Optimal Bus Sequencing for Escape Routing in Dense PCBs

Hui Kong[1] Tan Yan[1] Martin D.F. Wong[1] Muhammet Mustafa Ozdal[2]

[1]Department of ECE, University of Illinois at U-C, Urbana, IL 61801

[2]Intel Corporation, Hillsboro, OR 97124

Email: {huikong2,tanyan2,mdfwong}@uiuc.edu mustafa.ozdal@intel.com

***Abstract*— The PCB routing problem has become so difficult that no commercial CAD software can provide an automatic solution for high-end boards. Existing algorithms for escape routing, an important step in PCB routing, are net-centric. Directly applying these algorithms will result in mixing nets of different buses together. But in practice, it is preferred to bundle together nets in a bus. Thus the bus-centric escape routing problem can be naturally divided into two subproblems: (1) finding a subset of buses that can be routed on the same layer without net mixings and crossings, which we refer to as the *bus sequencing problem*, and (2) finding the escape routing solutions for each chosen bus, which can be solved by a net-centric escape router. In this paper, we solve the bus sequencing problem. We introduce a new optimization problem called the Longest Common Interval Sequence (LCIS) problem and model the bus sequencing problem as an LCIS problem. By using dynamic programming and balanced search tree data structure, we present an LCIS algorithm which can find an optimal solution in $O(n \log n)$ time. We also show that $O(n \log n)$ is a lower-bound for this problem and thus the time complexity of our algorithm is also the best possible.**

I. Introduction

During the past few years, we have seen dramatic advances in the IC technology. The shrinkage of die sizes and the increase in functional complexities have made circuit designs more and more dense. So, boards and packages have reduced in size while the pin counts have been increasing. Today, high-density fine-pitch packages typically contain pin counts in the order of thousands, while they occupy only minimal board space [7]. Due to these factors, traditional routing algorithms cannot handle the new challenges effectively. Today many high-end designs in the industry are routed manually, in a time-consuming manner.

The printed circuit board (PCB) routing problem can be decomposed into two separate problems: (1) routing nets from pin terminals to component (MCM, memory, etc.) boundaries, which is called *escape routing* (see the solid line in Figure 1), and (2) routing nets between component boundaries, which is called *area routing* (see the dashed line in Figure 1). In this paper, we only discuss the escape routing problem for a single layer. Previous escape routing algorithms [2], [3] are net-centric. However, in practice, as shown in industrial manual routing solutions, nets are usually organized in bus structures, and nets in a bus are expected to be routed together without foreign wires in between. Obviously, directly applying the net-centric algorithms in [2], [3] to all the buses will result in mixing nets of different buses together as shown in Figure 1. Thus, the escape routing problem is bus-centric and can be defined as finding the maximum number of nets, such that (1) the routed nets in a bus are bundled together without foreign wire (i.e., the buses are not mixed up), and (2) the nets escaped from the components do not lead to any crossing between component boundaries. Therefore, the bus-centric escape routing problem can naturally be divided into two subproblems: (1) finding a subset of buses that can be routed on the same layer without net mixings and crossings, which we refer to as the *bus sequencing problem*, and (2) finding the escape routing solution for each chosen bus, which can be solved by existing net-centric escape routing algorithms.

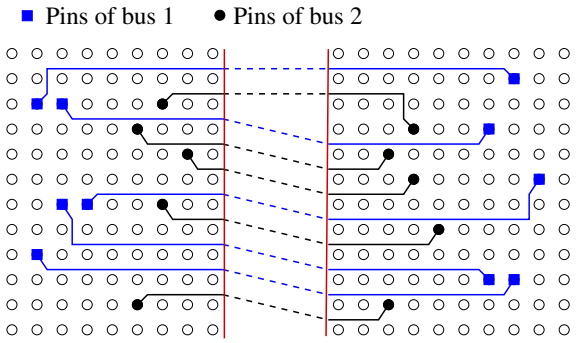


Fig. 1. A sample net-centric escape routing solution for a problem with two buses. Nets of these two buses are mixed up.

In this paper, we solve the bus sequencing problem. We introduce a new optimization problem called the Longest Common Interval Sequence (LCIS) problem and formulate the bus sequencing problem as an LCIS problem. The LCIS problem is a generalized version of the well-known longest common subsequence (LCS) problem. But traditional LCS algorithms cannot be directly applied to solve it. We propose an LCIS algorithm based on dynamic programming and balanced binary search tree (BST) data structure. Our algorithm guarantees to find an optimal solution in $O(n \log n)$ time. We also show that $O(n \log n)$ is a lower bound and thus the time complexity of our algorithm is also the best possible.

1-4244-1382-6/07/$25.00 ©2007 IEEE

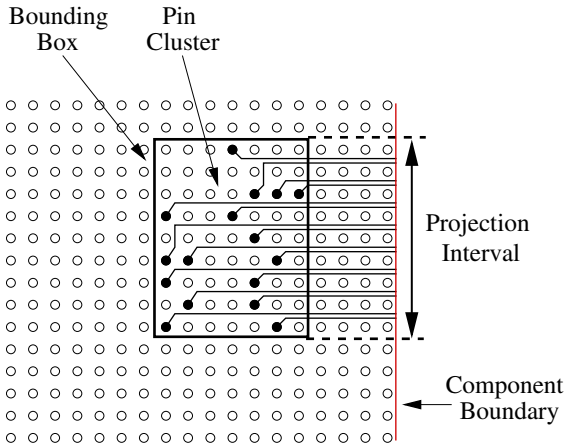


Fig. 2. Illustration of the bus projection interval.

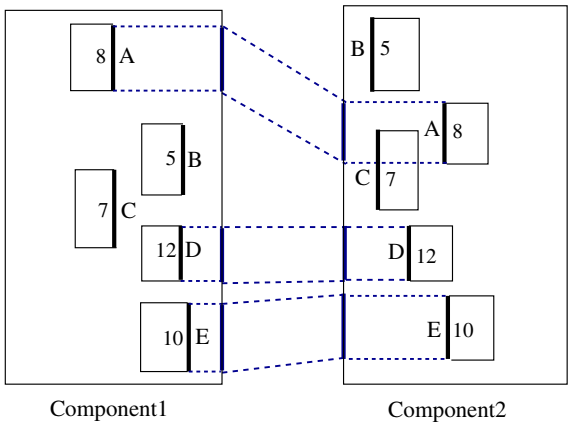


Fig. 3. A sample bus sequencing problem with five buses. Each bus is represented by the bounding boxes of their pin clusters. The thick edge of each bounding box should be projected to the boundary to get the projection interval. The number inside the box represents the bus weight. The dashed lines show the solution.

The rest of this paper is organized as follows: in Section 2, we formulate the bus sequencing problem as the LCIS problem. Then we present our algorithm and its time complexity analysis in Section 3. Section 4 demonstrates some experimental results to compare the escape routing results with and without using the LCIS algorithm. Finally, concluding remarks are given in Section 5.

II. Problem Formulation

A typical circuit board contains several chip components such as MCM, memory, and I/O module. These components are mounted on or plugged into the board, making a number of dense pin arrays on the board. Let a component be defined as a 2-D array of pins. The input to the escape routing problem is assumed to contain two components separated by a channel. A bus is a group of 2-pin nets, and it has a pin cluster in each component. The escape route for a given net is defined as the route from its terminal pins (within components) to the respective component boundaries. Then the bus sequencing problem for the escape routing is to find a subset of buses, such that: (1) the sum of the number of nets in the chosen buses is maximum, (2) the net escape routes belonging to different buses will not be mixed up, and (3) the net escape routes of different buses will not have crossings in the intermediate channel. For simplicity of presentation, we focus on a horizontal problem, where one component is to the right of the other. It is straightforward to extend the algorithm to a vertical problem.

For each bus, its projection interval on a component can be obtained by projecting the bounding box of its pin cluster onto the component boundary, as shown in Figure 2. From industrial manual routing solutions, we observe that the nets of a bus typically escape a component from its projection interval. Based on this observation, each bus can be represented by two intervals located in different components.

Naturally, if two bus intervals overlap in a component, the net escape routes of the corresponding buses are mixed up. Therefore, for the buses chosen to be routed together, their intervals in each component should have no overlapping. If we define the ordering of a group of intervals in a component without overlapping by counting them from top to bottom, then the intervals of the chosen buses correspond to two sequences of intervals, one in each component. For example, in Figure 3, intervals of buses B, C, and D overlap in component 1, so they do not form a sequence, but intervals of buses B, D, and E do form a sequence in component 1. Furthermore, these two sequences of intervals should have the same ordering; otherwise their nets have crossings in the intermediate channel. In Figure 3, the intervals of bus A, B, and D in component 1 correspond to interval sequence $\langle A, B, D\rangle$, but those in component 2 correspond to a different interval sequence $\langle B, A, D\rangle$. Therefore, a common interval sequence is a bus interval sequence existing on both components, such as $\langle A, D, E\rangle$ in Figure 3. In addition, we define the weight of a bus as the number of its nets. Then the bus sequencing problem looks for an optimal common interval sequence, which is a common interval sequence with maximum sum of weights (i.e. $\langle A, D, E\rangle$ in Figure 3). Thus the bus sequencing problem is equivalent to LCIS problem defined as follows:

***Definition** 1: Given a bus set $B = \{b_1, b_2, \ldots, b_n\}$, its corresponding intervals on the left side are $L = \{l_1, l_2, \ldots, l_n\}$ and the intervals on the right side are $R = \{r_1, r_2, \ldots, r_n\}$. Each bus b_i also has a weight w_i. The LCIS problem is to find a common interval sequence of L and R such that the total weight of the corresponding buses is maximized. This total weight is denoted as $LCIS(L, R)$.*

A related problem in the literature is the LCS problem [1], [4]–[6]. In the LCS problem, each sequence defines a linear ordering on its elements. But in our LCIS problem, due to the interval overlappings, the intervals in a component only have a partial ordering. Thus, the LCIS problem is a generalization of the LCS problem since if the intervals do not overlap, it is

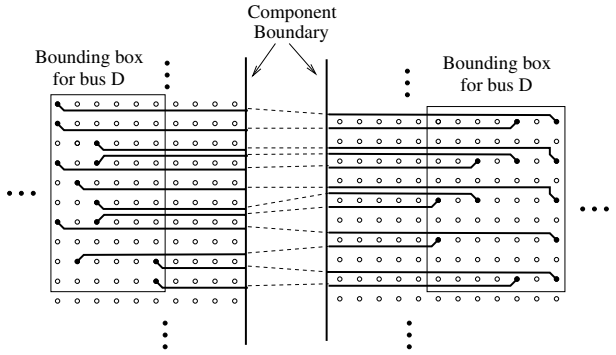


Fig. 4. The net-centric escape routing solution for bus D in the example showed in Figure 3. The solid lines show the escape routes inside the components and the dashed lines show the connections between component boundaries.

reduced to the LCS problem. No LCS algorithm can be applied to solve it directly. In the next section, we propose an LCIS algorithm which can find an optimal solution in $O(n \log n)$ time.

In practice, the bus sequencing algorithm can be applied to the bus-centric escape routing problem followed by the net-centric escape routing algorithm in [2], [3]. Here, the net-centric algorithm is applied to the buses chosen by the bus sequencing algorithm one by one, but not to all the buses directly. For the example in Figure 3, $\langle A, D, E\rangle$ is the longest common interval sequence, so buses A, D, and E are chosen to be routed on the same layer. Figure 4 gives a net-centric escape routing solution for bus D in the example in Figure 3. It is easy to see that all the nets in bus D escape the components from its projection intervals, so the net escape routes of bus D cannot mix or cross with those belonging to other buses. Similarly, apply the net-centric escape routing algorithm to bus A and E; we can then get the escape routing solution for this bus-centric escape routing problem, where 30 nets are routed.

III. Optimal LCIS Algorithm

Before we present our algorithm, we first introduce some terminologies and some definitions. For bus set B with left intervals L and R. Assume all intervals are parallel with the y axis, where the y coordinates increase from top to bottom. Then each interval $l_i = [l_i^l, l_i^u]$ in L is specified by its lower endpoint l_i^l and upper endpoint l_i^u, with $l_i^l > l_i^u$. Similarly, in R, r_i^l and r_i^u are the lower and upper endpoints of interval r_i, with $r_i^l > r_i^u$.

***Definition** 2: For a bus b_i, we define the set of buses above its lower endpoints on both sides as its* above set *A_i:*

$$A_i = \{b_j | l_j^l < l_i^l \text{ and } r_j^l < r_i^l\}$$

We also define the set of buses above its upper endpoints on both sides as its strictly above set *SA_i:*

$$SA_i = \{b_j | l_j^l < l_i^u \text{ and } r_j^l < r_i^u\}$$

See Figure 5 for an example. For bus b_2, the buses that are above the solid line in both sides compose A_2 ($A_2 = \{b_1, b_3, b_6\}$), and the buses that are above the dashed line in both sides compose SA_2 ($SA_2 = \{b_1, b_6\}$).

***Definition** 3: For a bus b_i, we also define the longest common interval sequence length $LCIS(b_i)$ as the longest length of the common interval sequence that are (inclusively) above the lower endpoint of b_i.*

In other words, if we draw a line between the lower endpoints of the two intervals of b_i, $LCIS(b_i)$ gives the length of the longest common interval sequence problem above that line. See Figure 5 for an example. $LCIS(b_2)$ defines a problem of the region above the solid line.

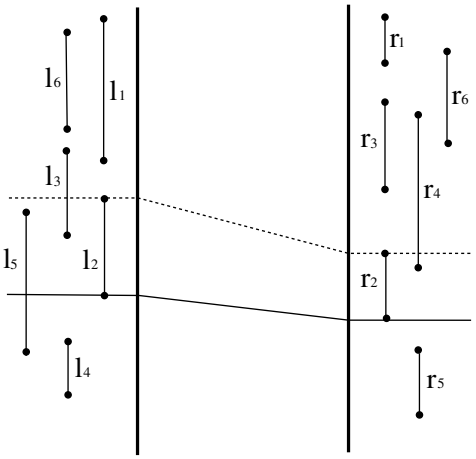


Fig. 5. Illustration for the definitions and Lemma 1.

A. The Algorithm

For bus b_i, there are two cases: either bus b_i is included in $LCIS(b_i)$ or it is not included. If it is included, then $LCIS(b_i)$ is the maximum of all $LCIS(b_j)$, in which b_j is in the strict above set of b_i, plus the weight of b_i. Otherwise, $LCIS(b_i)$ is the maximum of all $LCIS(b_k)$, in which b_k is in the above set of b_i. We can formally describe this fact in Lemma 1:

***Lemma** 1: $LCIS(b_i) = \max(\max_{b_j \in SA_i} LCIS(b_j) + w_i, \max_{b_k \in A_i} LCIS(b_k))$. If such $SA_i = \emptyset$ or $A_k = \emptyset$, then $LCIS(b_j)$ or $LCIS(b_k)$ is zero.*

We can see that this lemma gives a great hint: one LCIS problem could be solved by combining the solutions of two LCIS subproblems. Figure 5 gives an illustration. The LCIS length of bus b_2 is either the maximum of LCIS of all the buses above the dashed line (the maximum of $LCIS(b_1)$ and $LCIS(b_6)$) plus the weight of b_2, or the maximum of LCIS length of all the buses above the solid line (the maximum of $LCIS(b_1)$,$LCIS(b_6)$, and $LCIS(b_3)$). It should be the larger one of the two.

It is trivial that:

***Lemma** 2: The solution of the LCIS problem is the maximum of the LCIS length of each bus:*

$$LCIS(L,R) = \max_{b_i \in B}(LCIS(b_i))$$

The two lemmas naturally lead to a dynamic programming approach. Our basic idea is as follows: we scan through the end points on the left side from top to bottom. When we meet an upper end of a bus b_i, we find $\max_{b_j \in SA_i} LCIS(b_j)$ as in Lemma 1 and associate it with this upper endpoint. When we meet a lower endpoint of an interval b_i, we find $\max_{b_k \in A_i} LCIS(b_k)$ and compare it with the $LCIS$ value associated with the upper endpoint of this interval plus the weight of b_i. The larger value becomes the LCIS value of this bus. Finally, we find the largest LCIS value among all the buses. The algorithm is shown in Algorithm 1. In the

Algorithm 1 LCIS length computation

```
1: LCISset = ∅
2: UPPER[i] = 0 for all i
3: sort the upper and lower ends on the left side
4: for i = 1 to 2n do
5:    value = 0
6:    e = the i^th end on the left side
7:    if e is an upper end l_p^u of bus b_p then
8:       By binary search, find (endpoint, value) in LCISset
         satisfying
         (endpoint, value) = max_u{(u, v)|(u, v) ∈ LCISset
         and u < r_p^u}
9:       UPPER[p] = value
10:   else
11:      //e is a lower end l_q^l of bus b_q
12:      By binary search, find (endpoint, value) in LCISset
         satisfying
         (endpoint, value) = max_u{(u, v)|(u, v) ∈ LCISset
         and u < r_q^l}
13:      lower_q = max(UPPER[q] + w_q, value)
14:      add (r_q^l, lower_q) to LCISset
15:      remove all (u, v) from LCISset satisfying
         u > r_q^l and v < lower_q
16:   end if
17: end for
18: find (endpoint, value) in LCISset with the largest value
19: return value
```

algorithm, $LCISset$ is a sorted array that collects the lower endpoint of each interval on the right side and the LCIS length of its corresponding bus. The array is then sorted by the right lower endpoints of the intervals. Each of its elements is in the form of $(r_i^l, LCIS(b_i))$. The $UPPER$ array records all the LCIS values associated with the upper endpoints on the right.

We show the correctness of this algorithm by comparing it with Lemma 1. Lines 8 and 9 calculate $\max_{b_j \in SA_i} LCIS(b_j)$, line 12 calculates $\max_{b_k \in A_i} LCIS(b_k)$, then line 13 compares $\max_{b_j \in SA_i} LCIS(b_j) + w_i$ with $\max_{b_k \in A_i} LCIS(b_k)$ and $LCIS(b_i)$ is equal to the larger one of them. Since the endpoints on the left side are sorted and since we scan from top to bottom, we process a lower or upper endpoint on the left side only when all the lower endpoints above it are processed. On the right side, whenever we query for the LCIS, we enforce that the endpoint we query is above the upper endpoint or lower endpoint of the bus ($u < r_p^u$ in line 8 and $u < r_q^l$ in line 12). These two properties guarantee $b_j \in SA_i$ and $b_k \in A_i$ in the lemma. (Notice that if nothing can be found in line 8 and line 12, $value = 0$.) To find the maximal LCIS for all b_j and b_k as in the lemma, we use the $LCISset$ to keep track of the LCIS value for all the buses that are processed. $LCISset$ records the lower endpoints in the right side and the $LCIS$ values of their corresponding buses ($LCISset = \{(r_i^l, LCIS(b_i))\}$). Whenever we obtained the $LCIS(b_i)$ for bus b_i, we store its lower endpoint on the right side together with $LCIS(b_i)$ into $LCISset$ (line 14). After that, we should remove all the entries in $LCISset$ that has the lower endpoint below the newly added one but its LCIS value is smaller than the newly added one. Such an entry cannot lead to an optimal solution. This removing step guarantees that the $LCIS(b_i)$ value in $LCISset$ is increasing as the lower endpoint coordinate r_i^l increases. Then, whenever we need to find the maximal LCIS above an endpoint a on the right side, we just use a as the key to query in $LCISset$ and find the lowest lower-endpoint that is above a. The $LCIS$ value associated with a must be the largest among all the $LCIS$ values above a. This also explains why, in lines 8 and 12, we find the lowest endpoint (u) instead of largest LCIS value. The two are consistent: the lowest endpoint in $LCISset$ above the endpoint a means the largest LCIS length above it. In a later example, we will show how this guarantees that the maximal LCIS for all possible j and k are selected. Finally, we take the largest LCIS length from the LCISset, which reflects Lemma 2. From the discussion above, we can see that:

***Theorem** 1: Algorithm 1 gives the correct LCIS length.*

By this algorithm, we can obtain only the LCIS length. However, it is easy to obtain the LCIS sequence by adding a postprocess to back track $LCISset$.

B. An Example

Now we give a simple example to illustrate how our algorithm works. Figure 6 gives the problem configuration: the weights of each bus and the locations of each interval.

Now we simulate the process of our algorithm. Please refer to Figure 7 as we go through the whole process. We scan from top to bottom on the left side. The first two endpoints we encounter are l_1^u (the upper endpoint of l_1) and then l_3^u, since nothing has been added to $LCISset$ till now, we cannot find anything by line 8. Therefore, elements in $UPPER$ array are still 0 (Figure 7 (a) and (b)). We keep going and meet l_1^l. At this time, $UPPER[1] = 0$ and $LCISset = \emptyset$. Therefore, $lower_q = w_1 = 3$, we add $(8, 3)$ into $LCISset$, here $8 = r_1^l$ (Figure 7(c)). We then meet l_2^u. We try to find the largest endpoint smaller than $r_2^u = 1$ in $LCISset$, but cannot find

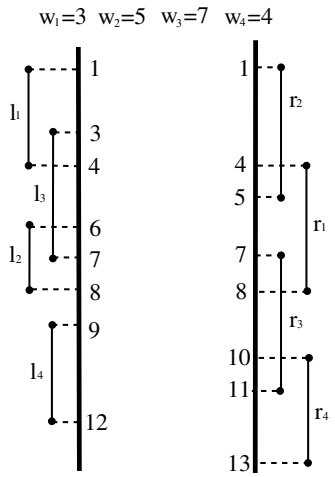


Fig. 6. Example problem configuration.

anything. Therefore, $UPPER[2] = 0$ (Figure 7(d)). For l_3^l, we use $r_3^l = 11$ to query in $LCISset$ and get $8 < 11$. We then compare 3 with $w_3+UPPER[3] = 7$ and choose $lower_q = 7$. After this, $(11, 7)$ is added into $LCISset$. Then we process l_2^l; r_2^l is 5 and the $lower_q$ we obtain is also 5. We insert $(5, 5)$ into $LCISset$. Notice that we should now remove $(8, 3)$ because $8 > 5$ and $3 < 5$. Otherwise, if we keep $(8, 3)$ in $LCISset$, we will have problem when processing l_4^u. In that case, we will use $r_4^u = 10$ to query in $LCISset$, and the largest endpoint that is smaller than 10 is 8. We would think that the LCIS length above r_4^u is 3 (bus 1 is selected). However, the best choice should be bus 2, and the LCIS should be 5. Therefore, we can see that if we do not remove $(8, 3)$ from $LCISset$, we cannot guarantee that the best choice is made every time we query $LCISset$, and we may lose the optimal solution. The rest of the simulation is similar to the previous steps. Finally, the longest LCIS length in this case is 9 (by choosing bus 2 and 4) which is optimal.

$UPPER = [\mathbf{0}, 0, 0, 0]$
$LCISset = \emptyset$
(a) Processing l_1^u

$UPPER = [0, \mathbf{0}, 0, 0]$
$LCISset = \emptyset$
(b) Processing l_3^u

$UPPER = [0, 0, 0, 0]$
$LCISset = \{$**(8,3)**$\}$
(c) Processing l_1^l

$UPPER = [0, \mathbf{0}, 0, 0]$
$LCISset = \{$(8,3)$\}$
(d) Processing l_2^u

$UPPER = [0, 0, 0, 0]$
$LCISset = \{$(8,3), **(11,7)**$\}$
(e) Processing l_3^l

$UPPER = [0, 0, 0, 0]$
$LCISset = \{$**(5,5)**,(8, 3), (11,7)$\}$
(f) Processing l_2^l

$UPPER = [0, 0, 0, \mathbf{5}]$
$LCISset = \{$(5,5), (11,7)$\}$
(g) Processing l_4^u

$UPPER = [0, 0, 0, 0]$
$LCISset = \{$(5,5), (11,7), (13,9)$\}$
(h) Processing l_4^l

Fig. 7. Execution of our algorithm. Bold means newly modified in $UPPER$ or added into $LCISset$. Underlined item means newly removed from $LCISset$.

C. Time Complexity

Assuming we have n buses, sorting the endpoints on the left takes $O(n \log n)$ time. With the $LCISset$ implemented by a height-balanced BST using the endpoint as key and its corresponding LCIS as value, finding the endpoint in $LCISset$ as in lines 8 and 12 takes $O(n \log n)$ time. Adding (line 14) one entry into the BST can also be done in $O(\log n)$. So the total time for one iteration in the **for** loop takes $O(\log n)$, except for line 15 in which there might be multiple entries to be removed in one iteration. We will discuss the time complexity of line 15 by amortized analysis. Removing one entry takes $O(\log n)$. Since we will add at most n lower endpoints into the BST, we have at most n entries to remove. Therefore, line 15 takes $O(n \log n)$ time for all iterations. Line 18 takes $O(n)$. Summing them up, we get $O(n \log n)$ time complexity for this algorithm.

***Theorem** 2: $O(n \log n)$ is the lower bound time complexity for LCIS problem.*

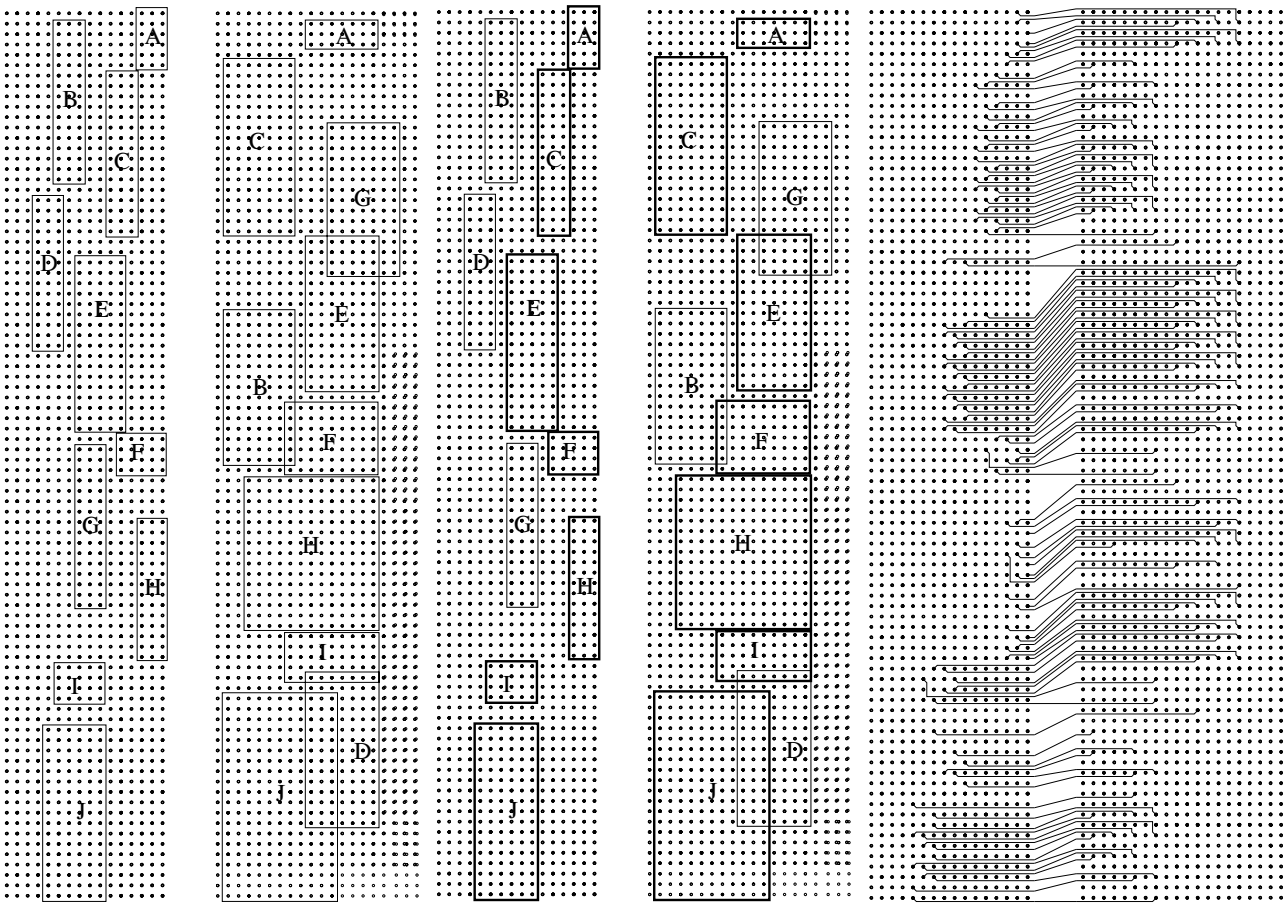
The proof of this theorem is similar to the proof of the $O(n \log n)$ lower bound for comparison-based sorting. Here we give a proof sketch. Since all the decisions can be made only by comparisons (of the interval endpoints and of the weights), we can model any algorithm as a decision tree with the leaves as solution and internal nodes as comparison operation. Since we have more than $n!$ possible solutions for this problem, the tree height must be over $O(n \log n)$. This indicates that we must do at least $O(n \log n)$ comparisons before we can reach a solution. The details of the proof are omitted.

IV. Experimental Results

In this section, we derive some two-component experiments from industrial data to compare the escape routing results between the escape router with and without using the LCIS algorithm. Table I gives the experimental results. From Table I, it is easy to see that the number of nets routed by the router with LCIS algorithm is much more than the number of nets routed by the original escape router [2] after eliminating all mixed-up nets. Through this comparison, we can conclude that the LCIS algorithm significantly improves the bus-centric escape router. For all our experiments, the LCIS algorithm is very efficient and it can be done in 1 second or less. We also illustrate an one-layer bus-centric escape routing problem and

Data	# of routed nets	
	Escape without $LCIS$	Escape with $LCIS$
Ex1	42	54
Ex2	47	56
Ex3	45	65
Ex4	62	80
Ex5	87	111
Ex6	91	128

TABLE I
COMPARISON OF THE ESCAPE ROUTING RESULTS BETWEEN THE ESCAPE ROUTER WITH AND WITHOUT USING THE $LCIS$ ALGORITHM.



(a) Original problem (b) Bus sequencing solution (c) Final escape routing solution

Fig. 8. A bus-centric escape routing problem with 10 buses from A to J and its solution. Here, $w_A = 8$, $w_B = 24$, $w_C = 26$, $w_D = 27$, $w_E = 25$, $w_F = 6$, $w_G = 21$, $w_H = 19$, $w_I = 7$, and $w_J = 20$. (a) The original bus-centric escape routing problem. The bounding boxes of all buses are shown. (b) The solution of the bus sequencing problem. Buses A, C, E, F, H, I and J, whose bounding boxes are highlighted, are chosen to be routed on the same layer. (c) The final escape routing solution of this problem by using the net-centric escape routing algorithm for buses A, C, E, F, H, I and J, respectively. In total 111 nets are routed.

its solution in Figure 8. In this example, 7 buses are chosen by the LCIS algorithm and 111 nets are routed finally.

V. Conclusions

In this paper, we gave the bus sequencing problem, which is a subproblem of the bus-centric escape routing problem, and then solved it. We introduced a new optimization problem called the Longest Common Interval Sequence (LCIS) problem and formulated the bus sequencing problem as an LCIS problem. We proposed an LCIS algorithm that can find an optimal solution in $O(n \log n)$ time. We also showed that $O(n \log n)$ is a lower-bound and thus the time complexity is also the best possible. Experimental results show that our algorithm performs well.

Acknowledgment

This work was partially supported by the National Science Foundation under grant CCF-0701821 and a grant from IBM.

References

[1] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

[2] M. M. Ozdal and M. D. F. Wong. Simultaneous escape routing and layer assignment for dense pcbs. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design (ICCAD)*, pages 822–829. IEEE Computer Society, 2004.

[3] M. M. Ozdal, M. D. F. Wong, and P. S. Honsinger. An escape routing framework for dense boards with high-speed design constraints. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design (ICCAD)*, pages 759–766. IEEE Computer Society, 2005.

[4] X. Tang, R. Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, pages 106–111. ACM Press, 2000.

[5] X. Tang and D. F. Wong. Fast-sp: a fast algorithm for block placement based on sequence pair. In *Proceedings of the 2001 conference on Asia South Pacific design automation (ASP-DAC)*, pages 521–526. ACM Press, 2001.

[6] T.H.Cormen, C.E.Leiserson, and R.L.Rivest. *Introduction to Algorithms*. MIT Press, 1992.

[7] D. Wiens. Printed circuit board routing at the threshold. *White Paper, Mentor Graphics*, 2000.