Fast Evaluation of Sequence Pair in Block Placement by Longest Common Subsequence Computation

Xiaoping Tang*, Ruiqi Tian*†, and D.F. Wong*
*Department of Computer Sciences, University of Texas at Austin
† Motorola Computational Technology Lab, Austin, Texas
{tang, ruiqi, wong}@cs.utexas.edu

Abstract

In [1], Murata et al introduced an elegant representation of block placement called sequence pair. All block placement algorithms which are based on sequence pairs use simulated annealing where the generation and evaluation of a large number of sequence pairs is required. Therefore, a fast algorithm is needed to evaluate each generated sequence pair, i.e. to translate the sequence pair to its corresponding block placement. This paper presents a new approach to evaluate a sequence pair based on computing longest common subsequence in a pair of weighted sequences. We present a very simple and efficient $O(n^2)$ algorithm to solve the sequence pair evaluation problem. We also show that using a more sophisticated data structure, the algorithm can be implemented to run in $O(n \log n)$ time. Both implementations of our algorithm are significantly faster than the previous $O(n^2)$ graph-based algorithm in [1]. For example, we achieve 60X speedup over the previous algorithm when input size $n = 128$.

1. Introduction

Rapid advances in integrated circuit technology have led to a dramatic increase in the complexity of VLSI circuits. According to the 1997 SIA National Technology Roadmap for Semiconductors [2], we will soon have designs in less than 0.1 micron technology with over 100 million transistors. Circuits with such enormous complexity have to be designed hierarchically. Circuit placement within each level of the hierarchy is a complex block placement problem. A good block placement solution not only minimizes chip area, it also minimizes interconnect cost which is crucial in determining circuit performance in deep submicron designs. Although block placement is a classical problem with many previous algorithms [3, 4], it remains to be a hard problem. Recently, there were two breakthroughs in block placement – two novel placement representations called sequence pair[1, 5, 6] and BSG [7] were invented. In this paper we shall only focus on the sequence pair representation.

In [1], Murata et al introduced an elegant representation of block placement called sequence pair. All block placement algorithms which are based on sequence pairs use simulated annealing where the generation and evaluation of a large number of sequence pairs is required. Therefore, a fast algorithm is needed to evaluate each generated sequence pair, i.e. to translate the sequence pair to its corresponding block placement. In [1], an $O(n^2)$ algorithm based on constructing a pair of horizontal and vertical constraint graphs and computing longest paths in both constraint graphs was used for sequence pair evaluation. In [8], Takahashi attempted to improve the speed for sequence pair evaluation and presented an $O(n \log n)$ algorithm. Unfortunately, his algorithm only determines the width and height of the block placement but not the positions of the individual blocks. This clearly limits the application of [8] to block placement since it is very important to obtain the positions of the blocks in order to compute the interconnect cost.

In this paper, we present a new approach to evaluate a sequence pair based on computing longest common subsequence in a pair of weighted sequences. We present a very simple and efficient $O(n^2)$ algorithm to solve the sequence pair evaluation problem. We also show that using a more sophisticated data structure, the algorithm can be implemented to run in $O(n \log n)$ time. Both implementations of our algorithm are significantly faster than the previous $O(n^2)$ graph-based algorithm in [1]. For example, we achieved 60X speedup over the original algorithm when the number of blocks is 128.

In the sections that follow, the concept of block placement by sequence pair is reviewed in section 2; longest common subsequence is introduced and its relation to sequence pair structure described in section 3; the longest common subsequence algorithm as well as its efficiency analysis is presented in section 4; some experimental results are presented in section 5; and finally, the concluding remarks along with further discussion about the longest common subsequence algorithm to handle the general case are in section 6.

2. Block Placement by Sequence Pair

A sequence pair is a pair of sequences of n elements representing a list of n blocks. The sequence pair structure is actually a meta-grid. Given a sequence pair (X, Y), one can construct a 45 degree oblique grid as shown in Figure 1(a). For every block, the plane is divided by the two crossing slope line into four cones as shown in Figure 1(b). Block 2 is in the right cone of block 1, then it is right to 1 (see Figure 1(c)). In general, the sequence pair imposes the relationship between each pair of blocks as follows:

$$(< ..x_i..x_j.. >, < ..x_i..x_j.. >) \quad \Rightarrow \quad x_i \text{ is left to } x_j \qquad (1)$$

$$(< ..x_j..x_i.. >, < ..x_i..x_j.. >) \quad \Rightarrow \quad x_i \text{ is below } x_j \qquad (2)$$

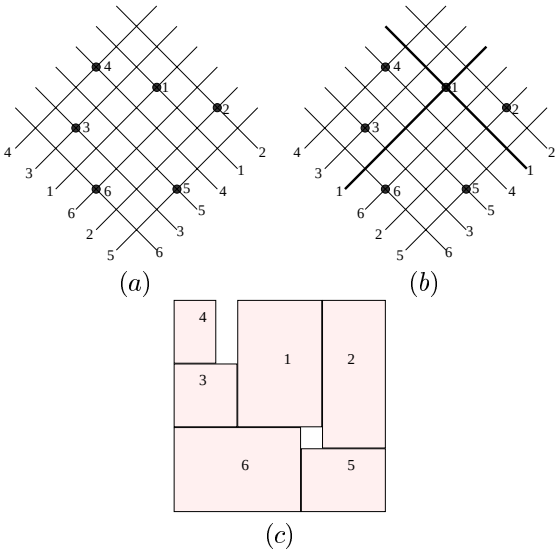

Figure 1: (a) Oblique grid for Sequence pair (<4 3 1 6 2 5>, <6 3 5 4 1 2>), (b) the four cones of block 1, and (c) the corresponding packing. The dimensions for the 6 blocks are: 1(4×6), 2(3×7), 3(3×3), 4(2×3), 5(4×3), and 6(6×4).

Consequently, given a sequence pair (X, Y), the horizontal relationship among blocks follows a horizontal-constraint graph $G_h(V, E)$, which can be constructed as follows:

1. $V = \{s_h\} \cup \{t_h\} \cup \{v_i | i = 1, ..., n\}$, where v_i corresponds to a block,
2. $E = \{(s_h, v_i) | i = 1, ..., n\} \cup \{(v_i, t_h) | i = 1, ..., n\} \cup \{(v_i, v_j) |$ block i is left to block $j\}$,
3. Vertex weight = width of block i for vertex v_i, but 0 for s_h and t_h.

The vertical-constraint graph $G_v(V, E)$ can be similarly constructed. As for the example shown in Figure 1, the corresponding constraint graphs $G_h(V, E)$ and $G_v(V, E)$ are shown in Figure 2.

Both $G_h(V, E)$ and $G_v(V, E)$ are vertex weighted, directed, acyclic graphs, so a longest path algorithm can be applied to determine the x and y coordinates of each block. The coordinates of a block are the coordinates of the lower left corner of the block.

The construction of constraint graphs G_h and G_v takes $\Theta(n^2)$ time. The longest path computation can be done in $O(n + m)$ time, where $n = \#vertices$, and $m = \#edges$ in the graph. The overall time for translating a sequence pair to a floorplan is then $\Theta(n^2)$. On the other hand, as shown in sections that follow, instead of constructing the graphs G_h and G_v followed by longest path searches, computing the longest common subsequence for the weighted sequence pair can efficiently obtain the same optimal floorplan under the constraints.

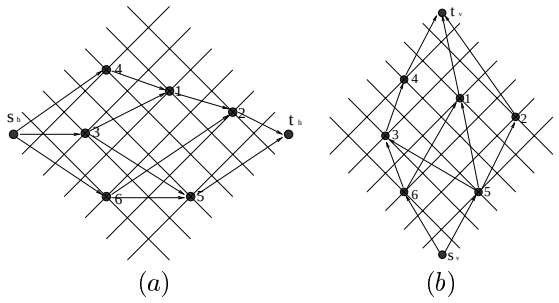

Figure 2: (a): horizontal constraint graph $G_h(V, E)$; (b): vertical constraint graph $G_v(V, E)$ for Sequence pair (X, Y) = (<4 3 1 6 2 5>, <6 3 5 4 1 2>). (for simplicity, transitive edges are omitted)

3. Longest Common Subsequence for Weighted Sequence Pair

In this section, we first introduce the definitions of weighted sequence and longest common subsequence.

Definition 1 *A weighted sequence is a sequence whose elements are in a given set S, while every element $s_i \in S$ has a weight. Let $w(s_i)$ denote the weight of s_i.*

Previously, elements in S have unit weight, i.e., $w(s_i) = 1$. In this paper, we only consider the situation $\forall s_i \in S,\ w(s_i) \geq 0$.

Definition 2 *Given two weighted sequences X and Y, a sequence Z is a common subsequence of X and Y if Z is a subsequence of both X and Y.*

Definition 3 *The length of a common subsequence $Z =< z_1, ..., z_2, ..., z_n >$ is:*

$$\sum_{i=1}^{n} w(z_i)$$

Therefore, the longest common subsequence for a sequence pair is the common subsequence of the two sequences with maximum length. In the following, let **LCS** denote longest common subsequence and $lcs(X, Y)$ denote the length of the longest common subsequence of X and Y.

Given the block set $B = \{1, ..., n\}$ and sequence pair (X, Y), a horizontal path in the packing is a common subsequence of (X, Y) as shown in Figure 3(a), and a path from s_h in horizontal constraint graph corresponds to a common subsequence of (X, Y) as shown in Figure 4(a). Let X^R denote the reverse of X. Then a vertical path in the packing is a common subsequence of (X^R, Y) as shown in Figure 3(b), and a path from s_v in vertical constraint graph corresponds to a common subsequence of (X^R, Y) as shown in Figure 4(b).

Suppose there is a block b in the sequence pair (X, Y). Let $(X, Y) = (X_1 b X_2, Y_1 b Y_2)$. Then $(X^R, Y) = (X_2^R b X_1^R, Y_1 b Y_2)$. We can see a path from s_h to b in the horizontal constraint graph corresponds to a common subsequence of (X_1, Y_1)

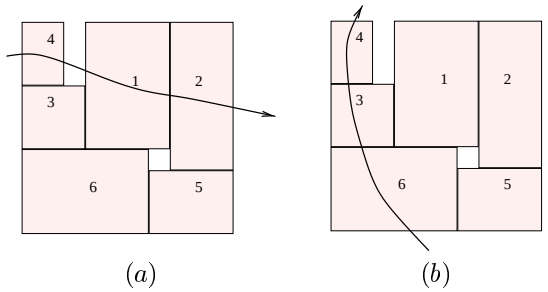

Figure 3: For the packing of Sequence pair (X, Y)=(<4 3 1 6 2 5>, < 6 3 5 4 1 2>), (a): a common subsequence <4 1 2> of (X, Y), shows horizontal constraint; (b): a common subsequence <6 3 4> of (X^R, Y)=(<5 2 6 1 3 4>, <6 3 5 4 1 2>), shows vertical constraint.

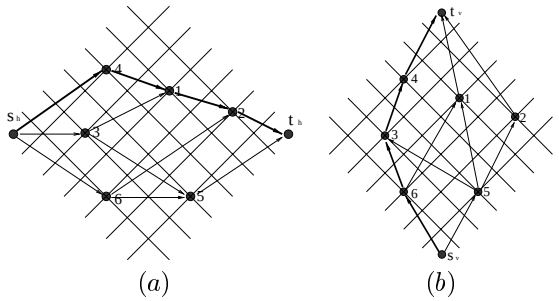

Figure 4: (a): in horizontal constraint graph $G_h(V, E)$, a path $s_h \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow t_h$ corresponds to <4 1 2>, a common subsequence of (X, Y)=(<4 3 1 6 2 5>, < 6 3 5 4 1 2>); (b): in vertical constraint graph $G_v(V, E)$, a path $s_v \rightarrow 6 \rightarrow 3 \rightarrow 4 \rightarrow t_v$ corresponds to <6 3 4>, a common subsequence of (X^R, Y)=(<5 2 6 1 3 4>, <6 3 5 4 1 2>).

(see Figure 4(a)). Similarly, a path from s_v to b in the vertical constraint graph corresponds to a common subsequence of (X_2^R, Y_1) (see Figure 4(b)). Note that the coordinates of a block is the coordinates of the lower-left corner of the block. Thus, if $w(i)$ equals the width of block i, $lcs(X_1, Y_1)$ is the x-coordinate of block b.

Lemma 1 *$\forall b \in$ the block set B, if sequence pair $(X, Y) = (X_1 b X_2, Y_1 b Y_2)$, then $lcs(X_1, Y_1)$ is the x coordinate of block b, where $w(i)$ is the width of block i, and $lcs(X, Y)$ is the width of the block placement.*

Proof As presented in [1], the longest path from vertex s_h to the vertex v_b representing b in graph $G_h(V, E)$ gives the x coordinate of block b. It is therefore suffice only to prove that the length of the longest path from s_h to v_b is equal to $lcs(X_1, Y_1)$. One observation is that the source vertex s_h of weight 0 and the sink vertex t_h of weight 0 can be safely omitted due to the fact that s_h can always be added to the head of both sequences and t_h to the tail without changing the length of common subsequence for any block.

By the definition of the constraint graph in the previous section, a path to v_b in $G_h(V, E)$ corresponds to a common subsequence of (X_1, Y_1), and vice versa. In addition, since the vertex weight in $G_h(V, E)$ for a block is equal to the weight of the corresponding element in the sequence pair, the length of the path to v_x in $G_h(V, E)$ is equal to the length of the corresponding common subsequence of (X_1, Y_1). Thus, the length of the longest path to v_x is equal to the length of the longest common subsequence of (X_1, Y_1), i.e. $lcs(X_1, Y_1)$. Since the path from s_h to t_h corresponds to a common subsequence of (X, Y), the length of the longest path from s_h to t_h is $lcs(X, Y)$. □

From the necessary and sufficient condition that

1. if b_i is after b_j in X and before b_j in Y, then b_i is before b_j in X^R and before b_j in Y, and
2. if b_i is before b_j in X^R and before b_j in Y, then b_i is after b_j in X and before b_j in Y

we can similarly conclude the following lemma:

Lemma 2 *$\forall b \in$ the block set B, if sequence pair $(X, Y) = (X_1 b X_2, Y_1 b Y_2)$, then $(X^R, Y) = (X_2^R b X_1^R, Y_1 b Y_2)$ and $lcs(X_2^R, Y_1)$ is the y coordinate of block b, where $w(i)$ is the height of block i, and $lcs(X^R, Y)$ is the height of the block placement.*

Based on the Lemma 1 and 2, we have the following theorem:

Theorem 1 *LCS computation can be applied to determine the x and y coordinates of each block and the width and height of the block placement for a given sequence pair. Therefore, the optimal packing can be obtained by computing LCS.*

Table 1: x and y coordinates of each block. X=<4 3 1 6 2 5>, Y=<6 3 5 4 1 2>, X^R=<5 2 6 1 3 4>. The dimensions for every block are: 1(4×6), 2(3×7), 3(3×3), 4(2×3), 5(4×3), and 6(6×4). See Figure 1(c) for the dimensions and the packing.

block	LCS of (X_1, Y_1)	x_coor	LCS of (X_2^R, Y_1)	y_coor
1	3	3	6	4
2	3 1	7	5	3
3		0	6	4
4		0	6 3	7
5	6	6		0
6		0		0

Table 1 shows the process that LCS is applied to determine the x and y coordinates of each block for the sequence pair (<4 3 1 6 2 5>,<6 3 5 4 1 2>).

4. A Fast LCS Algorithm

A fast algorithm for computing longest common subsequence was introduced in [9]. However, it only handles sequences with unit weight. The representation they used is tightly based on unit weight, and can not be adapted to the presence of weights. In the rest of this section, we first propose a very simple and efficient $O(n^2)$ algorithm to compute LCS for a given sequence pair where n is the number of elements and the weights are not restricted to be 1 or integers. Then, we show that using more sophisticated data structure, the algorithm can be improved to run in $O(n \log n)$ time.

Assume the blocks are $1...n$, and the input sequence pair is (X, Y). Both X and Y are then a permutation of $\{1...n\}$. Block position array $P[b], b = 1...n$ is used to record the x or y coordinate of block b depending on the weight $w(b)$ equals to the width or height of block b respectively. To record the indices in both X and Y for each block b, the array $match[b], b = 1...n$ is constructed to be $match[b].x = i$ and $match[b].y = j$ if $b = X[i] = Y[j]$. The length array $L[1...n]$ is used to record the length of candidates of the longest common subsequence. The algorithm is as follows.

Algorithm 1:

```
1  Initialize_Match_Array match;
2  Initialize_Length_Array L with 0;
3  for i = 1 to n
4     do b = X[i];
5        p = match[b].y;
6        P[b] = L[p];
7        t = P[b] + w(b);
8        for j = p to n
9           do if (t > L[j])
10             then L[j] = t;
11             else break;
12 return L[n];
```

As an example, Figure 5 shows the steps algorithm 1 took to compute the LCS of Sequence pair(<4 3 1 6 2 5>, <6 3 5 4 1 2>) in determining the x-coordinate of each block. The weights of blocks are their widths as shown in Figure 1. When the algorithm ends, the array $P[1...6]$ records the blocks' x-coordinates.

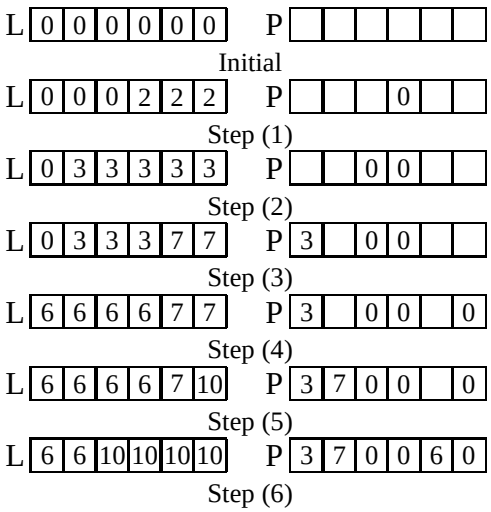

Figure 5: An Example of Algorithm 1 to compute the LCS of Sequence pair (<4 3 1 6 2 5>, <6 3 5 4 1 2>). The dimensions of the 6 blocks are shown in Figure 1. Here $w(i)$ = width of block i. L : candidates of LCS; P : positions of blocks

Theorem 2 *Algorithm 1 correctly returns $lcs(X, Y)$ and the position of each block is correctly recorded in $P[1...n]$.*

Proof We use an *induction* on i to prove that the positions of the blocks in subsequence $X[1...i]$ are correctly recorded.

Initially, all the $L[j](j = 1, ...n)$ is 0. When $i = 1$, the position of block b ($= X[1]$) is 0. Assume that all the positions of the blocks in $X[1...k]$ are correctly recorded. When $i = k + 1$, we let b_1 denote the block $X[k + 1]$, and p_1 denote the block's index in the sequence Y, i.e.

$$match[b_1].x = k + 1 \quad match[b_1].y = p_1$$

Therefore, according to *Lemma 1* and *2*, the position of block b_1 is $lcs(X[1...k], Y[1...(p_1 - 1)])$. Assume the last element of the LCS is b_2. Thus, $match[b_2].x \leq k$ and $match[b_2].y \leq p_1 - 1$. We get

$$\begin{aligned} & lcs(X[1...k], Y[1...(p_1 - 1)]) \\ = \; & lcs(X[1...(match[b_2].x - 1)], \\ & \quad Y[1...(match[b_2].y - 1)]) + w(b_2) \end{aligned}$$

Since all the positions of the blocks in $X[1...k]$ are correctly recorded according to the induction assumption, we get

$$\begin{aligned} P[b_2] \; = \; & lcs(X[1...(match[b_2].x - 1)], \\ & \quad Y[1...(match[b_2].y - 1)]) \end{aligned}$$

Then, $lcs(X[1...k], Y[1...(p_1 - 1)]) = P[b_2] + w(b_2)$.

The code from line 7 to line 11 guarantees that $P[b_2] + w(b_2)$ is recorded in $L[p_1]$.

When the algorithm ends, $L[n]$ records the maximal value of the position of block b plus its weight ($b = 1, ...n$), which is $lcs(X, Y)$. □

Initializing the match array can be done in $O(n)$ time by scanning the two sequences. The running time of Algorithm 1 is $O(n^2)$, and the space requirement is $O(n)$. In spite of its $O(n^2)$ running time, the code is very tight, so the hidden constant factor in its running time is small.

Note that the code from line 8 to line 11 actually records the greatest value for later queries. If we use *balanced search tree (BST)* to manage the array $L[1...n]$, we can achieve the $O(n \log n)$ running time. Every node in the BST is associated with two keys: (index, length), and both keys must be kept in increasing (non-decreasing) order. We use *index* as the primary key , and discard the node whose *length* is not increasing since it makes no contribution to subsequent LCS computation. The extra time charged for discarding nodes can be **amortized**. The algorithm is shown below.

Algorithm 2:

1 Initialize_Match_Array $match$;
2 Initialize BST with a node (0,0);
3 for $i = 1$ to n
4 do $b = X[i]$;
5 $p = match[b].x$;
6 $P[b]$=find_BST(p);
7 $t = P[b] + w(b)$;
8 insert (p, t) to BST;
9 discard the nodes with greater index than p
and less length than t;
10 return find_BST(n);

find_BST(p):
1 find the greatest index in BST which is less than p;
2 return the corresponding length.

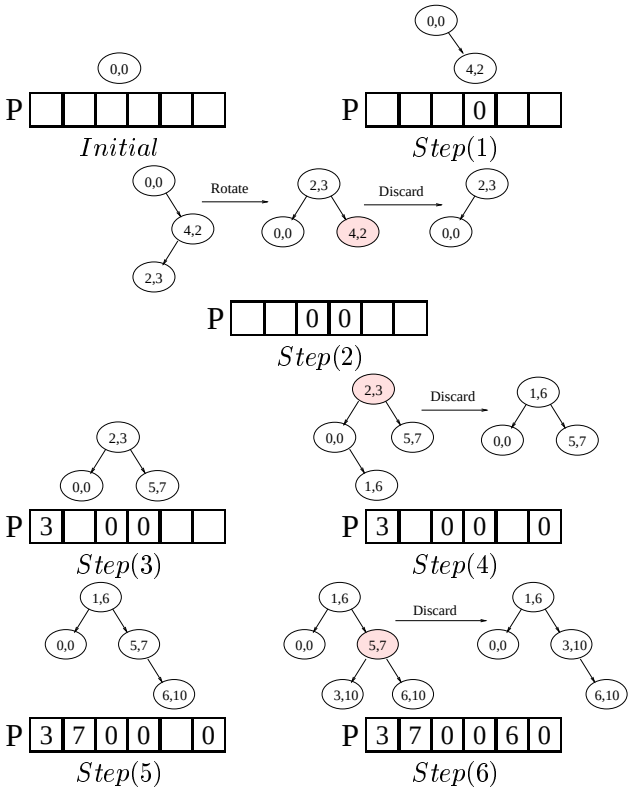

Figure 6: An Example of Algorithm 2 to compute the LCS of Sequence pair (<4 3 1 6 2 5>, <6 3 5 4 1 2>). The dimensions of the 6 blocks are shown in Figure 1. Here $w(i)$ = width of block i. P : positions of blocks.

As an example, Figure 6 shows the steps Algorithm 2 took to compute the LCS of sequence pair(<4 3 1 6 2 5>,<6 3 5 4 1 2>) in determining the x-coordinate of each block. The weights of blocks are their widths as shown in Figure 1. Like Algorithm 1, the array $P[1...6]$ is used to record the x-coordinate of each block.

Theorem 3 *Algorithm 2 finds the $lcs(X, Y)$ and records the position of each block in $O(n \log n)$ running time with $O(n)$ space requirement.*

Proof The correctness of the algorithm can be observed since $lcs(X[1...i], Y[1...j])$, where $X[i] = Y[j]$, is indicated by the element (j, t) in the BST. Similar to Algorithm 1, the algorithm returns *find_BST(n)*, which is exactly $lcs(X, Y)$.

Now we use **amortized analysis** to prove that its running time is $O(n \log n)$. In line 3, the loop has n iterations. Line 4, 5, and 7 take $O(1)$ time each. Line 6 takes $O(\log n)$ time. In line 8, the tree may need re-balance after inserting (p, t), which takes at most $O(\log n)$ time. In line 9, discarding one element from BST and re-balancing it take $O(\log n)$ time. Suppose there are d_k elements discarded during the kth iteration. Thus, the total running time is

$$O(n \log n) + \sum_{k=1}^{n} d_k \cdot \log n$$

Since there are at most n elements discarded, $\sum_{k=1}^{n} d_k \leq n$. Therefore, the total running time is $O(n \log n)$.

Clearly, the space requirement is $O(n)$. □

5. Experimental Results

In this section, we design experiments to compare our algorithm with original algorithm in two aspects. In Experiment 1, we compare the runtimes in evaluating a single sequence pair. In Experiment 2, we compare the runtimes in obtaining block placement. Both experiments are done on Sun-Ultra Enterprise3000 (200MHz).

5.1. Experiment 1

We randomly generated test cases with 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, and 16384 blocks. Table 2 lists the average running time (in second) for the original algorithm and our algorithms. The original algorithm uses an efficient method to construct the constraint graphs in $O(n^2)$ time, and uses an algorithm similar to that in [10] to find the longest path length to all vertices from source vertex in $O(n + m)$ time where $n = \#blocks(vertices)$, $m = \#edges$.

From Table 2, we can see that both Algorithm 1 and Algorithm 2 are significantly faster than the original algorithm. For example, we achieved 60X speedup over the original algorithm when the number of blocks is 128. When the number of input blocks is more than 8192, Algorithm 2 outperforms Algorithm 1. However, Algorithm 2 is slower than Algorithm 1 when the input size is small. The reason is that Algorithm 2 has the overhead of dynamic memory allocation and deallocation in maintaining the balanced search tree. When the input size gets larger, Algorithm 2 begins to exploit its $O(n \log n)$ running time characteristic.

5.2. Experiment 2

The experiments are carried out for the MCNC benchmarks. The evaluation of a sequence pair is a weighted sum of the

Table 2: time comparison for single computation between original algorithm and our algorithms.

block#	original(s)	alg 1(s)	alg 2(s)
16	5.50e-05	3.39e-06	5.55e-05
32	1.92e-04	7.21e-06	1.13e-04
64	7.25e-04	1.63e-05	2.34e-04
128	2.83e-03	4.47e-05	4.84e-04
256	1.13e-02	1.13e-04	9.95e-04
512	5.68e-02	3.29e-04	2.03e-03
1024	0.249	8.87e-04	4.13e-03
2048	1.06	2.45e-03	8.42e-03
4096	4.37	9.29e-03	1.73e-02
8192	19.6	2.93e-02	3.82e-02
16384	121	9.47e-02	8.88e-02

Table 3: Performance comparison for floorplanning experiments between the original and ours.

data	#mod	#net	original(s)	ours (s)	speedup
ami33	33	123	897	182	5X
ami49	49	408	1956	503	4X

area of the packing and the total wire length based on the half perimeter estimate of bounding box for each net, where terminals are assumed to be at the center of each block. The cost function C is defined as:

$$C = A + \lambda \cdot W$$

where A is the area, and W is the wiring cost. The weight parameter λ is used in the cost function for balancing the two factors. The initial temperature is decided such that the initial acceptance rate is greater than 95%. Note that, since we are only replacing the original sequence pair evaluation algorithm with our algorithm, the number of iterations and the final solutions are identical.

The comparison result is shown in Table 3. CPU-time and speedup are listed. Notably, the wiring cost estimations for the original and our method are the same. It accounts for a non-trivial part in the running time. For example, if wiring cost estimation accounts for 1/3 of running time for the original algorithm, then we could achieve at most a 3X speedup even if the time for the longest path computation is reduced to 0. In spite of this, as we can see, our method still outperforms the original one and achieved 5X speedup for ami33 and 4X speedup for ami49.

6. Concluding Remarks

Sequence pair is an elegant topology representation of block placement. In this paper, we have proposed a new algorithm to translate a sequence pair to a block placement. We demonstrate that its running time could be $O(n \log n)$ by using more sophisticated data structure. Given a sequence pair, the algorithm not only evaluate the area of the placement, but also gives the position of each block, which is essential to wire length estimation in practical VLSI placement. Experimental results show its efficiency.

Our LCS algorithm is motivated by translating a sequence pair to a block placement. So the algorithm presented above mainly handles the situation that each of the two sequences has the same length and is a permutation of n blocks. In the general case, the two sequences may have different length or repeated elements. For example, X=< a b c b b a >, Y=< b a b d a d b>. Both Algorithm 1 and Algorithm 2 can be easily expanded to deal with the general case.

References

[1] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence pair". *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, vol. 15:12, pp. 1518-1524, 1996.

[2] Semiconductor Industry Association, National Technology Roadmap for Semiconductors, 1997.

[3] N. Sherwani, Algorithms for VLSI Physical Design Automation, Kluwer Academic Publishers, Boston, 1995.

[4] M. Sarrafzadeh and C.K. Wong, An Introduction to VLSI Physical Design, McGraw Hill, 1996.

[5] H. Murata, K. Fujiyoshi, and M. Kaneko, "VLSI/PCB placement with obstacles based on sequence pair", ISPD-97, pp. 26-31, 1997.

[6] H. Murata, and E.S. Kuh, "Sequence-Pair Based Placement Method for Hard/Soft/Pre-placed Modules", ISPD-98, pp. 167-172, 1998.

[7] S. Nakatake, H. Murata, K. Fujiyoshi, and Y. Kajitani, "Module Placement on BSG-Structure and IC Layout Applications", Proceedings of ICCAD-96, Nov. 1996.

[8] T. Takahashi, "An Algorithm for Finding a Maximum-Weight Decreasing Sequence in a Permutation, Motivated by Rectangle Packing Problem ", *Technical Report of IEICE, VLD96*, vol. VLD96, No. 201, pp. 31-35, 1996.

[9] J.W. Hunt, and T.G. Szymanski, "A fast algorithm for computing Longest Common Subsequences". *Communications of the ACM*, Vol. 20:5, pp. 350–353, May 1977.

[10] J.A. McHugh, "Algorithmic Graph Theory", Prentice Hall (1990).

[11] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, " Optimization by simulated annealing". *Science*, vol. 220, pp. 671-680, 1983.