FAST-SP: A Fast Algorithm for Block Placement based on Sequence pair

Xiaoping Tang and D.F. Wong
UT Austin

Mingju Liu

Department of Electrical and Computer Engineering

University of Utah, Salt Lake City, UT

Background: Problem to solve

- Rapid advances in integrated circuit technology have led to a dramatic increase in the complexity of VLSI circuits.
- We need a good block placement solution to not only minimize chip area, but also minimize interconnect cost while satisfying all placement constraint.
- Although Block placement is a classical problem with many state-of-art solutions, it remains to be a hard problem.

Motivation: Why Should We Care

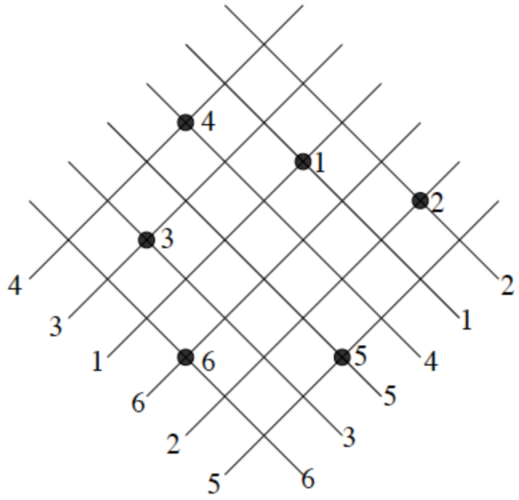
- Existing solutions are not fast enough:
  - Murata et al introduced sequence pair to represent block placement elegantly which constructs a pair of horizontal and vertical constraint graphs and computing longest paths in both graphs with $\mathcal{O}(n^2)$ time.
  - Tang et al proposed an $\mathcal{O}(n \log n)$ algorithm based on evaluation of sequential pair with computing longest common subsequence.
  - New representations such as O-tree and B*-tree produce better results than sequence-pair based algorithms due to sequence pair's inherently larger solution space.

Contributions

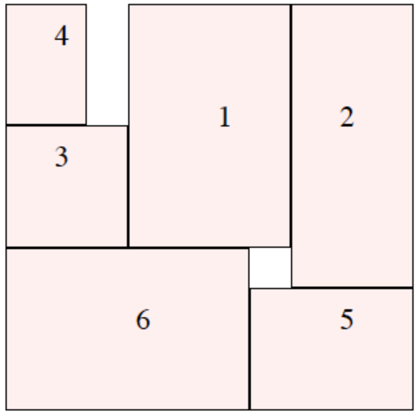
- This paper presents FAST-SP: a fast block placement algorithm based on the sequence-pair representation. Two main contributions are discussed:
  - Fast evaluation of sequence pair: Since all sequence-pair based algorithms are based on simulated annealing, a large number of sequence pairs are needed to be generated and evaluated. FAST-SP can reach $\mathcal{O}(n \log\ \log n)$ runtime and examine more sequence pairs.
  - Handle placement constraints: No previous sequence-pair based algorithm can handle placement constraint such as pre-placed constraint, range constraint, and boundary constraint.

Recap: Block Placement by Sequence Pair

- Sequence pair (X, Y) gives the relative positions of the blocks. A horizontal or vertical constraint graph $G_h(V, E)$ or $G_v(V, E)$ can be constructed as follows: (take horizontal as example)
  - $V = \{s_h\} \cup \{t_h\} \cup \{v_i \mid i = 1,\ldots,n\}$
    $E = \{(s_h, v_i) \mid i = 1,\ldots n\} \cup \{(v_i, t_h) \mid i = 1,\ldots n\}$
  - $\cup \{(v_i, v_j) \mid$ block i is to the left of block $j\}$
  - Vertex weight = width of block i for vertex v_i, 0 for s_h and t_h
- Longest path algorithm can be applied to determine the coordinates of each block. As for runtime, construction of constraint graphs takes $\Theta(n^2)$ time and longest path computation takes $\mathcal{O}(n + m)$ time overall time is then $\Theta(n^2)$.



(a)



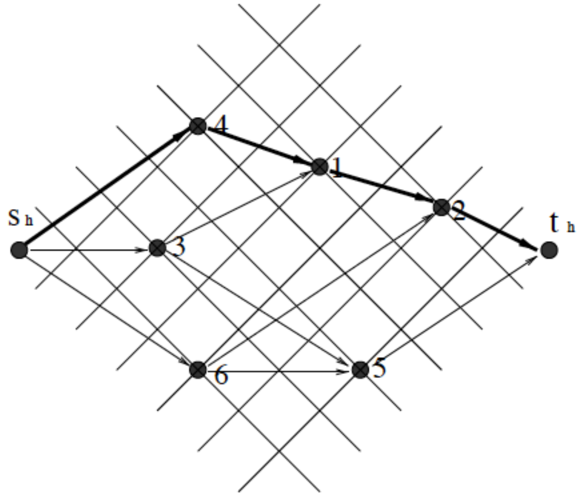
(b)

Technical Approaches

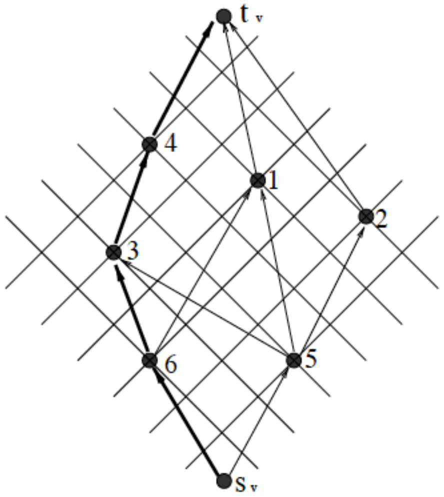
- FAST-SP is based on the evaluation of Longest Common Subsequence (LCS) for Weighted Sequence Pair.
- Weighted sequence is a sequence with every element s_i has a weight $w(s_i) \geq 0$.
- A common subsequence Z of weighted sequences X and Y is a subsequence of both X and Y, the length of Z is $\sum_{i=1}^{n} w(z_i)$.
    - $< 1\ 2 >$ is the common subsequence of $< 1\ 5\ 2 >$ and $< 4\ 1\ 2\ 5 >$

Technical Approaches (cont'd)

- For given sequence pair (X, Y), a path from s_h in horizontal constraint graph G_h corresponds to a common subsequence of (X, Y). For vertical graph, a path from s_v corresponds to (X^R, Y).
- E.g.



(a)



(b)

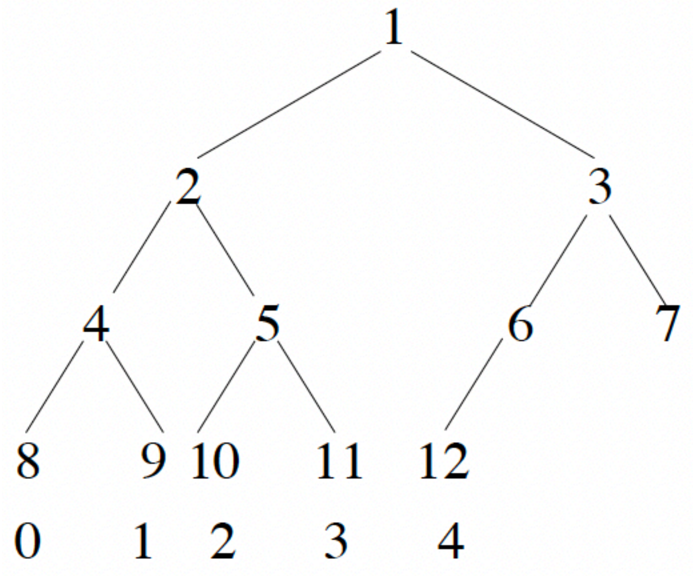
$(X, Y) = (<4\ 3\ 1\ 6\ 2\ 5>, <6\ 3\ 5\ 4\ 1\ 2>)$ $(X^R, Y) = (<5\ 2\ 6\ 1\ 3\ 4>, <6\ 3\ 5\ 4\ 1\ 2>)$

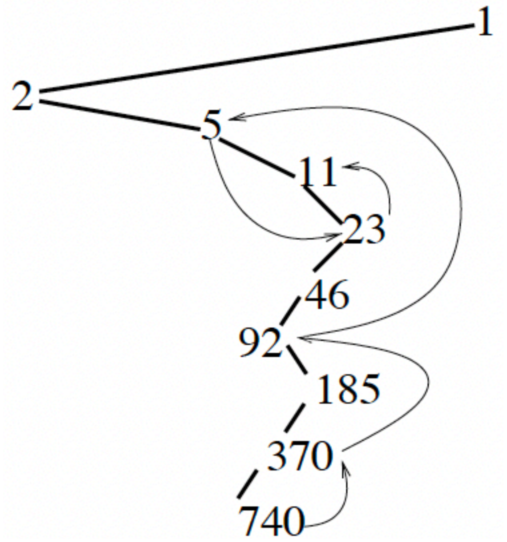
Technical Approaches (cont'd)

- Prove the equivalence of sequence pair evaluation and longest common subsequence computation:
  - Suppose a block b in the sequence pair (X, Y). Let $(X, Y) = (X_1 b X_2, Y_1 b Y_2)$, $(X^R, Y) = (X_2^R b X_1^R, Y_1 b Y_2)$
  - For horizontal Constraint graph, a path from s_h to b corresponds to a common subsequence of (X_1, Y_1); Vertically, a path from s_v to b corresponds to (X_2^R, Y_1).
  - If $w(i) =$ width of block i, $lcs(X_1, Y_1)$ is the x-coordinate of block b. Then $lcs(X, Y)$ is the width of the block placement.
  - If $w(i) =$ height of block i, $lcs(X_2^R, Y_1)$ is the y-coordinate of block b. Then $lcs(X^R, Y)$ is the height of the block placement.

Technical Approaches (cont'd)

- FAST-SP uses Priority Queue and Bucket List Data structure to store and sort LCS. Priority Queue can be represented by a complete binary tree H. The lowest leaf nodes correspond to the index of bucket node.
- H is represented as $\{1,...,2^h+n\}$ where n is the size of index domain and $h=\lceil \log(n+1) \rceil$ is the height of the tree. A bucket on the bucket list corresponds to the path $(1 \rightarrow f)$ of the related leaf f.
- Let D become the length of the smallest interval between the indices in the bucket list covering the index newly inserted or deleted. It has been proved that runtime of insertion or deletion on Priority Queue is $\mathcal{O}(\log \log D)$.





Technical Approaches (cont'd)

- $MATCH(b)$ gives the index of block b in the Sequence, e.g. $MATCH(b).x = i$ and $MATCH(b).y = j$ if $b = X[i] = Y[j]$
- $POS(b)$ records the x or y coordinate of b.
- $BUCKL[index]$ records the length of candidates of the longest common subsequence.
- $BUCKL[index_{max}]$ reports $lcs(X, Y)$ in $\mathcal{O}(n \log \log n)$ time with $\mathcal{O}(n)$ space requirement.

Algorithm Eval-Seq(X,Y)

1. Initialize_Match_Array $MATCH$;
2. Initialize H, insert the initial index 0;
3. Initialize $BUCKL$ with $BUCKL[0] = 0$;
4. **FOR** $i = 1$ **TO** n **DO**
5. $\quad b = X[i]$;
6. $\quad p = MATCH[b].y$;
7. $\quad$ insert p to H and $BUCKL$;
8. $\quad POS[p] = BUCKL[predecessor(p)]$;
9. $\quad BUCKL[p] = POS[p] + w(b)$;
10. $\quad$ discard the successors of p from H and $BUCKL$ whose value $\leq BUCKL[p]$;
11. **RETURN** $BUCKL[index_{max}]$;

Technical Approaches (cont'd)

- We know location of $b = lcs(X[1,...,i-1], Y[1,...,j-1])$ if we assume b' is the last element of the above LCS, then $MATCH[b'].x \le i-1$ and $MATCH[b'].y \le j-1$

  $$lcs(X[1,...,i-1], Y[1,...,j-1])$$

  then $= lcs(X[1,...,MATCH[b'].x-1],$

  $$Y[1,...,MATCH[b'].y-1) + w(b')$$
- $predecessor(p)$ will be $MATCH[b'].y$
- Line 10 is used to delete the element in the bucket list with higher index but less value to LCS computation to make sure the algorithm return $lcs(X, Y)$

Algorithm Eval-Seq(X,Y)

1. Initialize_Match_Array $MATCH$;
2. Initialize H, insert the initial index 0;
3. Initialize $BUCKL$ with $BUCKL[0] = 0$;
4. **FOR** $i = 1$ **TO** n **DO**
5. $\quad b = X[i]$;
6. $\quad p = MATCH[b].y$;
7. $\quad$ insert p to H and $BUCKL$;
8. $\quad POS[p] = BUCKL[predecessor(p)]$;
9. $\quad BUCKL[p] = POS[p] + w(b)$;
10. $\quad$ discard the successors of p from H and $BUCKL$ whose value $\le BUCKL[p]$;
11. **RETURN** $BUCKL[index_{max}]$;

Technical Approaches (cont'd)

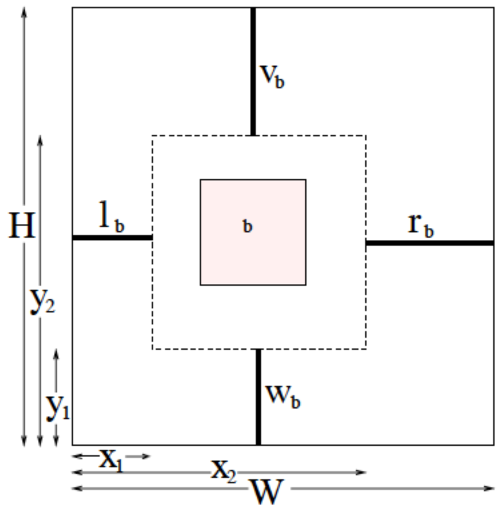
- For the space requirement, since $H = \{1,...,2^h + n\}$ and $h = \lceil \log(n+1) \rceil$, we have $2n + 1 \le H < 3n + 2$ hence the $\mathcal{O}(n)$.
- For the runtime,
  - Initialization would take $\mathcal{O}(n)$ time
  - Line 7 and 10 would take $\mathcal{O}(\log\ \log D)$ as discussed before.
  - At most n node discarded so that we confirm the runtime of $\mathcal{O}(n \log\ \log n)$.

Algorithm Eval-Seq(X, Y)

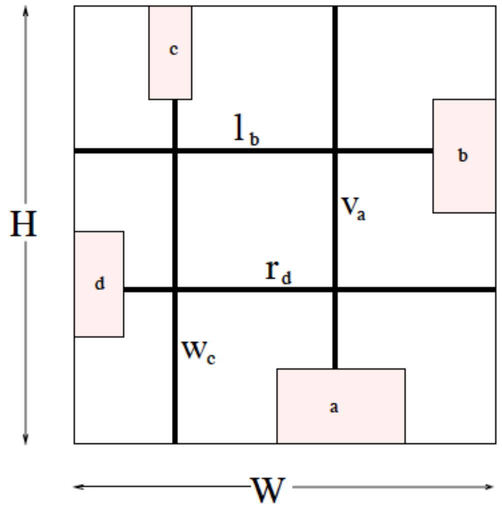
```
1.   Initialize_Match_Array MATCH;
2.   Initialize H, insert the initial index 0;
3.   Initialize BUCKL with BUCKL[0] = 0;
4.   FOR i = 1 TO n DO
5.       b = X[i];
6.       p = MATCH[b].y;
7.       insert p to H and BUCKL;
8.       POS[p] = BUCKL[predecessor(p)];
9.       BUCKL[p] = POS[p] + w(b);
10.      discard the successors of p from H and BUCKL
           whose value ≤ BUCKL[p];
11.  RETURN BUCKL[index_max];
```

Technical Approaches (cont'd)

- Pre-place constraint: for a block b and a point (x_1, y_1), block b must be placed with its lower-left corner at the point.
- Range constraint: block b must be placed at the range $\{x_1 \le x \le x_2, y_1 \le y \le y_2\}$ (Pre-place is a special case)
- Boundary constraint: block b must be placed at the side of the final packing.
- Dummy blocks are introduced.



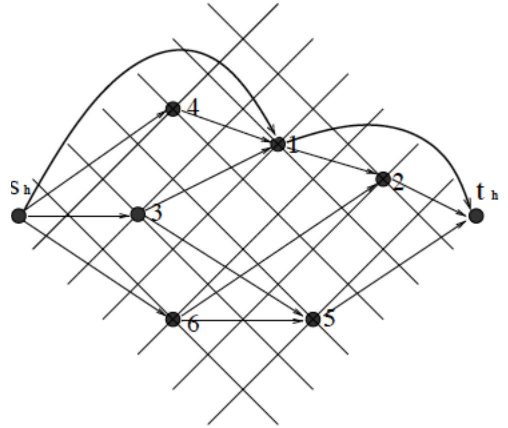
(a) Range

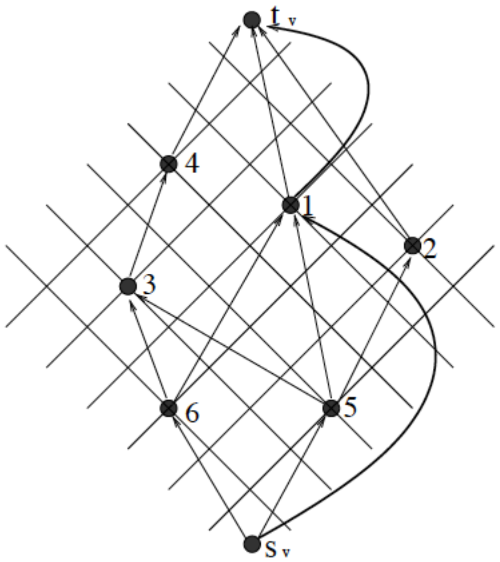


(b) Boundary

Technical Approaches (cont'd)

- Dummy blocks would not show in sequence pair but add additional edges to meet the constraints.
- However, when adding such constraints, there may not exist packing for some sequence pair which makes it infeasible pair.
- More specifically, A sequence pair (X, Y) is feasible if and only if the length of the longest path from s_h to t_h in G_h is no more than W and the length of the longest path from s_v to t_v in G_v is no more than H.





Technical Approaches (cont'd)

- Modified algorithm without runtime penalty.
- A "sink" variable t is introduced to record the intermediate lcs imposed by dummy blocks in placement constraints.

Algorithm Eval-Seq(X,Y)

1. Initialize_Match_Array $MATCH$;
2. Initialize H, insert the initial index 0;
3. Initialize $BUCKL$ with $BUCKL[0] = 0$;
4. $t = 0$;
5. **FOR** $i = 1$ **TO** n **DO**
6. $\quad b = X[i]$;
7. $\quad p = MATCH[b].y$;
8. $\quad$ insert p to H and $BUCKL$;
9. $\quad POS[p] = BUCKL[predecessor(p)]$;
10. $\quad$ if l_b exists, $POS[p] = max(POS[p], width(l_b))$;
11. $\quad BUCKL[p] = POS[p] + w(b)$;
12. $\quad$ if r_b exists, $t = max(t, BUCKL[p] + width(r_b))$;
13. $\quad$ discard the successors of p from H and $BUCKL$ whose value $\leq BUCKL[p]$;
14. **RETURN** $max(t, BUCKL[index_{max}])$;

Technical Approaches (cont'd)

- A unified cost function is introduced:
- From the return value (set as $lcs'(X, Y)$)from the modified algorithm, we can get the area for the given sequence pair.
  - $A = lcs'(X, Y) \cdot lcs'(X^R, Y)$
  - The unified cost function will be $C = \alpha A + \beta W$.
  - With balance factor α and β and interconnect cost W.

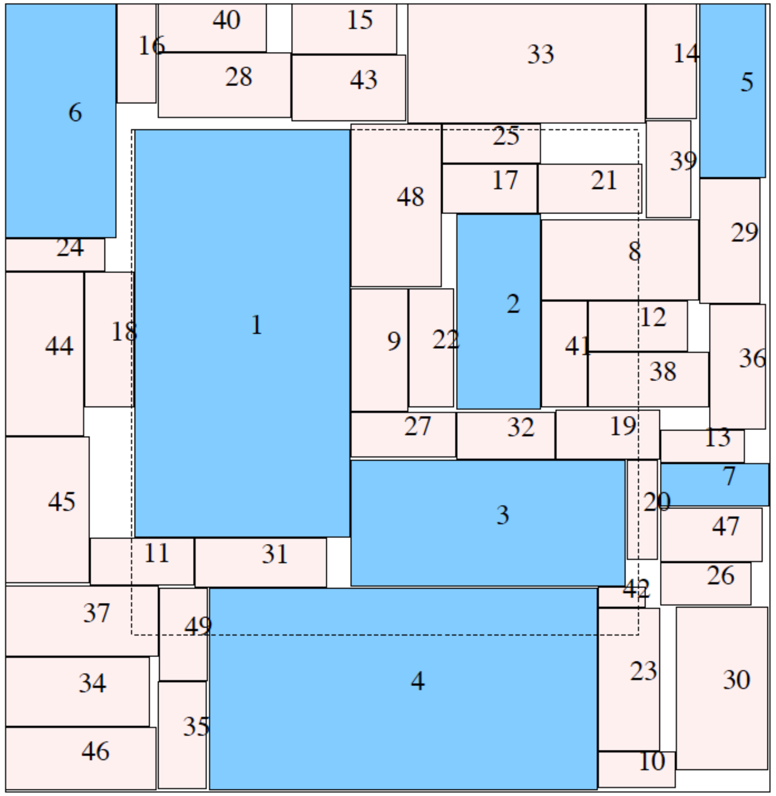
Experimental Results

circuit	block	O-tree		B*-tree		FAST-SP	
		time(Ultra60) (s)	area (mm^2)	time(Ultra1) (s)	area (mm^2)	time(Ultra1) (s)	area (mm^2)
apte	9	11	46.92	7	46.92	1	46.92
xerox	10	38	20.21	25	19.83	14	19.80
hp	11	19	9.159	55	8.95	6	8.947
ami33	33	119	1.242	3417	1.27	20	1.205
ami49	49	526	37.73	4752	36.80	31	36.50

- O-tree and B*-tree have reported the best results for these benchmarks.
- FAST-SP outperforms the other two methods.

Experimental Results

- FAST-SP can handle problems with placement constraints.



The result packing of ami49.

Pros and Cons of the Work

- Pros:
  - FAST-SP improves the runtime of evaluating a sequence pair significantly to $\mathcal{O}(n \log \log n)$.
  - It can also handle placement constraints without increasing runtime.
- Cons:
  - The proposed method uses LCS method which might have some limitations, e.g., to handle rectilinear shape constraint.
  - The solution may be sub-optimal for other constraints, such as minimizing wire length, routing congestion and buffer allocation.

Summary

- A fast block placement algorithm based on sequence pair — FAST-SP is presented.
- FAST-SP can reach a significant lower runtime $\mathcal{O}(n \log \log n)$ and can also handle some placement constraints such as pre-placed constraint, range constraint and boundary constraint without runtime penalty.
- A unified cost function was derived for the evaluation.
- Experimental results proved the significant improved performance of FAST-SP.