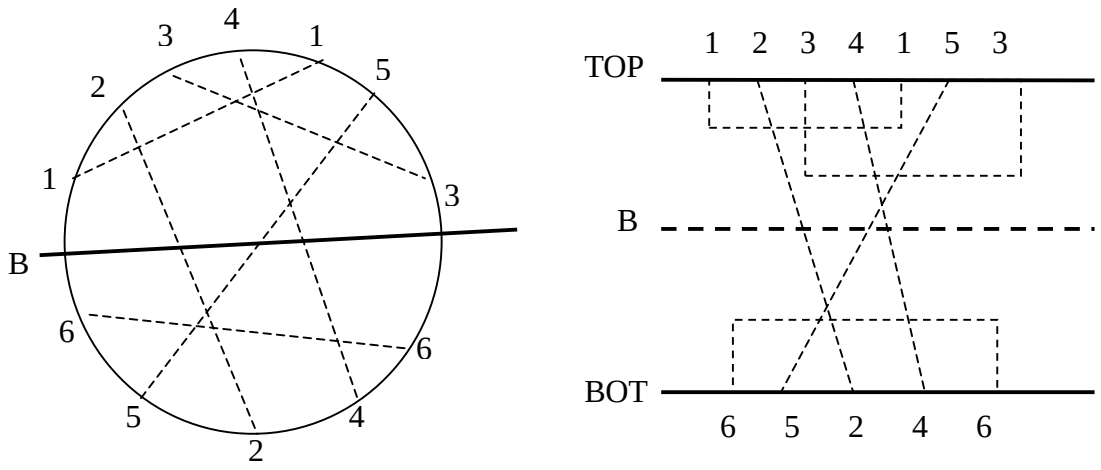


Figure 2-1 The crossing distribution problem of two regions.

Let TOP and BOT be two horizontal lines on which terminals are placed. A two-terminal net N = (p, q) is *two-sided* if N has a bottom terminal p on BOT and a top terminal q on TOP. A two-terminal N = (p, q) is one-sided if both p and q are on a line BOT (or TOP). A *crossing* is an intersection of two different nets. We distinguish the *inherent (necessary* or *force*d) crossings and *redundant* crossings. Intuitively, an inherent crossing between two nets is the one that cannot be removed by a connection homotopy[2], Consider two nets a and b, three different types of inherent crossings between a and b are shown in Figure 2-2. Some redundant crossings are shown in Figure 2-3.

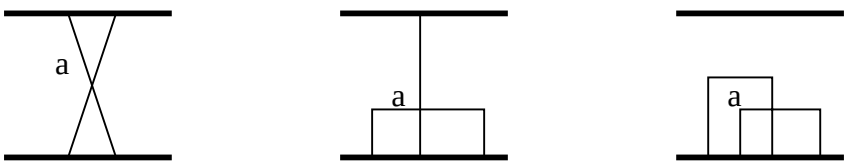


Fig.2-2Three types of inherent crossings between a and b.

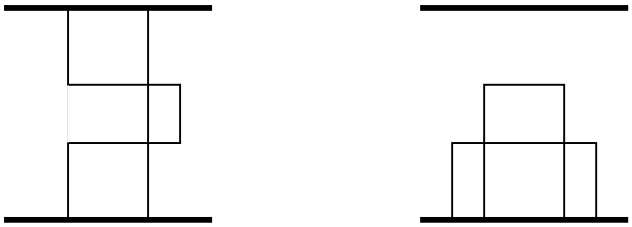

Figure 2-3 some redundant crossings

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 R1 and R2 denote two routing regions sharing a common boundary B. Given a global routing of N 2-terminal nets whose terminals are located on the boundary of the routing regions R1 and R2 respectively. Let C denote the total minimal number of crossings that exist in R1 and R2, and K an integer ($K \leq C$). The 2-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 R1, and C-K crossings are located in R2. As mentioned before, we assume that the boundary of the two adjacent routing regions R1 and R2 can be represented by a circle whose circumference contains the terminals. Furthermore we assume that the top part of the circle refers to region R1 and the bottom part to region R2. Fig. 2-1 gives an illustration. Let TOP and BOT represent the top and bottom boundary, respectively, that contain an ordering of the nets whose global routes are dissected by boundary B.

The crossing distribution problem in two regions R1 and R2 is to determine 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 following:

*Problem I.* Let $\Psi = \{N1, N2, ..., Nn\}$ be a set of $n(n \geq 1)$ nets. Let TOP = {a1,a2, ..., au} and BOT = {b1,b2, ..., bv} be two sequences of terminals of the nets on the top and bottom lines, respectively, where ai(or bj) 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 and BOT. Given a boundary B and an integer quota K( $K \leq C$, C is the total number of crossings), distribute exactly K crossings to R1 and C-K crossings to R2. A net is said to be two sided if its two terminals lie on different sides, TOP and BOT respectively. It is said to be one-sided, if both terminals lie on the same side, TOP or BOT.

Let P denote the set of crossings in the global routing in two regions. Let X, Y, Z denote the set of one-sided nets on TOP , two-sided nets and one-sided nets on BOT, respectively. The set S of nets is thus partitioned into three pairwise disjoint subsets, X, Y, and Z. That is, we have $S = X \cup Y \cup Z$, $X \cap Y = \varnothing$, $X \cap Z = \varnothing$, and $Y \cap Z =$

∅. Let (A, B) denote the crossing between nets A and B. Define the following three sets of crossings:

$P_1 = \{(A, B) \mid ((A \in X) \wedge (B \in X)) \vee ((A \in Y) \wedge (B \in X)) \vee ((A \in X) \wedge (B \in Y))\}$

$P_2 = \{(A, B) \mid (A \in Y) \wedge (B \in Y)\}$

$P_3 = \{(A, B) \mid ((A \in Z) \wedge (B \in Z)) \vee ((A \in Y) \wedge (B \in Z)) \vee ((A \in Z) \wedge (B \in Y))\}$

Figure 2-4 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 these three case 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 $P_3$).

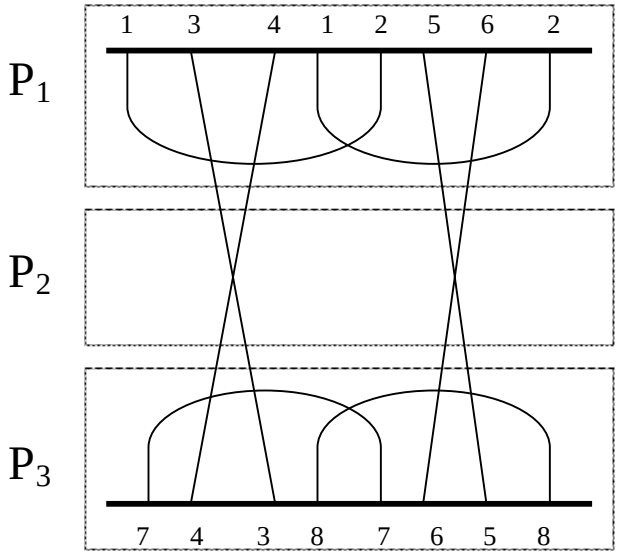


Figure 2-4 Three types of crossing $P_1$, $P_2$ and $P_3$

With the definition above, we can divide *problem 1* into *problem 2* and *problem 3* below:

*Problem II. Crossing distribution for two-sided nets* ($P_2$). Let $\Psi = \{N_1, N_2, ..., N_n\}$ be a set of $n(n \geq 1)$ two-sided nets. Let $TOP = \{a_1,a_2, ..., a_n\}$ and $BOT = \{b_1,b_2, ..., b_n\}$ be two sequences of terminals of the nets of $\Psi$ on the top and bottom lines, respectively, 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 R1 and C–K crossings are located at R2.

*Problem III. Crossing distribution with one-sided* (Figure 4 $P_1$,$P_3$). Let $\Psi = (N_1, N_2, ..., N_n)$ be a set of $n(n > 1)$ nets such that $N_i$ ($i = 1, ..., n$) is either an one-sided net on a line G or a two-sided net having a terminal on G. Let L = (t1,t2, ..., tr), $r \leq 2n$, be the sequence of terminals of the nets in $\Psi$ on G. Given a boundary B and an integer quota

K(K ≤ C, C is the total number of crossings), distribute exactly K crossings above B.

## 3. Crossing Distribution for Two-Side Nets

Two-sided nets $N_i = (p,q)$, $N_j = (r,s)$ are crossing iff ($p<r$ and $q>t$) or ($p>r$ and $q<t$). Without loss of generality, we assume *TOP* = (1,2,3,…,n) and *BOT* is a permutation of (1, 2, 3, …, n ). In Song and Wang's paper[4], they note that the crossing number is equal to the inversion[5][6] number of the permutation. For instance, (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 $s_j$ is larger than $s_i$. For each integer $i$ in (1, 2, …, 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 integers lying to the left of $i$, then the number of inversions induced by integer $i$ is $i$ -1- m. We call the number m as magic number for $i$. We will discuss this in more detail in section 4.

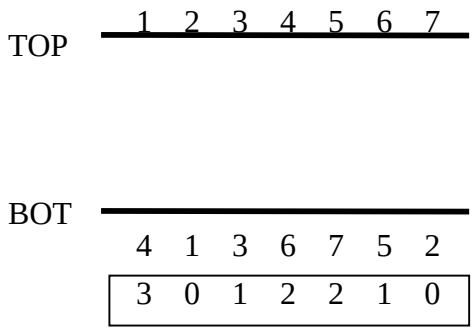

Figure 3-1 crossing number with net's number less than it

Consider Figure 3-1. 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 Figure 3-1 are the inversion numbers induced by the integer immediately above.

How to count this number efficiently is left to next section. Now we consider the lemma below:

*Lemma 1*: The sum of all the magic numbers of a permutation of (1,2,…,n) and the inversion number is equal to n(n-1)/2.

*Proof*: Obvious. □

As will be shown later, CDP for two-sided nets can be solved easily once we compute the magic number for each integer. We now give an algorithm to compute a

permutation of (1,2,…,n) whose inversion/crossing number is k≤ C, where C is the total number of inversions of a given permutation *BOT* of (1,2,…,n). We build a list 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 crossings are located at $R_2$ (in the lower region). We also maintain a pointer array P-ary to every node of B-list. If we output a node from the B-list, we delete it at the same time.

*Algorithm 1: Crossing distribution for two-sided nets*
*Input: k an integer; BOT=($b_1$,…,$b_n$)*
*Output: a sequence of numbers*
1. Build *B-list* as (1,2,…,n) and pointer array *P-ary*, one for each entry in *B-list.*
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 figure 3-2 the initial *B-list* is the same as *TOP*, we assume k= 5 and that the list of inversion numbers for 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 is set to k-3 = 2 (See figure 3-3).

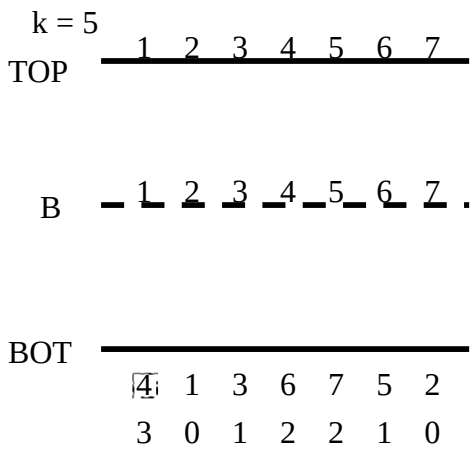


Figure 3-2 Initial situation, assuming magic numbers are given

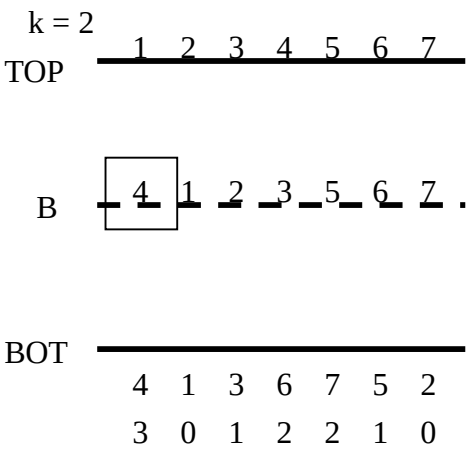


Figure 3-3 After an iteration, 4 has been output, and k becomes 2

In figure 3-3 the square containing 4 on B is the current output sequence. Let us pause for a moment here. In the initial sequence routing of the first four nets as shown in Figure 3-4 causes four crossings in $R_2$. In Figure 3-5 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 will increase or decrease the total number of inversions by one [6]. Now the next element is 1 and its inversion number is 0. We do nothing but output this number (Figure 3-6). Next we find 3 and its inversion number 1 (Fig.3-7). Since k = 2 > 1 = r, we output 3 and update k to be 1. Now we have four crossings in $R_1$.

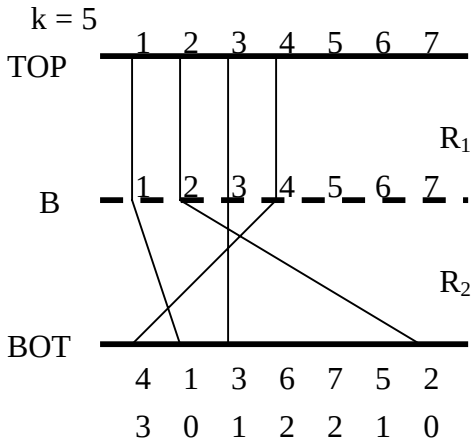


Figure 3-4 four crossings in $R_2$

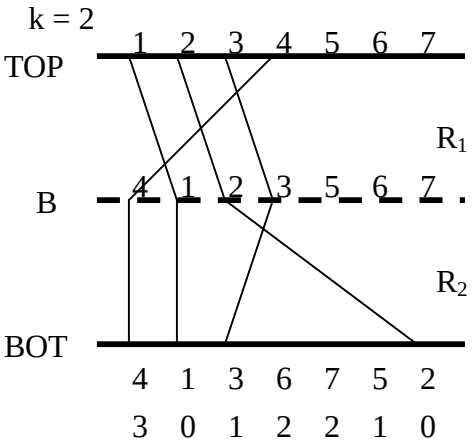


Figure 3-5 three crossings in $R_1$ and one crossing in $R_2$

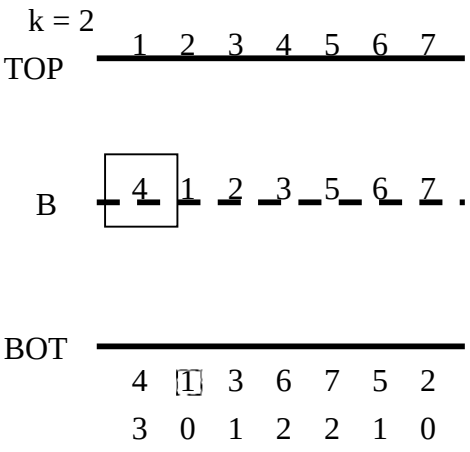


Figure 3-6 find 1

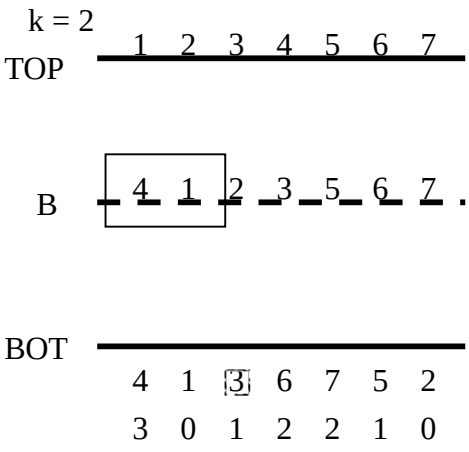


Figure 3-7 find 3

When we find 6, k = 1 < 2 = r, we according to Step 2.3 start to output B-list r-k nodes. It means that we only need to make k exchanges of adjacent positions to meet 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 figure.3-8,3-9). Figure 3-10 is the final routing result, which satisfies the quota requirement and there is no redundant crossing.

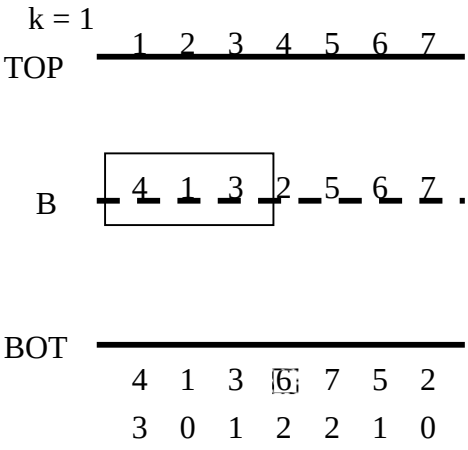


Figure 3-8 find 6 but m > k.

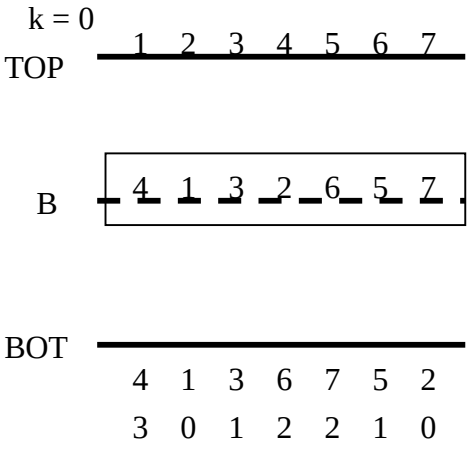


Figure 3-9 final situation.

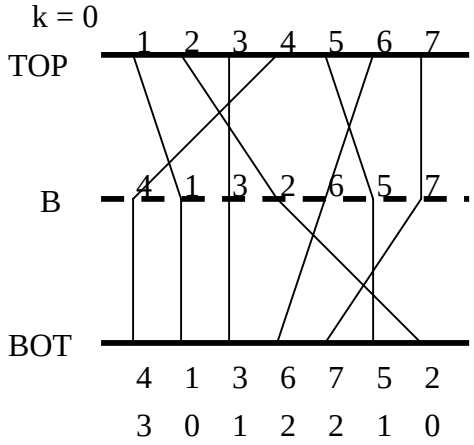


Figure 3-10 totally 9 crossings. 5 in $R_1$, 4 in $R_2$.

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

When k = 0 or when we have scanned the whole BOT list, the algorithm terminates. Since the algorithm does not create redundant crossings (two-sided nets by default have no redundant crossing) and it satisfies the quota[6], so 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 in linear. As we will show below the time needed to compute the magic numbers and hence the inversion numbers is O(n log n).

# 4. Finding Magic Number Efficiently

To find the magic number of bi, we should find out how many numbers are less than bi which occur before bi 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 bi. So if we could find the magic number in O(log n) time, the total time complexity would be O(nlogn).

Since this problem is rather straightforward, we illustrate this 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 left-number(v). When a left subtree of a node v is followed, left-number(v) is incremented by 1 (indicating that the newly inserted number is less than 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.

See Figure 4-1, *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.

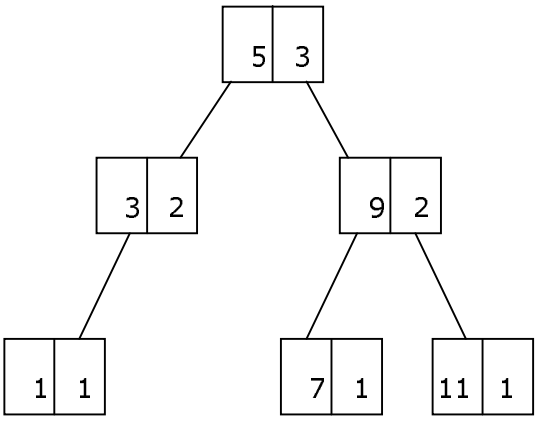


Figure 4-1 an example of left-numbers in an AVL tree

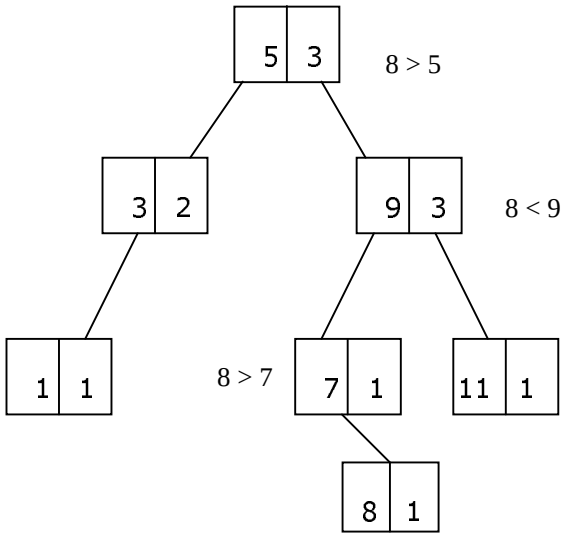


Figure 4-2 after inserting 8 to AVL tree

See Figure 4-2. When we insert 8 into this tree, since 8 > 5, we follow the right subtree, 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 correct place and add its left-number 1 to obtain its final magic number, which is 4.

Figure 4-3 shows an LL rotation and illustrates how the left-numbers of the nodes A, B, and C should be updated when the height balance operation is performed. In LL rotation (Figure 4-3) the left-numbers of A, B, C initially are a+b+c+3, a+b+2 and a+1, respectively. After rotation, the left-numbers of A, B, C become are c+1, a+b+2 and a+1 respectively. That is, we can set A' = A – B, B' = B, C' = C. Other rotations are similar.

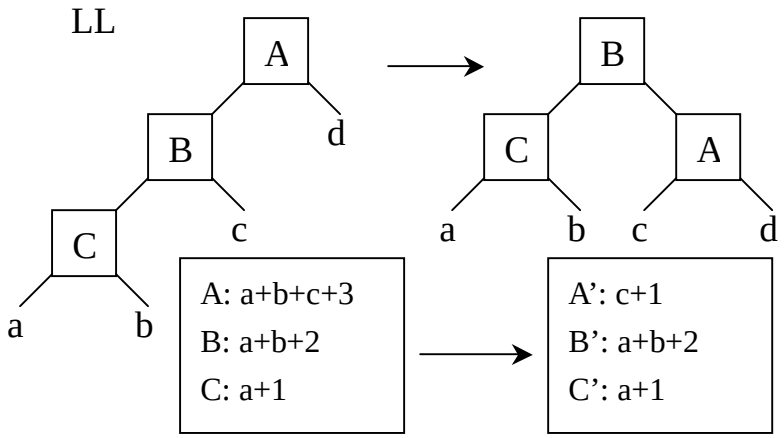


Figure 4-3 LL rotation

We will present an algorithm below, whose correctness is immediate.

*Algorithm 2: Find the magic number from a left-numbered tree*
*Input: a node p; a left-numbered height-balanced tree T*
*Output: magic number m of p*
1. m=0; u = T.root
2. *if* p.key > u.key
    *then* m = m + u.left-number
        u = u.right
    *else* u = u.left
3. *if* u = null
    *then* return m
    *else* goto 2

# 5. Crossing Distribution Problem for One-Sided Nets

We will only discuss $P_3$ here (one-sided net on BOT), The method for $P_1$ is similar. In Song and Wang's paper [4], they enumerate all crossings and use 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 crossings to solve this problem.

In this section we also use the left-numbered height-balanced tree to solve this problem. We make some observations to help us understand this problem better. Figure 5-1 shows an example of one-sided net routing, and $N_3$, $N_5$ and $N_7$ are two-sided nets, 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 left and right terminal of the net.

*Definition:* Net $N_i$ is said to contain Net $N_j$, if and only if $Begin(N_i) < Begin(N_j) < End(N_j) < End(N_j)$ (In figure 5-1 $Begin(N_1)$ is 1, $End(N_1)$ is 7, and $N_1$ contains $N_2$.).

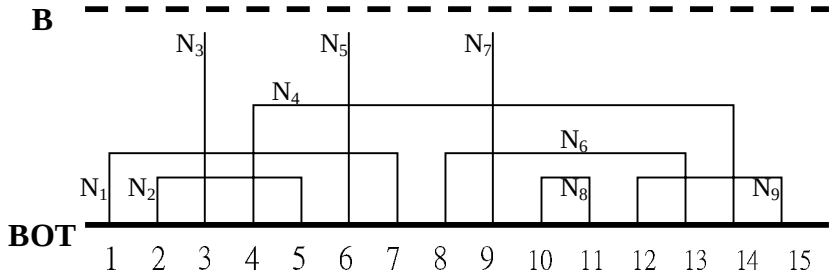


Figure 5-1 an example of one-sided net routing

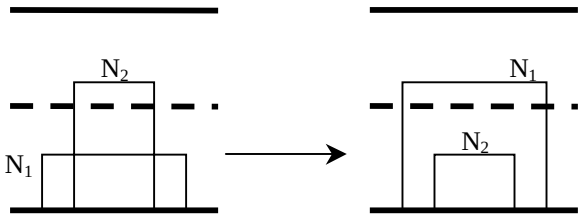


Figure 5-2 to prevent redundant crossings

In figure 5-2 $N_1$ contains $N_2$. If we route $N_2$ in the upper region and $N_1$ in the lower region, two redundant crossings would occur. So if $N_i$ contains $N_j$, we should always route $N_i$ to upper region before $N_j$ to prevent redundant crossings.

*Lemma 2:*

If $Begin(N_i) < Begin(N_j)$, $N_j$ cannot contain $N_i$.

*Prrof:*

It is proved by definition. □

With lemma 2, we can use the order of left terminal to solve this problem. We will scan the BOT list once to find some information: 1. begin time of all one-sided

nets, 2. end time of all one-sided nets, and 3. positions of all two-sided nets. Method for finding these information is scanning the BOT one time, it needs linear time.

After finding these 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 for one-sided nets.*
*Input: BOT = {$b_1$,$b_2$,…,$b_n$}; k integer quota.*
*Output: B = {$c_1$,$c_2$,…,$c_m$} as a permutation on B*

1. find the information of all nets about their positions.
2. build left-numbered height-balanced tree T by two-sided nets
3. scan BOT from $b_1$ to $b_n$
   - 3.1 let current node is $b_i$, and its correspond net called $N_j$
   - 3.2 if ($N_i$ has been processed (we encounter its right terminal) or it is a two-sided net) then continue
   - 3.3 find magic numbers of Begin($N_j$) and End($N_j$) in T, named m1 and m2
   - 3.4 if (m2-m1 ≥ k) then
     insert End($N_j$) to T; Break
     else
     insert Begin($N_j$) and End($N_j$) to T
     k = k – (m2-m1)
4. output nodes in tree T by inorder traversal. After outputting *m2-k* nodes, we output Begin($N_j$) . Then continue output other nodes in tree T.

In line 3.3, the number m1 (respectively m2) denotes the number of nets piercing the boundary lying to the left of Begin($N_j$) (respectively End($N_j$)). So m2-m1 denotes the number of nets piercing the boundary lying between the two terminals of one-sided net $N_j$. For instance, figure 5-3 shows an initial state of a one-sided net routing, and the initial left-numbered tree contains $N_3$, $N_5$ and $N_7$. At first iteration of step 3 we find $N_1$. Using Begin($N_1$) and End($N_1$) to find m1=0, m2=2. Quota k=5 > 2=m2-m1, so we insert Begin($N_1$) and End($N_1$) to the left-numbered tree and decrease k by 2 (figure 5-4).

What this means is that we route net $N_1$ in the upper region $R_1$, we will make m2-m1 crossing 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 to the left-numbered tree.

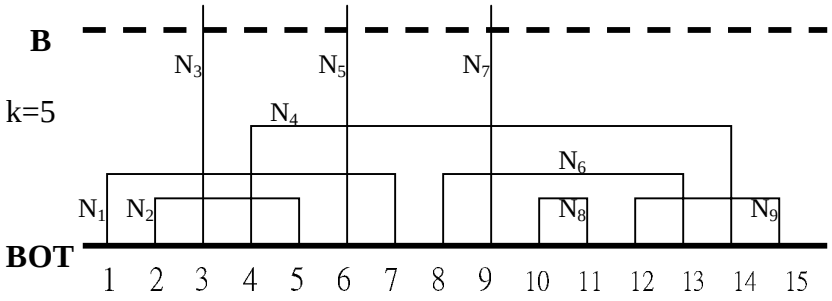


Figure 5-3 initial state

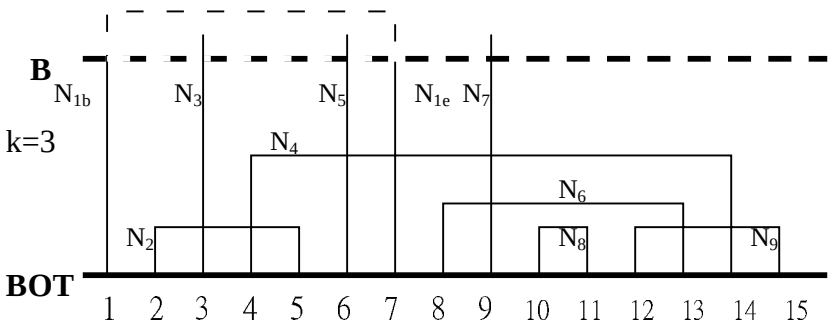


Figure 5-4 after processing $N_1$

On the next iteration, $N_2$ is scanned, and m1=1, m2=3. Since k=3 > 1=m2-m1, Insert Begin($N_2$) and End($N_2$) to the left-numbered tree and decrease k by 1 (figure 5-5).

Next finding $N_4$, we have m1=3, m2=7. k < m2-m1, we insert End($N_4$) into T to obtain the sequence ($N_{1b}$, $N_{2b}$, $N_3$, $N_{2e}$, $N_5$, $N_{1e}$, $N_7$, $N_{4e}$), and go to Step 4. We then output 5 (m2- k) nodes in inorder traversal of T, output Begin($N_4$), and then the rest of nodes in T. That is, the final permutation is ($N_{1b}$, $N_{2b}$, $N_3$, $N_{2e}$, $N_5$, $N_{4b}$, $N_{1e}$, $N_7$, $N_{4e}$) and the routing is shown in Figure 5-6. Initially (fig. 5-3), there are 10 crossings in $R_2$. After running this algorithm, we have now 5 crossings in $R_1$ and 5 crossings in $R_2$.

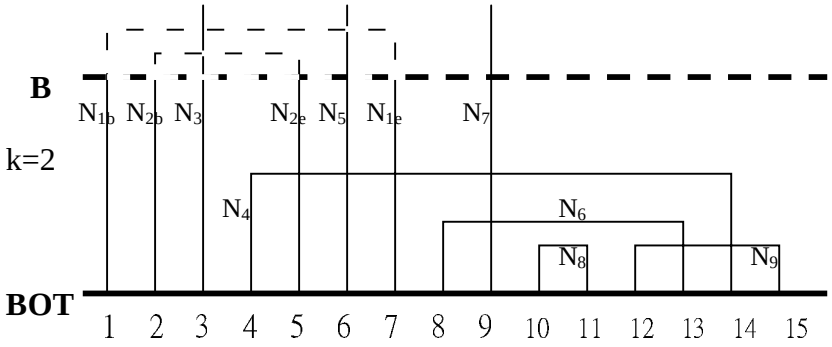


Figure 5-5 after processing $N_2$

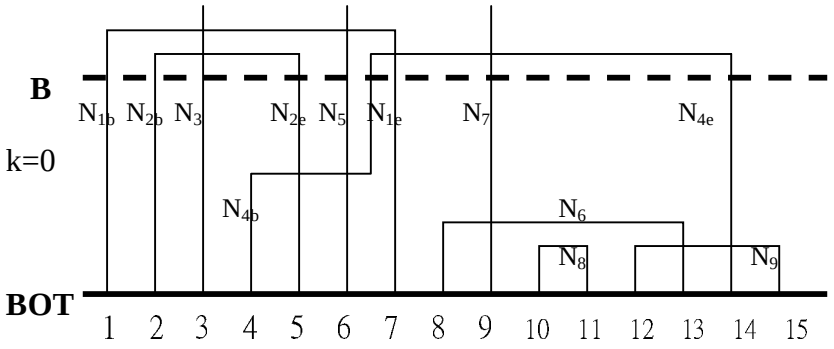

Figure 5-6 final result

*Theorem 2: The one-sided CDP problem for n nets can be solved in O(n log n) time.*
*Proof:*
The algorithm terminates when k ≤ m2-m1 or the entire list BOT is scanned. The quota is not exceeded when k > m2-m1. When k ≤ m2-m1, we output Begin($N_i$) at k+1 positions to the left of End($N_i$) to create k crossings in the upper region to meet the quota requirement.
*Time complexity:*
Step 1, 2 and 4 each need linear time, and step 3 takes O(n log n) time for the operations needed for the left-numbered tree searching and insertion. So totally O(nlogn).

# 6. Conclusion and Future Work

We have considered the crossing distribution problem (CDP) of n 2-terminal nets in two regions and presented an O(n log n) time algorithm to solve this problem, improving upon a previously known result[4] which takes $O(n^2)$ time.

By introducing more cutting lines $B_1, B_2 \ldots, B_t$ we can solve the CDP by distributing all C crossings in regions $R_1, R_2 \ldots, R_{t+1}$ each containing $k_1, k_2 \ldots, k_{t+1}$ =C crossings respectively in time O(t n log n) by repeating the two-region CDP problem t times. When the number of terminals in each net is more than 2, so-called multi-terminal net [2] or the routing region is not a simply connected region, i.e., it has holes whose boundary also contains terminals, the CDP problem becomes a lot harder. Whether one can find a more efficient algorithm than the previous result which makes use of Max-flow model [2] remains to be seen.

**References**


[1] GROENVELD, P, 1989, On global wire ordering for macro-cell routing. In *Proceedings of the 26th ACM/IEEE Conference on Design Automation* (DAC '89, Las Vegas, NV, June 25–29,1989), D. E. Thomas, Ed. ACM Press, New York, NY, 155–160.

[2] MAREK-SADOWSKA,M.AND SARRAFZADEH, M. 1995. The crossing distribution problem,. *IEEE Trans. Comput.-Aided Des. 1*4, 4 (Jan.).

[3] WANG,D.C.AND SHUNG, C. B. 1992. Crossing distribution. In *Proceedings of the European Conference on Design Automatio*n. 354–361.

[4] X. SONG and Y. WANG, 1999, "On the crossing distribution problem," in ACM Press , New York, NY, 39 – 51.

[5] CRAMER, G. 1750. *Introduction l'analyse des lignes courbes algèbrique*s. Geneva, Geneva, Switzerland.

[6] KNUTH, D. E. 1973. *The Art of Computer Programmin*g. Addison-Wesley Longman Publ. Co., Inc., Reading, MA.