Stockmeyer Algorithm

Vee Yeow Edwin Benedict Khoo

Introduction

- Given a slicing floorplan, optimally determine the rotation of each block to minimize overall area
- Used as a post-process to further optimize the area objective
- Traverse the tree bottom-up and compute the candidate dimensions of each internal node
- At the root node, select the candidate dimension with the smallest area and traverse the tree top-down
  - Select the corresponding dimension for internal nodes
  - Leaf node's rotation determined from parent node's dimension

Implementation

- Leaf nodes contain a list of dimensions before/after rotation

- Internal nodes sort their children's list
  - If parent node is vertical, sort with increasing width, decreasing height. Else sort with increasing height, decreasing width

- If internal node is vertical, merging the dimensions of the left child (w_L, h_L) and right child (w_R, h_R) gives ($w_L + w_R$, $\max\{h_L, h_R\}$)
  - If $h_L < h_R$, merge the current candidate of the left child with the next candidate of the right child
  - Else if $h_L > h_R$, merge the next candidate of the left child with the current candidate of the right child
  - Else if $h_L = h_R$, merge the next candidates of both children
  - Terminate merging when any child's list has been exhausted

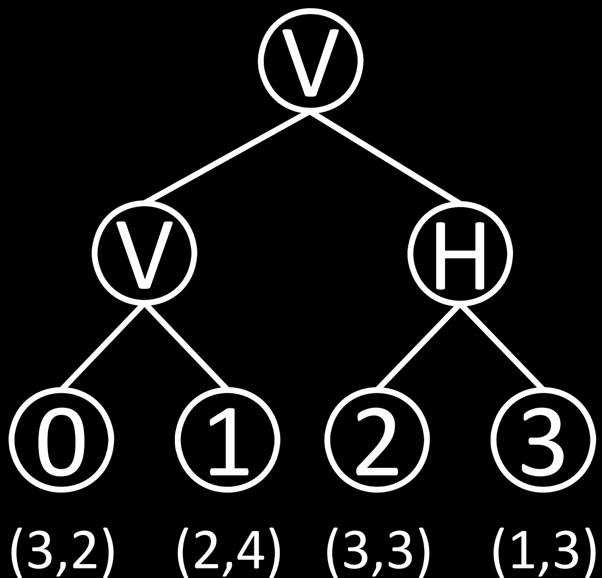
Implementation

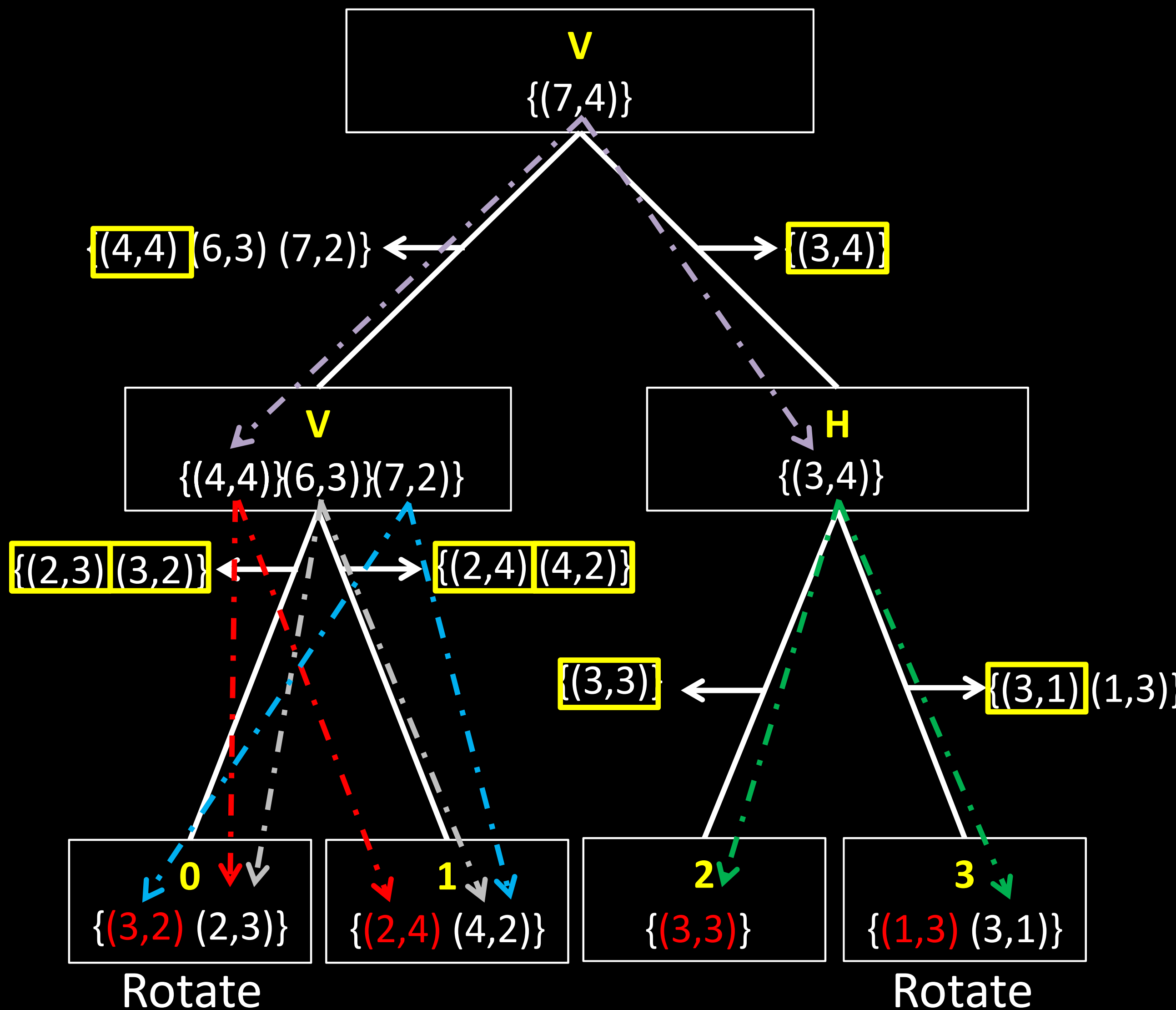
- Similar merging process with horizontal internal nodes
  - Merging the dimensions of the left child (w_L, h_L) and right child (w_R, h_R) gives ($\max\{w_L + w_R\}$, $h_L + h_R$)
  - If $w_L < w_R$, merge the current candidate of the left child with the next candidate of the right child
  - Else if $w_L > w_R$, merge the next candidate of the left child with the current candidate of the right child
  - Else if $w_L = w_R$, merge the next candidates of both children
  - Terminate merging when any child's list has been exhausted

- Merging process ensures the internal node's list is sorted
  - If vertical, list has been sorted with increasing width, decreasing height
  - Else, list has been sorted with increasing height, decreasing width
  - Implies no need to sort child's list if parent and child are internal nodes of the same type

Example

- Assume xyV implies x (left) y (right), xyH implies x (top) y (bottom)



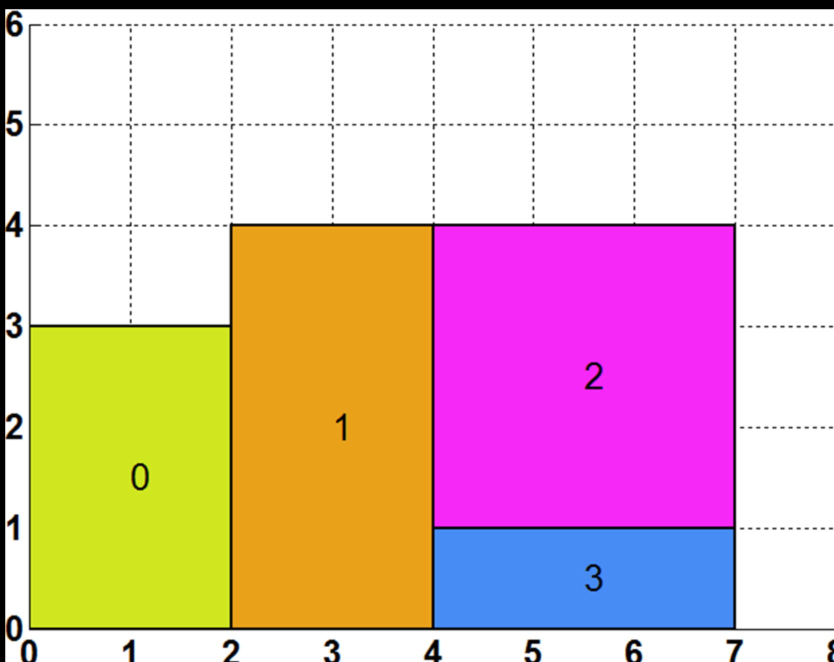


Example

- Assume xyV implies x (left) y (right), xyH implies x (top) y (bottom)

Before rotation, area = 48

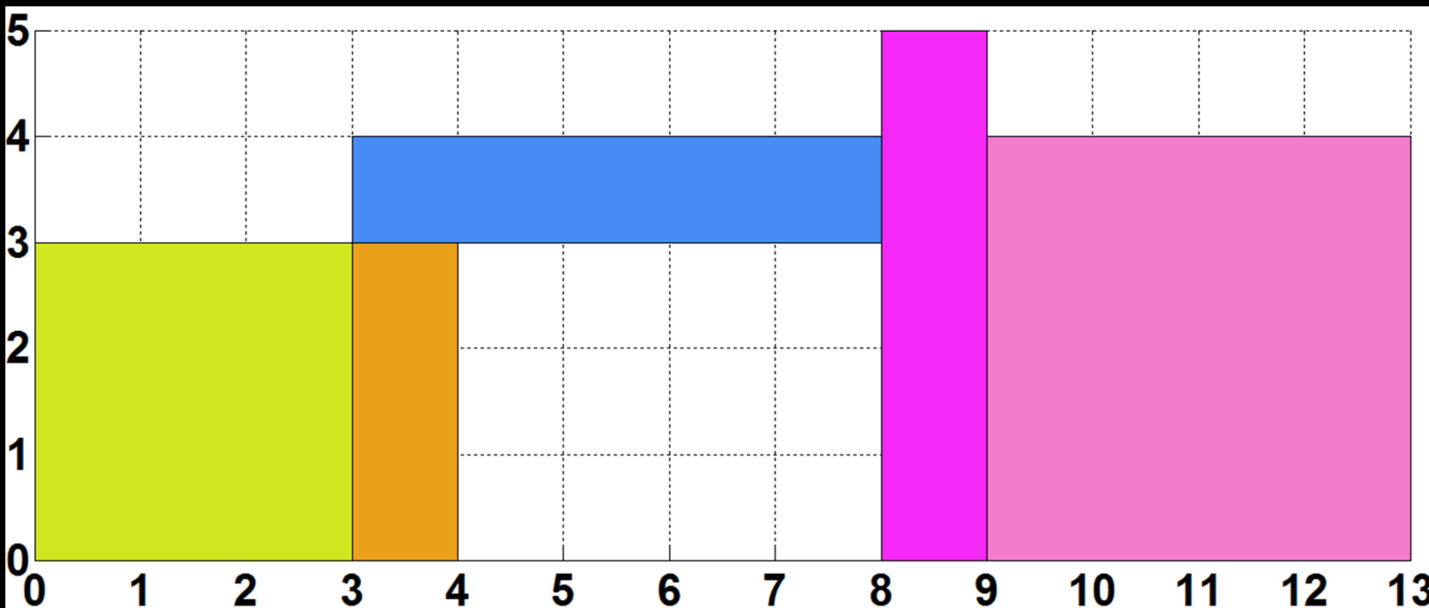
After rotation, area = 28



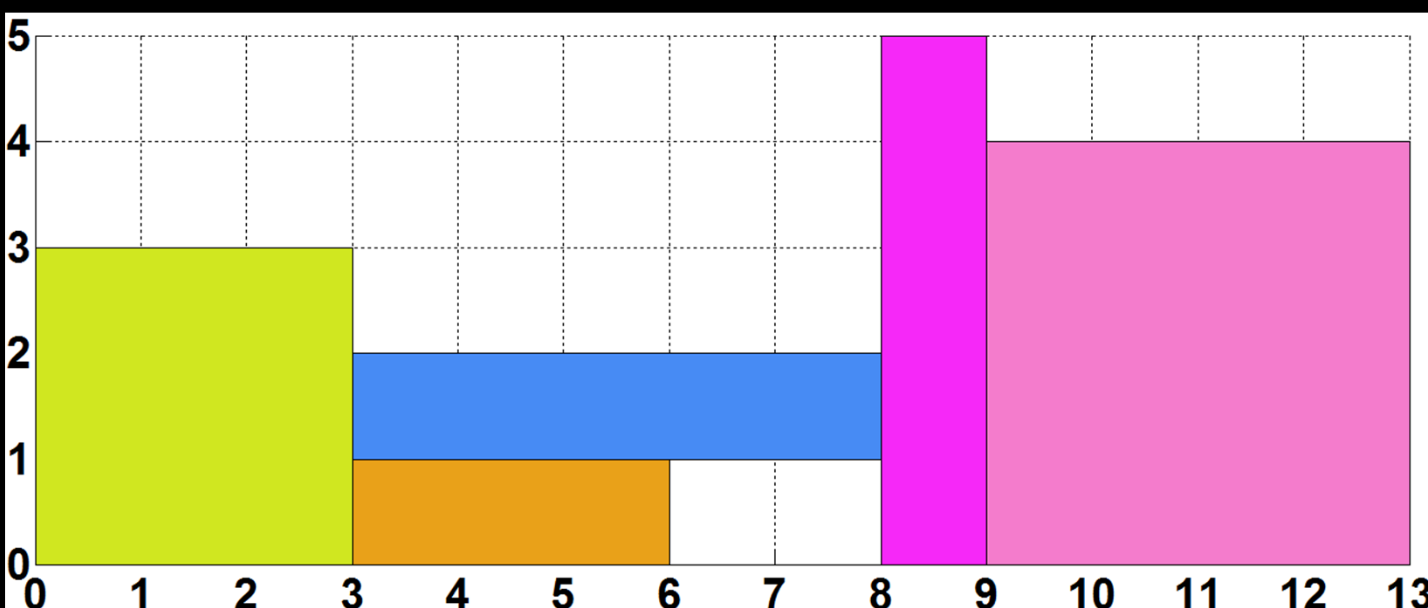
Results

# Blocks	Pre-Rotation Area	Post-Rotation Area	Area Reduction (%)	# Blocks Rotated	Rotated Blocks	Runtime (ms)
5	65	65	0	1	0	0.303
10	147	95	35.4	4	1, 3, 5, 7	0.507
30	1075	748	30.4	9	2, 15, 16, 18, 22, 23, 24, 27, 28	1.111
100	7119	4264	40.1	38	1, 2, 3, 5, 6, 8, 10, 11, 16, 18, 21, 23, 26, 32, 33, 41, 42, 49, 50, 54, 55, 62, 63, 64, 66, 68, 74, 75, 76, 77, 78, 80, 81, 82, 87, 90, 91, 96	3.492
150	14104	8316	41.0	56	3, 7, 8, 10, 18, 22, 23, 25, 31, 32, 34, 35, 39, 44, 45, 46, 47, 58, 63, 64, 65, 66, 70, 71, 72, 73, 74, 78, 79, 82, 83, 85, 86, 88, 91, 97, 98, 99, 102, 105, 113, 114, 118, 119, 121, 124, 134, 135, 137, 139, 141, 142, 143, 144, 146, 147	5.086

5 Blocks



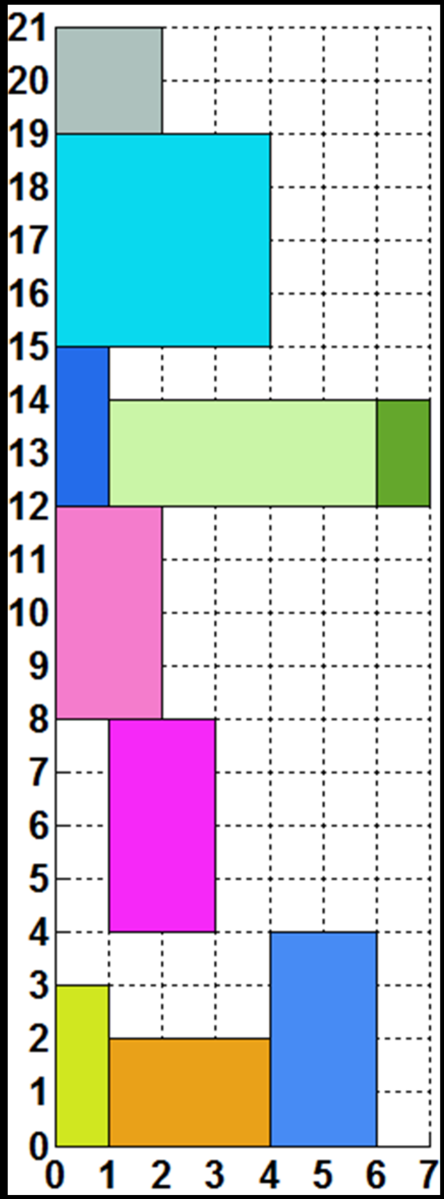
Before rotation,
area = 65

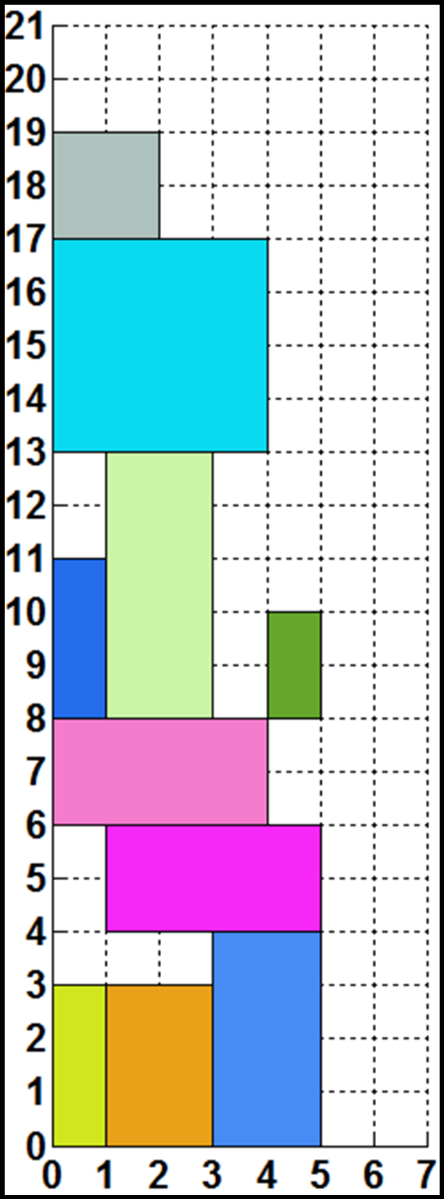


After rotation,
area = 65

10 Blocks

Before rotation, area = 147

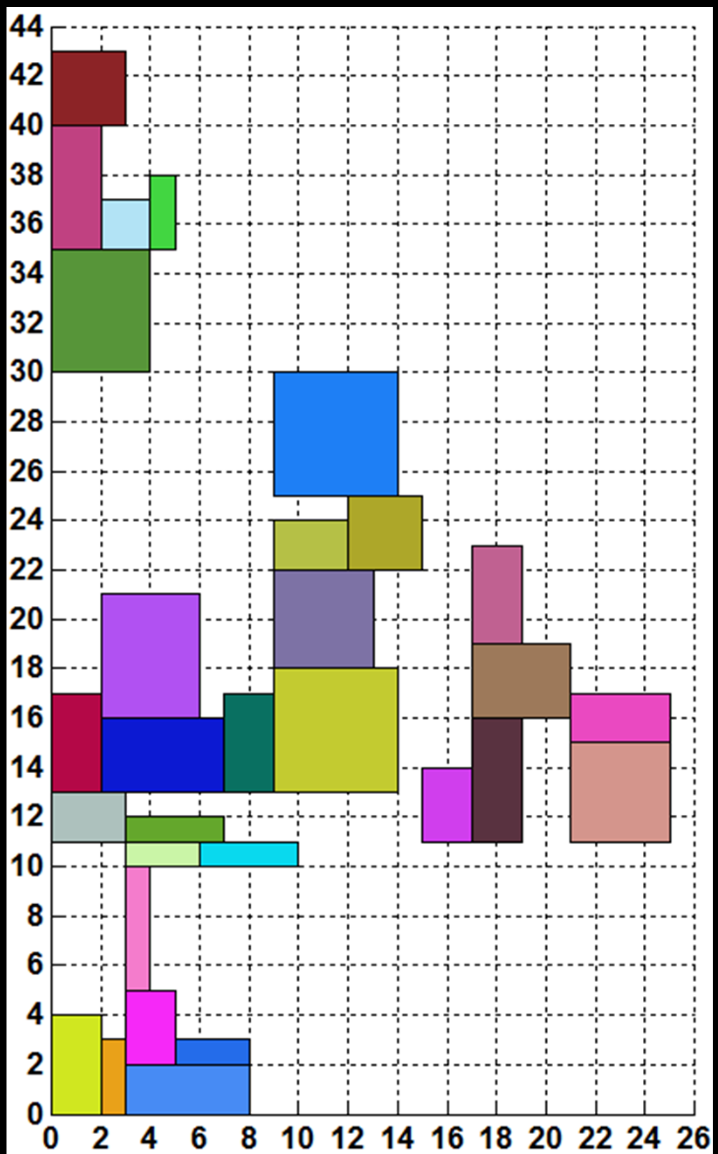




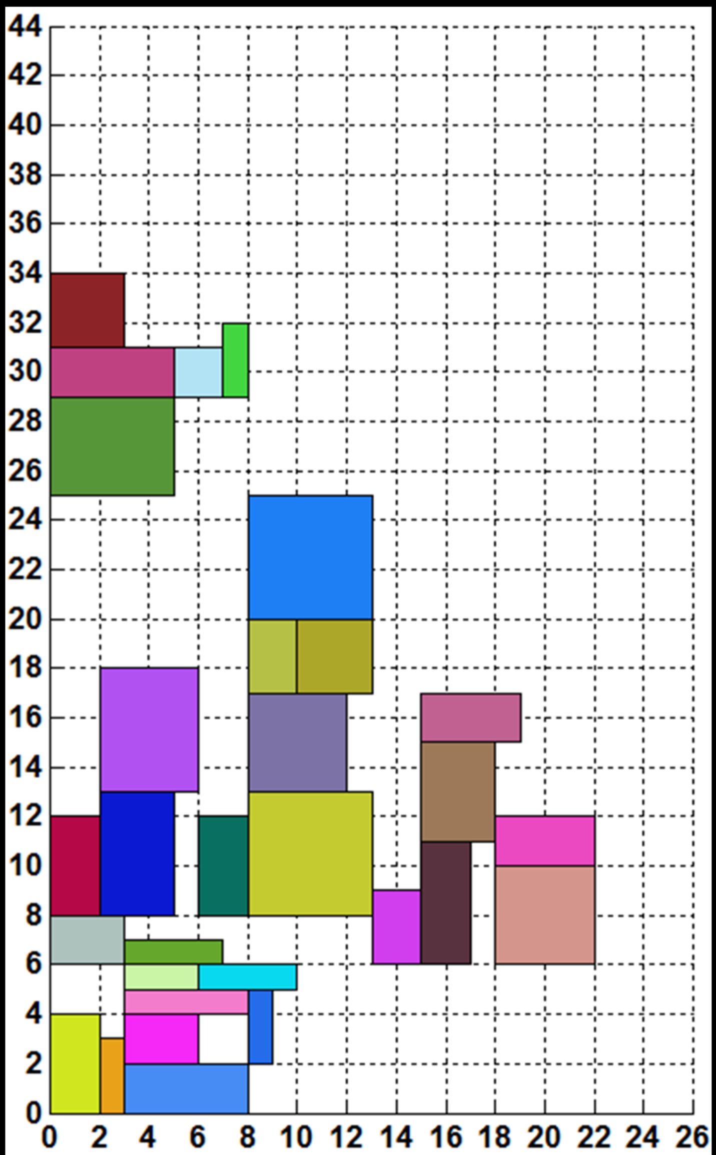
After rotation, area = 95

30 Blocks

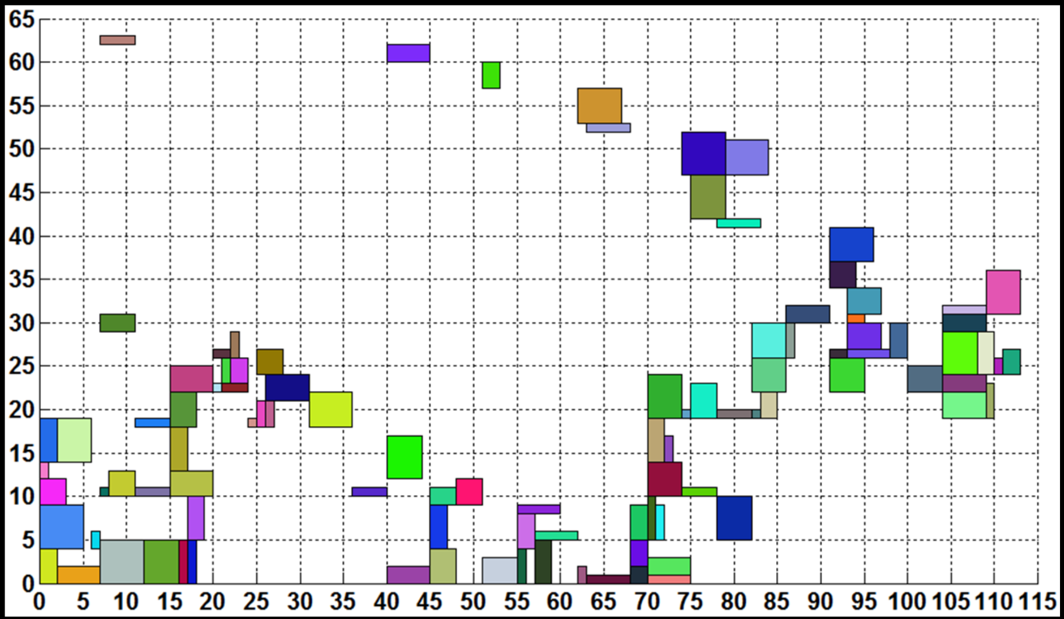
Before rotation, area = 1075



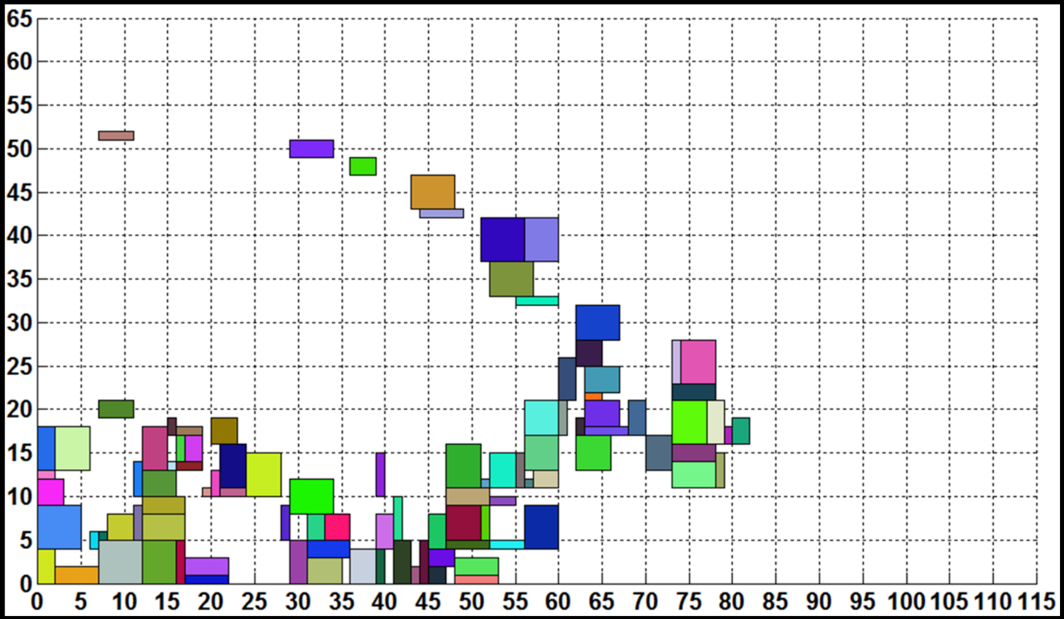
After rotation, area = 748



100 Blocks

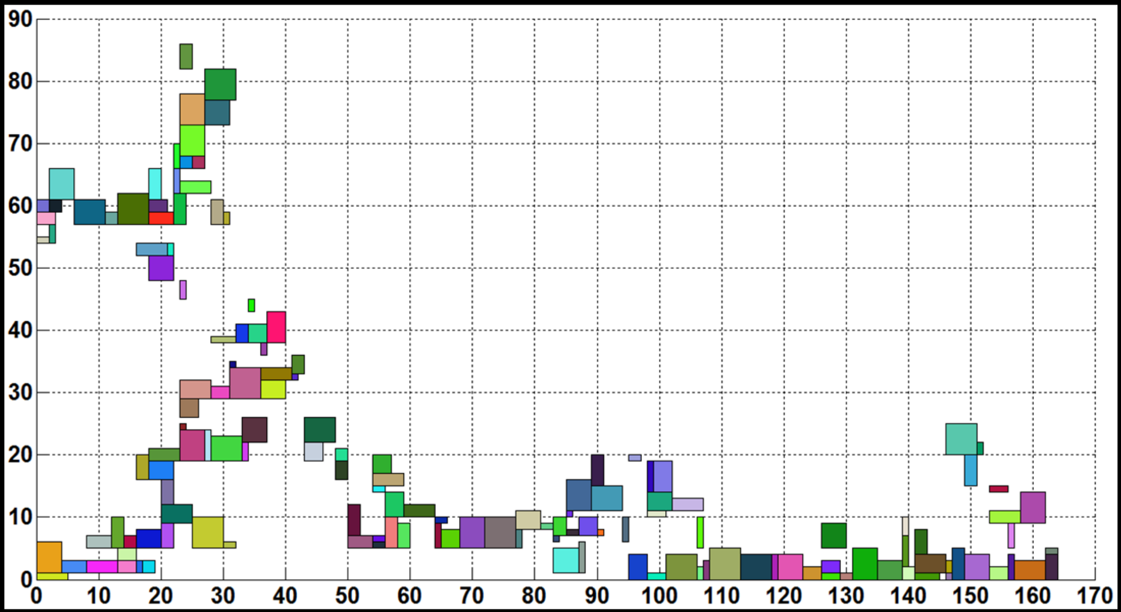


Before rotation, area = 7119

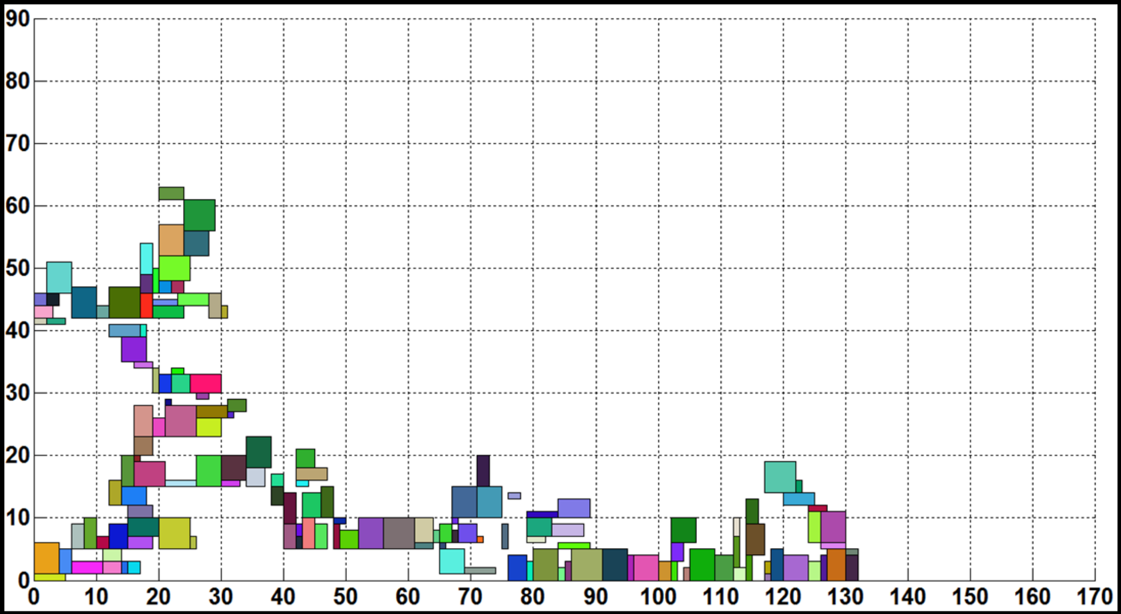


After rotation, area = 4264

150 Blocks



Before rotation, area = 14104



After rotation, area = 8316

Conclusion

- Optimal solution with polynomial runtime
  - Total number of candidates for any internal node is O(L + R) instead of O(L×R), where L and R are the number of candidates in the left and right child
- Reduces area most of the time
- Does not take into account wire routing
  - What if a block only has ports on one side?