VLSI DESIGN
1996, Vol. 4, No. 1, pp. 41-51
Reprints available directly from the publisher
Photocopying permitted by license only

An Efficient Algorithm for the Split *K*-Layer Circular Topological Via Minimization Problem

J. S. HUANG
Advanced Technology Center, Computer & Communication Research Laboratories, Industrial Technology Research Institute. E000, Bldg. 11, 195 Sec. 4, Chung Hsing Rd, Chutung, Hsinchu, Taiwan 31015

Y. H. CHIN†
Institute of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan

(Received September 18, 1992, Revised December 13, 1993)

The split k-layer ($k \geq 2$) circular topological via minimization (k-CTVM) problem is reconsidered here. The problem is finding a topological routing of the n nets, using k available layers, such that the total number of vias is minimized. The optimal solution of this problem is solved in $O(n^{2k+1})$ time. However, such an algorithm is inefficient even for $n \geq 8$ and $k \geq 2$. A *heuristic* algorithm with complexity of $O(k\, n^4)$ is presented. When the experimental results of this algorithm and that of an exhaustive algorithm are compared, the same number of optimal solutions is obtained from this heuristic algorithm for all permutations of 1) $n = 8$ with $k = 2$ or 3, and 2) $n = 10$ with $k = 3$. For other cases, the number of optimal solutions from this algorithm depends on the permutations been selected; and this number, in general, will increase as k increases.

Key Words: *Algorithm, Circular Channel, Cyclic Increasing Sequence, Layers, VLSI Layout, via Minimization*

1 INTRODUCTION

For the purpose of increasing both the yield and the circuit performance of the VLSI chip, the number of vias being used in a VLSI layout should be kept as few as possible. Thus, via minimization is an important issue in the routing problem of a VLSI layout. There are two approaches to minimizing the number of vias: *constrained via minimization* and *topological via minimization*. The theoretical study of the constrained via minimization problem has been completely resolved from the standpoint of maximum junction degree [9].

The topological via minimization, the other approach to the routing problem, was first proposed by Hsu [4]. This topic is interested in many practical or theoretical researchers [2], [4], [6], [8], [10], [11], [12], [14], [15]. The k-layer topological via minimization (k-TVM) problem is formally defined as follows. A set of nets on a k-layer ($k \geq 2$) routing region R is given. Each net is a set of terminals to be electrically connected. The problem is to route the terminals of each net by a set of wire segments, in which each wire segment is assigned to a layer and vias are formed to connect every two adjacent wire segments on different layers. Accordingly, no two wire segments representing two different nets intersect on the same layer, and the number of vias is minimized. The wires and the vias assume infinitely small widths and sizes, respectively. Moreover, there are no shape restrictions for wires and vias (hence name *topological* for the problem).

The k-TVM problem has been shown [2], [9], [11] to be NP-hard [3], even when the routing region R is a switchbox with only two-terminal nets. However, several special cases of the problem can be solved in polynomial time [2], [6], [10], [11], [12]. One of these cases, called a circular *permutation* channel [10], is reexamined here. In this case, the routing region R is a circular channel with n two-terminal nets; each two-terminal net has one terminal located on the inner circle and the other terminal located on the outer circle of the circular channel. Such an k-TVM problem on a circular permutation channel was studied by Rim *et al.* [10], where it is referred to as a split k-layer circular topological via minimization (k-CTVM) problem. Based on a dynamic programming approach, they proposed an algorithm of time complexity

†To whom all correspondence should be addressed. This work was supported by the Minister of Economic Affairs under Grant 37H3100.

$O(n^{2k+1})$ to solve this problem. However, such a polynomial time algorithm may not be efficient enough, even for $k = 2$, since the net number n in a VLSI layout can be very large. A heuristic algorithm with the time complexity of $O(k\, n^4)$ is presented. The proposed algorithm is based on the strategy namely, "generating a single-layer routable subset at a time while keeping the length of the longest cyclic decreasing subsequence of the remaining sequence as short as possible."

The paper is further organized as follows. Section 2 lists some definitions and preliminary results. Section 3 presents the heuristic algorithm. The correctness and time complexity of the algorithm are given in Section 4. Section 5 summarizes the experimental results. Finally, Section 6 concludes the paper.

2 PRELIMINARIES

Without loss of generality, the n terminals on the outer circle of a circular permutation channel can be numbered clockwise as $1, 2, \ldots, n$. The n terminals (also numbered clockwise) on the inner circle of the circular permutation channel can be regarded as a permutation of the numbers $1, 2, \ldots, n$. Let the permutation be $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. The two terminals of each net i are represented as the terminal i on the outer circle and the terminal $\pi(j) = i$ on the inner circle for some j. Let π_s denote *cyclic* permutation by circularly shifting s positions of π counter-clockwise, i.e., $\pi_s(i) = \pi(s + i)$, $\forall\ 1 \le i \le n - s$, and $\pi_s(i) = \pi(s + i - n)$, $\forall\ n - s < i \le n$. Let S be a subset of the n nets of the circular permutation channel. The following two lemmas were shown by Rim *et al.* [10].

Lemma 1. [10]. S is single-layer routable, if and only if the set of corresponding terminal numbers of S on the inner circle (clockwise) is an increasing subsequence of some cyclic permutation π_s, $1 \le s \le n - 1$.

Lemma 2. [10]. Let m be an integer such that $0 \le m \le n$. The n nets can be routed on k layers using m vias, is and only if there exists k disjoint single-layer routable subsets $S_1, S_2, \ldots, S_k$ of the m nets such that $\sum_{i=1}^{k} |S_i| = n - m$.

Rim *et al.* [10], then, showed that the split k-CTVM problem, by Lemma 2, can be solved by finding k disjoint single-layer routable subsets $S_1, S_2, \ldots, S_k$ of the n nets such that $\sum_{i=1}^{k} |S_i|$ is maximized. The nets in each single-layer subset S_i are routed entirely in the i-th layer, $i = 1, 2, \ldots, k$. For those nets that are not in the k single-layer routable subsets, by using two adjacent layers one via is sufficient for routing each net [8], [10]. The via number m is minimized since the given net number n is fixed and $\sum_{i=1}^{k} |S_i|$ is maximized.

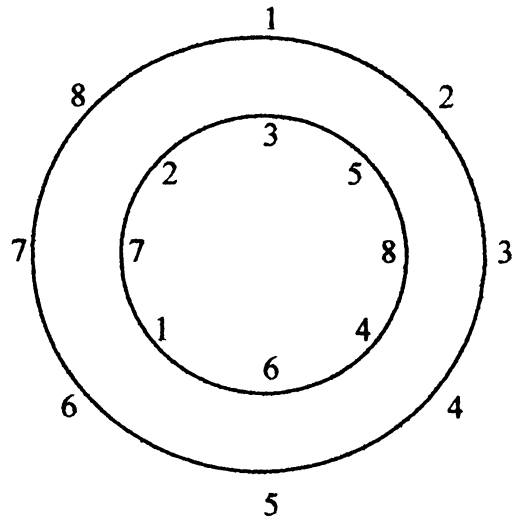
An instance ($n = 8$) of the k-CTVM problem with $k = 2$ is shown in Figure 1(a), where the permutation is given as $\pi = (3, 5, 8, 4, 6, 1, 7, 2)$. An optimal solution for the instance without any via is shown in Figure 1(b). The two disjoint single-layer routable subsets $S_1 = (1, 2, 3, 5, 8)$ and $S_2 = (4, 6, 7)$ are routed by layer 1 wire segments and layer 2 wire segments, respectively. A possible non-optimal solution for the instance with one via is shown in Figure 1(c), where the two disjoint subsets of nets $S_1 = (1, 2, 3, 4, 6)$ and $S_2 = (5, 7)$ are single-layer routable subsets. The nets in S_1 are routed by layer 1 wire segments and the nets in S_2 are routed by layer 2 wire segments. The net 8 is neither in the subset S_1 nor in the subset S_2; however, it is routed by the wire segment on layer 1 and the wire segment on layer 2, both of which are connected by one via.

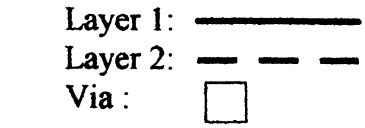
Some terms are needed to be explained as follows. An increasing (decreasing) subsequence of π_s for some s, $0 \le s \le n - 1$, is called a *cyclic increasing (decreasing) subsequence* (CI(D)S) of π. Note that a CIS or CDS of π has a length of at least 2. The following lemma is trivial by Lemma 1.

Lemma 3. The maximum single layer routable subset of the n two-terminal nets on the circular permutation channel is a *longest* CIS (LCIS) of π.

The LCIS problem is to find an LCIS of π, i.e., a CIS with a maximum cardinality among all cyclic increasing subsequences of π. This paper's authors have recently solved this problem in $O(n \log (t + 1) + n\, t)$ time, where t is the size of the LCIS. For example, the LCIS of $\pi = (3, 5, 8, 4, 6, 1, 7, 2)$ is $(1, 2, 3, 4, 6)$, $(1, 2, 3, 5, 6)$, $(1, 2, 3, 5, 8)$, or $(2, 3, 4, 6, 7)$. For completeness, the LCIS algorithm is given in Appendix together with the proof of the correctness and the time complexity analysis.

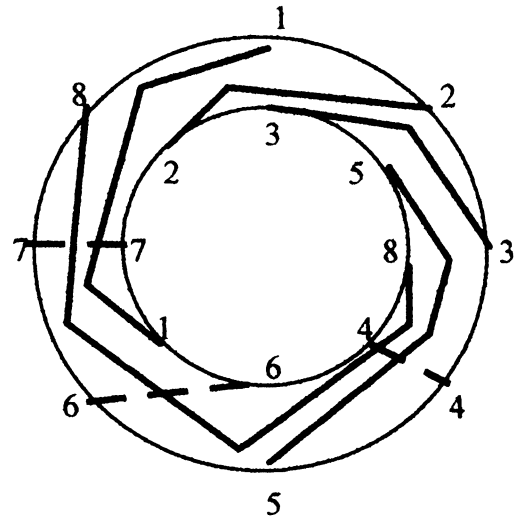
Returning to the split k-CTVM problem. Lemma 1 and the definition of CIS suggest that the problem is equivalent to finding k CIS's $S_1, S_2, \ldots, S_k$ of π such that $\sum_{i=1}^{k} |S_i|$ is maximized. A simple and straightforward approach to the problem is to apply the LCIS algorithm k times. This approach has a time complexity of $O(k\, n^2)$. However, such an approach may not generate an optimal solution. Consider the instance in Figure 1(a) with $k = 2$ as an example. If an LCIS $S_1 = (1, 2, 3, 4, 6)$ for $\pi = (3, 5, 8, 4, 6, 1, 7, 2)$ is first obtained by the LCIS algorithm, then the remaining permutation is $(5, 8, 7)$ after removing the LCIS S_1 from π. The second and final LCIS S_2 to be found is $(5, 7)$ from the remaining permutation $(5, 8, 7)$. This solution requires one via for the un-selected net 8 as shown in Figure 1(c), which is not an optimal solution. Therefore, the following heuristic algorithm is designed, which may provide a more promising solution.





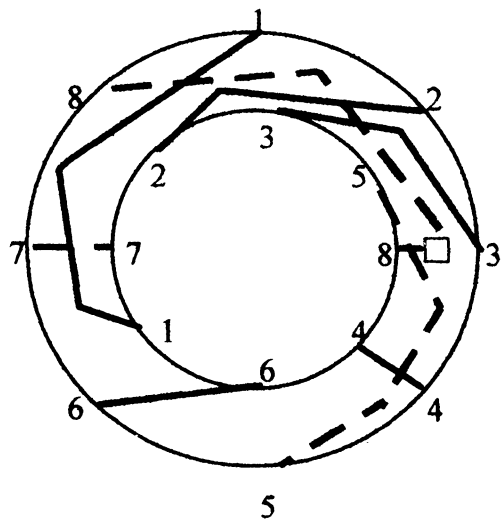
$\Pi = (3, 5, 8, 4, 6, 1, 7, 2)$ with LCDS $= (7, 5, 4, 1)$

(a)



$S_1 = (1, 2, 3, 5, 8), S_2 = (4, 6, 7)$

(b)



$S_1 = (1, 2, 3, 4, 6), S_2 = (5, 7)$

(c)

FIGURE 1 (a) An instance of the split k-CTVM problem with $k = 2$. (b) An optimal solution without vias for the instance. (c) A possible solution with one via for the instance.

3 THE ALGORITHM FOR THE SPLIT *K*-CTVM PROBLEM

Let the length of a longest CDS (LCDS) of the π be ℓ^r. For an LCDS of length 2, the minimum layer number for routing the corresponding two nets is one. In this case, all the n nets in the circular permutation channel can be routed in one layer since $\pi_i = (1, 2, \ldots, n-1, n)$ exists for some integer i. Furthermore, the minimum layer number of two for routing the corresponding nets in an LCDS of length 3 or 4 can be easily verified. By induction, the corresponding nets in an LCDS of length ℓ^r require $\lceil \ell^r/2 \rceil$ layers to route them. All the other nets which are not in the LCDS may use either the existing $\lceil \ell^r/2 \rceil$ layers or new layers to realize them. A via-free routing solution for the circular permutation channel requires at least $\lceil \ell^r/2 \rceil$ layers.

Since a CDS of $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ is a CIS of the reverse permutation $\pi^r = (\pi(n), \pi(n-1), \ldots, \pi(1))$, the problem of finding an LCDS of π is equivalent to finding an LCIS of π^r, which is solvable in $O(n \log(t+1) + nt)$ time (see Appendix), where t is the size of the LCDS of π. For example, the circular permutation channel instance in Figure 1(a) has the permutation $\pi = (3, 5, 8, 4, 6, 1, 7, 2)$. The reverse permutation is $\pi^r = (2, 7, 1, 6, 4, 8, 5, 3)$. An LCIS of π^r is (1, 4, 5, 7); therefore, the corresponding LCDS of π is (7, 5, 4, 1) with a length of $\ell^r = 4$. A via-free routing solution of the instance requires at least $\lceil 4/2 \rceil = 2$ layers.

Using the strategy of "generating a CIS of the current π while keeping the LCDS length of the remaining permutation (by excluding the CIS from the current π) as short as possible," the algorithm, first, finds an LCDS M of π. Let the length of M be ℓ^r. The algorithm, then, takes k iterations. At the i-th iteration, $\forall\ i = 1, 2, \ldots, k$, the current length of S_i, say *currentlength*, is initialized to zero. Then, the following steps are executed: (I) If $\ell^r \leq 2$, then all the n nets of the current π can be routed in one layer. Let S_i be the current π and the execution of the algorithm stops. Otherwise, (II) a CIS of the current π is generated as S_i such that the LCDS length, say ℓ', of the remaining permutation (by excluding the CIS from the current π) is kept as short as possible. (III) The CIS S_i is removed from the current π. Let the remaining permutation be the current π with n updated as $n = n -$ *currentlength* and with possibly updated ℓ^r in step (II). Then, the next iteration is started.

At the i-th iteration, the details of the step (II) are as follows. This step takes $\binom{n}{2} = n(n-1)/2$ loops. In each loop, two elements $\pi(j_1)$ and $\pi(j_2)$ of the current π are selected as the starting pair. For each j_1 in the range of $1 \leq j_1 \leq n-1$, j_2 is increased from $j_1 + 1, j_1 + 2, \ldots, n$. Then, the following substeps are executed: (1) The pair of two elements $\pi(j_1)$ and $\pi(j_2)$ is used to generate a CIS (to be described in the next paragraph). Let the length of the CIS be ℓ. (2) Exclude the CIS from the current π temporally, and let the remaining permutation be π'. (3) An LCDS of the remaining permutation π' is generated. Let the LCDS length be ℓ'. (4) If $\ell' \leq 2$ and $i < k$, then the set of the nets in the remaining permutation π' is one-layer routable. Let S_i be the CIS, set *currentlength* as ℓ and go to step (III). (5) If $\ell' < \ell^r$, then the CIS is replaced with the old S_i as the new S_i and the length ℓ^r is updated as ℓ'. Set *currentlength* as ℓ. Otherwise, (6) if $\ell' = \ell^r$, but ℓ is greater than *currentlength* of the previous selected S_i, then the CIS is replaced with the old S_i as the new S_i and the *currentlength* of the new S_i is updated as ℓ. Then, another pair of elements from the current π is selected and goes to the next loop.

Let *IS* represent an increasing subsequence. The above step (1) of generating a CIS by using the pair of elements $\pi(j_1)$ and $\pi(j_2)$ consists of the following substeps: (i) The scanning order is always, first, from $\pi(j_1)$ to $\pi(j_1 + 1), \ldots, \pi(n)$, then cyclic back to $\pi(1), \ldots, \pi(j_1 - 1)$. (ii) The CIS has two forms: (A) $(\pi(j_1), IS_1, \pi(j_2), IS_2, IS_3)$ for the case of $\pi(j_1) < \pi(j_2)$, where IS_1 (if any) is laid clockwise between $\pi(j_1)$ and $\pi(j_2)$, and each element in IS_1 is greater than $\pi(j_1)$ and less than $\pi(j_2)$; IS_2 (if any) is laid clockwise between $\pi(j_2)$ and $\pi(j_1)$, and each element in IS_3 is less than $\pi(j_1)$. (B) $(\pi(j_1), IS_2, IS_3, \pi(j_2), IS_1)$ for the case of $\pi(j_1) > \pi(j_2)$, where the roles of $\pi(j_1)$ and $\pi(j_2)$ in the definitions of IS_1, IS_2, IS_3 are switched. (iii) The corresponding IS_j, $j = 1, 2, 3$, is generated in any possible way whenever a scanned element of π other than $\pi(j_1)$ and $\pi(j_2)$ meets the definition of IS_1, IS_2, or IS_3. That is, each *IS* is generated such that a subsequent element selected is always greater than the element found previously until the condition of another *IS* is satisfied. Let the steps (i) to (iii) be called the procedure of GENCIS(π, j_1, j_2). Using the permutation $\pi = (3, 5, 8, 4, 6, 1, 7, 2)$ in Figure 1(a) as an example, the procedure GENCIS (π, 1,5) will generate $IS_1 = (5)$, $IS_2 = (7)$, and $IS_3 = (2)$, if $\pi(1) = 3$, $\pi(5) = 6$. The generated CIS is, therefore, (3, 5, 6, 7, 2). If $\pi(2) = 5$, $\pi(4) = 4$, then the procedure GENCIS(π, 2, 4) will generate $IS_1 = 0$, $IS_2 = (8)$, and $IS_3 = 0$. The generated CIS is, therefore, (5, 8, 4). If $\pi(4) = 4$, $\pi(6) = 1$, then the procedure GENCIS(π, 4, 6) generates $IS_1 = (2, 3)$, $IS_2 = (6)$, and $IS_3 = 0$. The generated CIS is (4, 6, 1, 2, 3). From the discussion, the formal algorithm for finding a set of k disjoint CIS's of π is as follows.

Algorithm. Finding k disjoint CIS's $S_1, S_2, \ldots, S_k$ such that $\sum_{i=1}^{k} |S_i|$ is maximized.

Input:	A permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ and an integer $k \geq 2$
Output:	k disjoint CIS's $S_1, S_2, \ldots, S_k$ such that $\sum_{i=1}^{k} \lvert S_i \rvert$ is maximized.

Step 1. Find an LCDS M of π (by finding an LCIS of the reverse permutation π^r). Let its length be ℓ^r.

Step 2. For i: = 1 to k Do
currentlength: = 0;
Step 2.1 If $\ell^r \leq 2$, then /* The nets in the current π is one-layer routable.*/
let S_i be the current π and stop.
Endif
/* Generate a CIS of the current π while keeping the LCDS length of the remaining permutation as short as possible.*/
Step 2.2. For j_1: = 1 to $n - 1$ Do
For j_2: = $j_1 + 1$ to n Do
Step 2.2.1. Generate a CIS by using the procedure GENCIS(π, j_1, j_2). Let its length be ℓ.
Step 2.2.2. Excluding the CIS from π, let the remaining permutation be π'.
Step 2.2.3. Find an LCDS of π'. Let its length be ℓ'.
Step 2.2.4. If $\ell' \leq 2$ and $i < k$, then
let S_i be the CIS. Set *currentlength*: = ℓ and ℓ^r: = ℓ'. Go to Step 2.3.
Endif.
Step 2.2.5. If $\ell' < \ell^r$, then
let S_i be the CIS. Set *currentlength*: = ℓ and ℓ^r: = ℓ'
Else
Step 2.2.6. If $\ell' = \ell^r$ and $\ell >$ *currentlength*, then let S_i be the CIS. Set *currentlength*: = ℓ.
Endif
Endif
Endfor /* The j_2 forloop. */
Endfor /* The j_1 forloop. */
Step 2.3. Let π: = $\pi - S_i$ be the current permutation; n: = n − *current length*.
Endfor /* The i forloop.*/
End of Algorithm.

Take for an example, the instance in Figure 1(a) has n = 8, k = 2 and the original permutation π = (3, 5, 8, 4, 6, 1, 7, 2). An LCDS M = (7, 5, 4, 1) of π with a length of ℓ^r = 4 is first obtained in step 1. The algorithm will, then, run in two iterations since k = 2. During the first iteration (i = 1), step 2.1 is not executed because $\ell^r > 2$ and the problem is not one-layer routable. The step 2.2 is executed $8(8 - 1)/2 = 28$ loops, at most. The pair of elements $\pi(1)$ = 3 and $\pi(2)$ = 5 is first selected. The step 2.2.1 will generate a CIS (3, 5, 8, 1, 2) with a length of ℓ = 5 using the procedure GENCIS(π, 1, 2). The step 2.2.2 excludes this CIS from π, thus, the remaining permutation is π' = (4, 6, 7). In step 2.2.3, an LCDS (7, 4) of π' is obtained with a length of ℓ' = 2. In the step 2.2.4, the CIS (1, 2, 3, 5, 8) is selected to be S_1 since ℓ' = 2 and $i < k$. The length ℓ^r is updated as $\ell^r = \ell'$ = 2 and the execution goes to step 2.3. The step 2.3 removes this LCIS from π. The current permutation is π = (4, 6, 7) with n = 3 and updated ℓ^r = 2 for the second iteration (i = 2). In step 2.1, the current permutation π = (4, 6, 7) is obviously one-layer routable because ℓ^r = 2. The current permutation π = (4, 6, 7), which is a CIS, is, therefore, selected as S_2. Then, the execution of the algorithm stops. The required solution is represented by these two subsets S_1 = (1, 2, 3, 5, 8) and S_2 = (4, 6, 7) as shown in Figure 1(b).

4 THE CORRECTNESS AND COMPLEXITY OF THE ALGORITHM

The correctness of the algorithm is proved in the following. A CIS of the current π is generated at the end of each iteration. Since the remaining permutation π' is obtained by removing the elements of the CIS from the current π for each iteration, the LCDS length of the remaining permutation π' cannot be greater than that of the current π. Otherwise, there would exist another LCDS of the original permutation, and its length is longer than the one just selected. This is impossible. Therefore, the algorithm must be stopped when k CIS's $S_1, S_2, \ldots, S_k$ are generated. When all nets are assigned to $j (j < k)$ CIS's, then the subsets $S_{j+1}, \ldots, S_k$ become empty sets.

The time complexity of the algorithm is analyzed as follows. Step 1 takes $O(n^2)$ time, in the worst case, to generate an LCDS. Step 2 takes k iterations. From steps 2.1 to 2.3, a constant time is taken to assign *currentlength* := 0 at each iteration. Step 2.1 requires $O(n)$ time. Step 2.2 is a nested loop that has substeps 2.2.1 to 2.2.6. These substeps will be executed at most $\binom{n}{2} = n(n-1)/2 = O(n^2)$ times; the dominating step 2.2.3 computes the LCDS which takes $O(n^2)$ time in the worst case. Step 2.3, also, requires $O(n)$ time. Hence, the complexity of the algorithm is $O(n^2 + k(n + n^2 n^2 + n)) = O(k\, n^4)$.

5 EXPERIMENTAL RESULTS

This heuristic algorithm is written in C language and tested on a SUN workstation. To measure how far the number of solutions obtained from the algorithm are away from the number of optimal solutions and the algorithm's speed, an exhaustive algorithm for finding all optimal solutions is, also, written. The exhaustive algorithm first creates k *directed acyclic graphs* [10], then finds the corresponding k disjoint directed paths such that the total length of these k directed paths is maximized. The search method tries all possible directed paths; each directed path of the k directed acyclic graphs corresponds to a CIS. Using the optimal solutions obtained by the exhaustive algorithm as a base, the optimal solution's discrepancy found by the proposed algorithm can be measured and summarized as follows.

First, all permutations for $n = 8$, 10, and 12 are generated and tested with $k = 2$, 3, respectively. To measure the discrepancy between the number of optimal solutions obtained from the heuristic algorithm and from the exhaustive algorithm, the *optimal ratio* of the heuristic algorithm is defined as the percentage of the number of optimal solutions obtained by the heuristic algorithm from these $n!$ permutations/the $n!$ optimal solutions found by the exhaustive algorithm. Table I lists the results.

When $n = 20$ or 30, the execution time is too long to generate all optimal solutions for all possible permutations; hence, only 1000 permutations are randomly generated and tested with $k = 2$, 3 and 4. Therefore, the optimal ratio is defined as the number of optimal solutions obtained by the heuristic algorithm from these 1000 permutations/the 1000 optimal solutions found by the exhaustive algorithm. As indicated in Table II, the ratio is not impressive in the cases of $k = 2$ and $n = 20$ or 30, or $n = 30$ and $k = 2$ or 3. These results indicate that the proposed algorithm cannot always find the optimal solution for some permutation.

Two observations are made from Tables I and II. First, the optimal ratio decreases as n increases for fixed k. The reason is simple. The possibility that the number of nets can be grouped into one layer is lowered when the total number of nets is increased for the fixed k's. Second, the optimal ratio of the algorithm increases as k increases for the fixed n's because the more layers are available, the more nets can be selected into these layers.

Table III shows the *total execution*† time between the proposed algorithm and the exhaustive algorithm for some test cases with $k = 2$. For $n = 10$ and 12, the data are calculated for all permutations. For $n = 20$ and 30, the data are based on these 1000 permutations generated randomly. When the execution time of the proposed algorithm and that of the exhaustive algorithm are compared, the execution time of the proposed algorithm can be almost neglected when $n \geq 12$.

TABLE I
The Optimal Ratio of the Proposed Algorithm for all Permutations of $n = 8$, 10, 12 and $k = 2,3$

Optimal Ratio	n = 8	n = 10	n = 12
k = 2	100%	98.6%	96.4%
k = 3	100%	100%	99.9%

TABLE II
The Optimal Ratio of the Proposed Algorithm for 1000 Random Permutations of $n = 20$, 30 and $k = 2$, 3, 4

Optimal Ratio	n = 20	n = 30
k = 2	72.7%	36.8%
k = 3	93.9%	62.9%
k = 4	98.9%	96.2%

TABLE III
The Total Execution Time Between the Proposed Algorithm and the Exhaustive Algorithm for $k = 2$

Execution Time (seconds)	n = 10	n = 12	n = 20	n = 30
Exaustive algorithm	4760	37910	7524000	23323850
Proposed algorithm	271	1075	8100	39550

6 CONCLUSIONS

A heuristic algorithm with complexity of $O(k\ n^4)$ is proposed for solving the split k-CTVM problem. By the experimental results, this heuristic algorithm does provide an efficient solution to the problem. However, it may not always find an optimal solution. The reason is as follows. At each iteration, the algorithm only selects the first CIS so that the LCDS length of the remaining permutations may be as short as possible. There may be another CIS which gives a better solution for the remaining permutations.

The circular permutation channel has two related problems. One is determining a minimum number of layers, say r, needed to provide a via-free routing solution (as described in Section 3) for the n nets on a circular permutation channel. This problem, by similar arguments as in Section 2, is equivalent to determining the minimum number r needed to partition the permutation π into r disjoint CIS's $S_1, S_2, \ldots, S_r$ such that $\sum_{i=1}^{r} |S_i| = n$. It seems that the strategy, generating a CIS of π while keeping the remaining permutation's LCDS length (by excluding the CIS from π) as short as possible, could be used to solve this problem.

The other problem is finding a single-layer routable subset of the n nets with maximum cardinality. This problem, according to Lemma 3, is equivalent to finding a CIS S_1 of π such that $|S_1|$ is a maximum, i.e., an LCIS of π. The $O(n\ log(t + 1) + n\ t)$ LCIS algorithm in Appendix can be used to solve this problem, where t is the size of the LCIS. An $O(n\ log\ n + n\ t)$ LCIS algorithm

†According to the element's sequence in a permutation, the execution time for each permutation is widely discrepant. As an example, the optimal solution's execution time of the permutation (1, 2, 3, 4, 5) is much shorter than that of the permutation (5, 4, 3, 2, 1). Hence, to represent the relative speed between the heuristic algorithm and the exhaustive algorithm, the total execution time is more appropriate than the average time.

proposed by Lou and Sarrafzadeh [7] has recently been brought to our attention. Both algorithms improve the $O(n^2 \log n)$ results as previously found in [10].

Note that the problem of finding a single-layer routable subset with maximum weight for a circular channel with n weighted multi-terminal nets has been studied by Liao *et al.* [5]. They proposed an $O(n^2 t)$ algorithm for solving this problem under global routing information, where t is the total number of terminals. Whether this same problem could be solved without a global routing remains open.

APPENDIX

THE LCIS PROBLEM

The LCIS problem is to find an LCIS of π. According to the definition of CIS, an LCIS is a *longest increasing subsequence* (LIS) for some cyclic permutation π_i, $\forall\ 0 \le i \le n-1$. There exists a fact that the first element of π_j is the last element of π_{j+1}, $\forall\ j = 0, 1, \ldots, n-2$. Using the above fact and other properties stated below, an efficient algorithm is presented which can find such an LCIS of π.

A.1 The LCIS Algorithm

The algorithm consists of two phases. In the first phase, a set of the best partial solutions (increasing subsequences) for $\pi_0 = \pi$ is formed in which an LIS of π_0 is obtained. Then, in the second phase, the set of best partial solutions for π_{i-1} is transformed into the set of best partial solutions for π_i in which an LIS of π_i is obtained, $\forall\ i = 1, 2, \ldots, n-1$. Among the n LIS's, the one with maximal cardinality is selected to be the LCIS. The details of these two steps are stated below.

The first phase consists of the following steps: (1) Uses the same concept as in [13] to keep track of the best partial solutions for π_0. This step has n loops starting with an empty solution set $CS = 0$, it, then, adds the elements of $\pi_0 = (\pi(1), \pi(2), \ldots, \pi(n))$, one by one (from left to right), into the solution set CS. The last element of the best partial solution of length l is stored in $CS(l)$. While the element $\pi(i)$, $i = 1, 2, \ldots, n$, is being processed, the smallest last element (assume stored in $CS(j)$) is replaced by $CS(j) := \pi(i)$. This occurs if there are previous partial solutions with last elements greater than $\pi(i)$; otherwise, $\pi(i)$ is added to the current longest partial solution, and forms a new longer partial solution. The best partial solution is represented by its last element and by a link to a partial solution which has one element less than the one being processed currently. In addition to the links, the first elements of the best partial solutions are recorded.

The second phase has $(n-1)$ loops. Each element $\pi(i)$, $\forall\ i = 1, 2, \ldots, n-1$, has been executed in each loop. It is clearly visible that $\pi(i)$ is the last element of π_i and the first element of π_{i-1} because π_i is obtained by circularly shifting one element of π_{i-1}. Moreover, the element $\pi(i)$ is either (I) the first element of the longest partial solution or (II) does not in any partial solution of π_{i-1}. The purpose of each i loop is to obtain the best partial solutions for the preceding $(n-1)$ elements of π_i from the best partial solutions of π_{i-1}, and then the element $\pi(i)$ is inserted into the right position to obtained the best partial solutions of π_i. Each i loop consists of the following steps: (i) If the case (I) holds, then the length of the partial solutions of π_{i-1} with $\pi(i)$ as the first element is shortened one unit. This procedure is used to retain the best partial solutions for the preceding $(n-1)$ elements of π_i. The first element's corresponding information for the shortened partial solutions is also modified. Otherwise, for case (II), the best partial solutions for π_{i-1} are already the best partial solutions for the preceding $(n-1)$ elements of π_i. (ii) The element $\pi(i)$ is, then, inserted into the best position using the same concept as in [13]. The corresponding previous element and the first element's information are added. After inserting $\pi(i)$, a new set of the best partial solutions for π_i is formed in which the longest partial solutions is the LIS of π_i. (iii) If the LIS is longer than the one found in the previous LCIS solution, then the LIS is obtained by back-tracking the corresponding links and it is saved as the current LCIS solution. The next element $\pi(i+1)$ is picked; and the execution goes to the $(i+1)$th loop.

Using the permutation $\pi = (3, 5, 8, 4, 6, 1, 7, 2)$ as an example, Figure 2 shows a sample execution of this step. When the element $\pi(1) = 3$ is processed, this element is added into CS by setting $CS(1) = 3$, and a partial solution (3) of length 1 is formed. The first element of this partial solution is the element 3 itself. The notation $F(CS(j))$ denotes the first element of the partial solution of length j with $CS(j)$ as the last element. Thus, for this case, $F(CS(1)) = F(3) = 3$ are shown in Figure 2(a), where the number in the brace is the information for the first element. When the element $\pi(2) = 5$ is processed, it can, then, be added into CS by setting $CS(2) = 5$ and a partial solution (3, 5) of length 2 can, then, be formed. The previous element of $CS(2) = 5$ is the element 3. The notation $P(CS(j))$ denotes the link to the previous element of $CS(j)$ for the partial solution of length j with $CS(j)$ as the last element. The notation is used to trace back the partial solution of length j since the CS set may be overwritten by the subsequent elements. For this case, $P(CS(2)) = P(5) = 3$, $F(CS(2)) = F(5) = 3$ are shown in Figure 2(b), where the previous element is represented by a directed edge pointed to that element. When the element $\pi(3) = 8$ is processed, the partial solution (3, 5, 8) of length 3 is formed with $CS(3) = 8$, $P(CS(3)) = 5$,

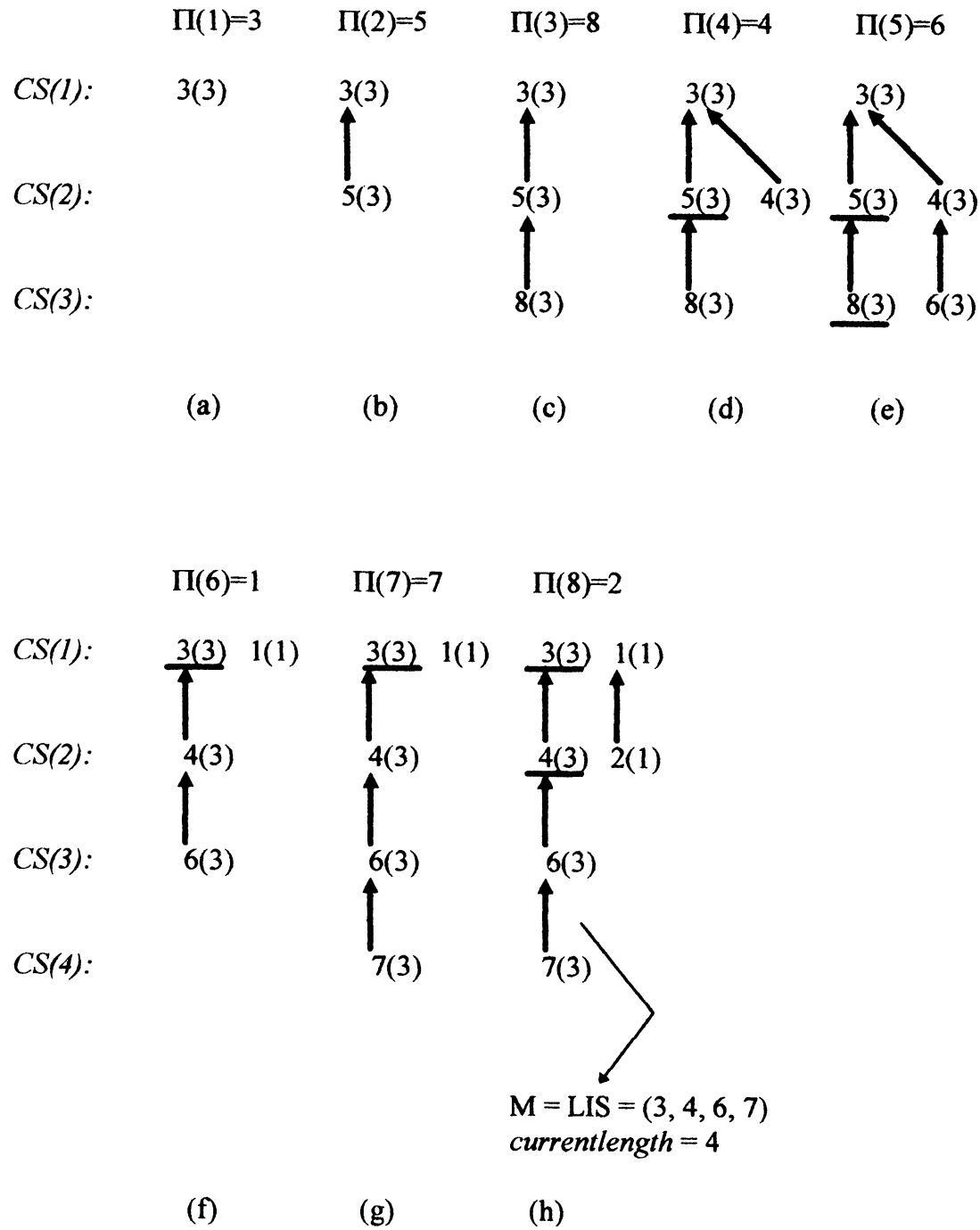


FIGURE 2 A sample execution sequence of the first phase of the LCIS algorithm in which the best partial solution for $\pi_0 = \pi = (3,5,8,4,6,1,7,2)$ is obtained.

$F(CS(3)) = 3$ as shown in Figure 2(c). When the element $\pi(4) = 4$ is processed, the partial solution (3, 4) of length 2 is found to be better than the previous partial solutions (3, 5) because the length of the partial solution (3, 4) has a better chance to be lengthened in the future. Thus, $CS(2) = 5$ is overwritten by setting $CS(2) = 4$, and a new partial solution (3, 4) of length, 2 is, therefore, formed with $P(CS(2)) = 3$ and $F(CS(2)) = 3$ as shown in Figure 2(d), where the overwritten element 5 is marked by an underline. When the element $\pi(5) = 6$ is processed, the partial solution (3, 4, 6) is better than the previous partial solution (3, 5, 8), since 6 is less than 8; therefore choosing 6 as the member will give the length of the partial solution (3, 4, 6) a better chance to be lengthened. $CS(3) = 6$, $P(CS(3)) = 4$ and $F(CS(3)) = 3$ are shown in Figure 2(e). When the element $\pi(6) = 1$ is processed, the

partial solution (1) of length 1 is found to be better than the previous partial solution (3). $CS(1) = 1$ and $F(CS(1)) = 1$ are shown in Figure 2(f). The first six elements 3, 5, 8, 4, 6, 1 have been processed with the best partial solutions being so far (1) of length 1, (3, 4) of length 2, and (3, 4, 6) of length 3. When the next element $\pi(7) = 7$ is processed, it is useless to extend the partial solutions (1) to (1, 7) and (3, 4) to (3, 4, 7) because the better partial solutions (3, 4) and (3, 4, 6) already exist. When element 7 is added into the partial solution (3, 4, 6), its length is increased from 3 to 4, and the new partial solution is (3, 4, 6, 7). $CS(4) = 7$, $P(CS(4)) = 6$ and $F(CS(4)) = 3$ are shown in Figure 2(g). When the final element $\pi(8) = 2$ is processed, only the partial solution (1) can be extended to (1, 2), which is better than (3, 4). The subsequence (1, 2), thus, becomes a new partial solution of length 2 by setting $CS(2) = 2$, $P(CS(2)) = 1$ and $F(CS(2)) = 1$ as shown in Figure 2(h). At this moment, the best partial solutions are (1) of length 1, (1, 2) of length 2, (3, 4, 6) of length 3, and (3, 4, 6, 7) of length 4. The LIS of π_0 is (3, 4, 6, 7). This LIS can be obtained by back-tracking the corresponding links from $P(CS(4))$, and then saved in the set M. This is the final step of the first phase. The set M will be updated, if possible, in the second phase of the algorithm.

A sample execution of the second phase of the LCIS algorithm is shown in Figures 3(a) to (g). When the element $\pi(1) = 3$ was processed, the best partial solution $\pi_0 = (3, 5, 8, 4, 6, 1, 7, 2)$ is transformed into the best partial solution $\pi_1 = (5, 8, 4, 6, 1, 7, 2, 3)$ as follows. The element 3 is found to be the first element of the longest partial solution of π_0 as shown in Figure 2(h). Thus, by removing 3 from the partial solutions (3, 4, 6) and (3, 4, 6,7), they are shortened to (4, 6) and (4, 6, 7), respectively. This is done by setting $F(7) = 4$ and $F(6) = 4$. The subsequence (4, 6) is not better than the previous partial solution (1, 2) of π_0. Thus, the best partial solutions for the preceding seven elements of $\pi_1 = (5, 8, 4, 6, 1, 7, 2, 3)$ are (1), (1, 2) and (4, 6, 7). The element 3 is, then, inserted into these partial solutions; (1, 2) is extended to (1, 2, 3) which is better than (4, 6, 7). Therefore, the best partial solutions for π_1 are (1), (1, 2) and (1, 2, 3) in which the LIS of π_1 is obviously (1, 2, 3). This LIS is not longer than the previously found LIS (saved in the set M), therefore, the LIS for π_1 will not be saved. The result is shown in Figure 3(a). Then, the next element $\pi(2) = 5$ is processed for finding the best partial solutions of $\pi_2 = (8, 4, 6, 1, 7, 2, 3, 5)$ and so on. Figure 3 shows these steps.

The formal LCIS algorithm is described as follows. The variable t is the size of the current solution set $CS = (CS(1), CS(2), \ldots, CS(t))$. The variable $CS(0) = 0$ is used as a dummy number and the variable T is used as a temporary variable. The initialization of $F(\pi(i)) := 0, \forall\ i = 1, 2, \ldots, n$, means that the first elements of all partial solutions are not known in the beginning of the algorithm.

Algorithm LCIS. Finding an LCIS of a permutation π.

```
Input.  A permutation π = (π(1), π(2), ..., π(n))
Output. An LCIS M of π
Step 0.   t := 0; CS(0) := 0; T := 0;
          For i := 1 to n Do F(π(i)) := 0;
/*Generating an LIS of π_0 = π, and save it in the set M.*/
Step 1.   i := 1 to n Do
          If CS(t) < π(i), then t := t + 1; j := t
          Else j := min{ℓ | 1 ≤ ℓ ≤ t, CS(ℓ) > π(i)};
          CS(j) := π(i); P(π(i)) := CS(j − 1);
          If j = 1, then F(CS(j)) := π(i)
          Else F(CS(j)) := F(CS(j−1));
          Endfor
Step 2.   M(t) := CS(t);
          For i := t − 1 down to 1 Do M(i) := P(M(i
          + 1));
          currentlength := t;
/*Generating an LIS of π_i, ∀i = 1, 2, ..., n − 1, and the
longer one replaces the old one by saving it in the set M.
*/
Step 3.   For i := 1 to n − 1 Do
  Step 3.1  If π(i) = F(CS(t)), then
            If i = n − 1, then STOP.
            For i' := 1 to t − 1 Do
            If F(CS(i')) = π(i), then
            CS(i') := CS(i' + 1);
            If i' = 1, then F(CS(i')) := CS(i')
            Else
            If P(CS(i')) = CS(i' − 1), then
            F(CS(i')) := F(CS(i' − 1))
            Else
            T := P(CS(i'));
            If i' > 2, then Do (i' − 2) times of T :=
            P(T);
            F(CS(i')) := T;
            Endif
          Endif
        Endif
      Endfor
      t := t − 1;
      Endif /* end of step 3.1 */
Step 3.2.   If CS(t) < π(i), then t := t + 1; j := t
            Else j := min{ℓ | 1 ≤ ℓ ≤ t, CS(ℓ) >
            π(i)};
            CS(j) := π(i); P(π(i)) := CS(j − 1);
            If j = 1, then F(CS(j)) := π(i)
            Else F(CS(j)) : = F(CS(j − 1)); /* end
            of step 3.2*/
Step 3.3  If t > currentlength, then M(t) := CS(t);
          For j := t − 1 down to 1, Do M(j) :=
```

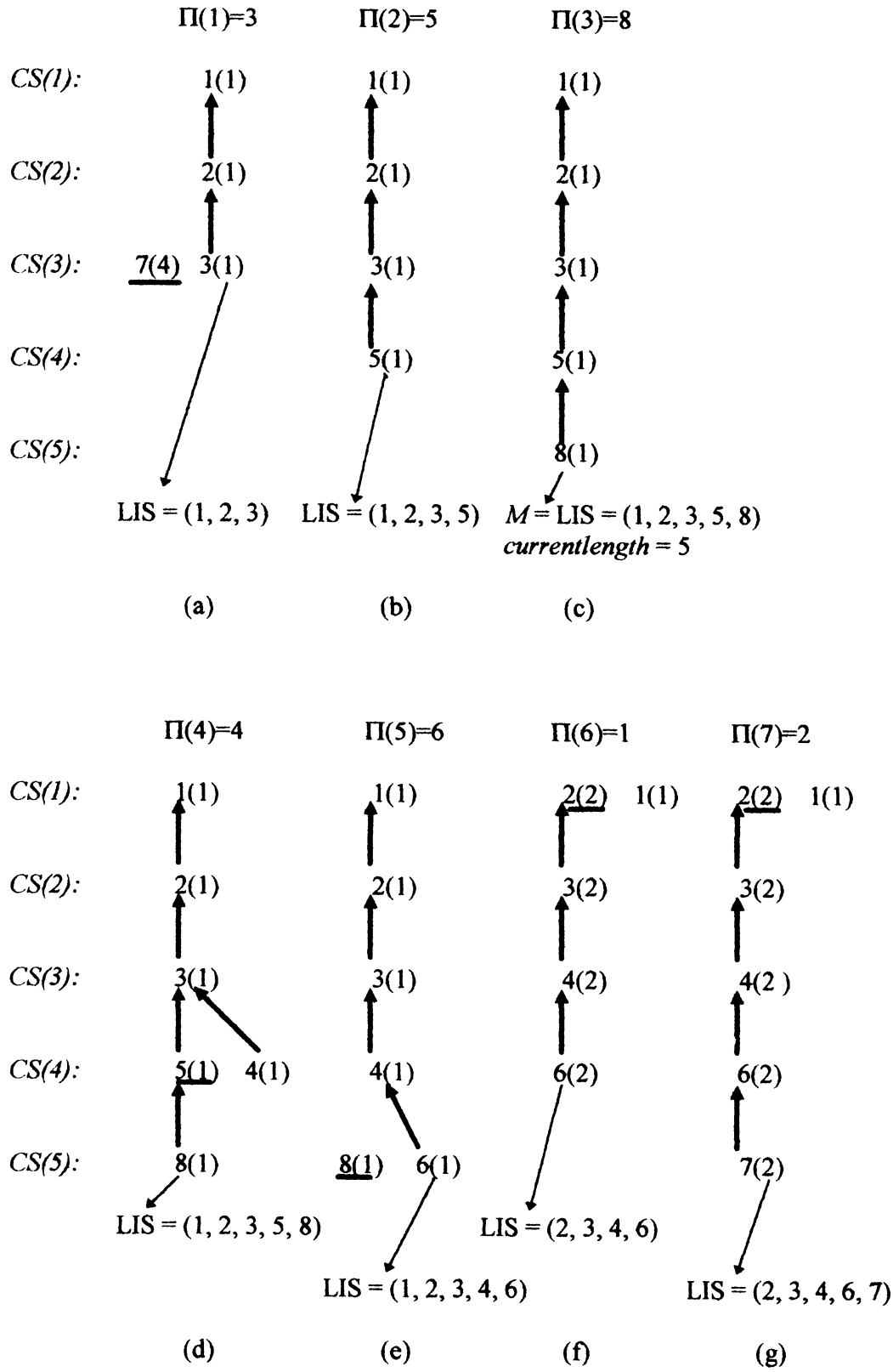


FIGURE 3 A sample execution sequence of the second phase of the LCIS algorithm in which the best partial solution for π_i is obtained as the element $\pi(i)$ is being processed, $\forall\ i = 1,2,\ldots, n-1$.

$P(M(j + 1))$;
currentlength:= t;
Endif /* end of step 3.3. */
Endfor /* end of step 3.*/
End of Algorithm.

A.2 The Correctness and Complexity of The LCIS Algorithm

The correctness of the LCIS algorithm is shown in the following lemma.

Lemma A1. The set M obtained by the LCIS algorithm is an LCIS of π.

Proof: The correctness of finding an LIS for π_0 in the first phase of the LCIS algorithm has been proved by Fredman [1]. The LIS is temporally saved in the set M. In the second phase of the LCIS algorithm, the best partial solution for π_{i-1} is transformed into the best partial solution for π_i, $\forall\ i = 1, 2, \ldots, n - 1$. The correctness of finding an LIS for π_i can be proved by using the same technique in [1]. Since the longer LIS is saved in the set M, therefore, the LCIS of π can always be found from an LIS of π_j for some j.

The time complexity of the LCIS algorithm is analyzed as follows. Step 0 takes $O(n)$ time. Step 1 takes $O(n + t + (n - t)\ log\ (t + 1))$ time by similar arguments as in [13]. Step 2 takes $O(t)$ time. Step 3 has $(n - 1)$ iterations. In each iteration, step 3.1 takes $O(t)$ time as in the following explanation. In the worst case of updating $F(CS(.))$ values, one element in CS takes no more than $O(t)$ time and each of the other elements in CS takes no more than $O(1)$ time. The other substeps take no more than $O(t)$ time. Step 3.2 takes, at most, $O(log(t + 1))$ time. Step 3.3 takes $O(t)$ time. The algorithm, thus, has a time complexity of $O(n\ log(t + 1) + nt)$ from which the following theorem is obtained.

Theorem 1. The LCIS problem can be solved in $O(n\ log(t + 1) + nt)$ time.

Proof: The set M obtained by the LCIS algorithm is indeed an LCIS of π by Lemma A1. The theorem follows by the timing analysis of the LCIS algorithm.

References

[1] M. L. Fredman, "On computing the length of the longest increasing subsequence," *Discrete Math.*, vol. 11, no. 1, pp. 29–35, 1975.

[2] J. Cong and C. L. Liu, "On the k-layer Planar Subset and Via Minimization Problems," *Proc. of The European Design Automation Conf.*, 1990, pp. 459–463.

[3] M. Garey and D. Johnson, *Computer and Intractability: A Guide to the Theory of NP-completeness*, San Francisco, CA: Freeman, 1979.

[4] C. P. Hsu, "Minimum via topological routing," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 235–246, Oct. 1983.

[5] K. F. Liao, D. T. Lee and M. Sarrafzadeh, "Planar subset of multi-terminal nets," *INTEGRATION, the VLSI journal*, vol. 10, pp. 19–37, 1990.

[6] R. D. Lou, M. Sarrafzadeh and D. T. Lee, "An Optimal Algorithm for the Maximum Two-Chain Problem", *Proc. of First SIAM-ACM Conf. on Discrete Algorithms*, San Francisco, CA, Jan. 1990.

[7] R. D. Lou and M. Sarrafzadeh, "Circular Permutation Graph Family with Applications," to appear in *Discrete Applied Math.*

[8] M. Marek-Sadowska, "An unconstrained topological via minimization problem for two-layer routing", *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 184–190, July 1984.

[9] C. S. Rim, *A study on some VLSI layout problems*, Ph. D. dissertation, Elec. Eng. Dept., Univ. of Maryland, College Park, MD, Mar. 1989.

[10] C. S. Rim, T. Kashiwabara, and K. Nakajima, "Exact Algorithms for Multilayer Topological Via Minimization," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 11, pp. 1165–1173, Nov. 1989.

[11] M. Sarrafzadeh and D. T. Lee, "A new approach to topological via minimization," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 8, pp. 890–900, Aug. 1989.

[12] M. Sarrafzadeh and R. D. Lou, "Maximum k-coverings of Weighted Transitive Graphs with Applications," *Proc. of Internat. Symp. on Circuits and Systems*, 1990, pp. 332–335.

[13] P. Widmayer and C. K. Wong, "An optimal algorithm for the maximum alignment of terminals," *Information Processing Letters*, vol. 20, pp. 75–82, 15 Feb. 1985.

[14] X. M. Xiong, "A new algorithm for topological routing and via minimization," *Proc. 1988 IEEE Int. Conf. on Computer-Aided Design*, 1988, pp. 410–413.

[15] X. M. Xiong and E. S. Kuh, "A unified approach to the via minimization problem," *IEEE Trans. Circuit Syst.*, vol. 36, no. 2, pp. 190–204, Feb. 1989.

Biographies

JAI-SHEN HUANG received the B. S. and M. S. degrees in applied mathematics from the National Tsing-Hua University in 1977 and 1980, respectively. Currently, he is a researcher at the Advanced Technology Center of Computer and Communication Research Laboratories, a division of Industrial Technology Research Insitute. His research interests include design and analysis of algorithm, CAD for VLSI layouts, digital signal processing.

Y. H. CHIN received the B. S. E. E. degree form the National Taiwan University, and the M. S. and Ph.D. degrees from the University of Texas at Austin in 1970 and 1972, respectively. He was a member of the faculty at Northwestern University, Cleveland State University, and National Chiao-Tung University. His industrial experiences included Control Data Corporation, Sunnyvale, CA., Telecommunication Labs, Chung-li, Taiwan, and AT&T Bell Laboratories, Holmdel, NJ. Currently, He has been a professor at the Institute of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan 30043 since 1981. His research interests include database management systems, operating systems, design and analysis of algorithms, and VLSI design.