CAD Algorithms for Physical Design - Floorplanning

Christos P Sotiriou

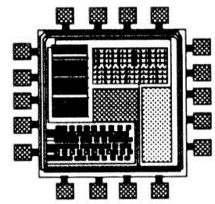
Contents

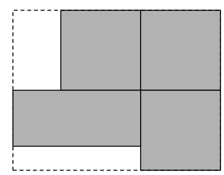
- Floorplanning
- Stockmeyer's Algorithm
- Normalized Reverse Polish
- Sequence Pair

Hierarchical Design

- Several blocks after partitioning:
- Need to:
  - Put the blocks together.
  - Design each block.

  Which step to go first?
- How to put the blocks together without knowing their shapes and the positions of the I/O pins?
- If we design the blocks first, those blocks may not be able to form a tight packing.





Floorplanning

- The floorplanning problem is to plan (i) positions and (ii) module shapes, at the beginning of the design cycle, to optimize circuit parameters:
  - chip area
  - total wirelength
  - delay of critical path
  - routability
  - others, e.g., noise, heat dissipation, etc.

Floorplanning v.s. Placement

- Both determines block positions to optimize the circuit performance.
- Floorplanning:
  - Details like shapes of blocks, I/O pin positions, etc. are not yet fixed (blocks with flexible shape are called soft blocks).
- Placement:
  - Details like module shapes and I/O pin positions are fixed (blocks with no flexibility in shape are called hard blocks).

Floorplanning Problem

- Input:
  - *n* Blocks with areas $A_1, \dots, A_n$
  - Bounds r_i and s_i on the aspect ratio of block B_i
- Output:
  - Coordinates (x_i, y_i), width w_i and height h_i for each block such that $h_i w_i = A_i$ and $r_i \leq h_i/w_i \leq s_i$
- Objective:
  - To optimize circuit parameters
    - WL, Core Area, Timing, *etc.*

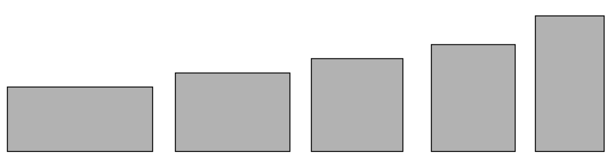
Aspect Ratio Bounds

- If there is no bound on the aspect ratios, can we pack everything tightly?
  - \- Sure!

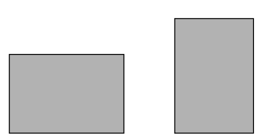
- But we don't want to layout blocks as long strips, so we require $r_i \leq h_i/w_i \leq s_i$ for each i

Bounds on Aspect Ratios

- We may also allow several shapes for each block:



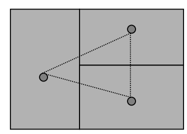
- For hard blocks, the orientation may be changed:

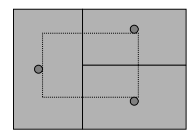


Objective Function and WL Estimation

- A commonly used objective function is a weighted sum of area and wirelength:
  - **cost = α.A + β.WL**,
  - where A is the total area of the packing, WL is the total wirelength, and α and β are constants.

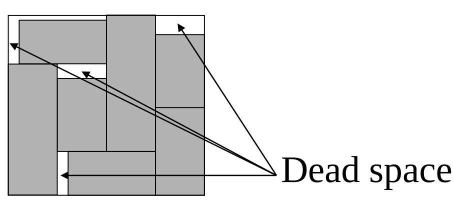
- Exact WL value is not known until routing is done
- During floorplanning, even pin positions are unknown
- Some possible wirelength estimations:
  - Center-to-center estimation
  - Half-perimeter estimation





Dead Space

- Dead space is the space that is wasted:

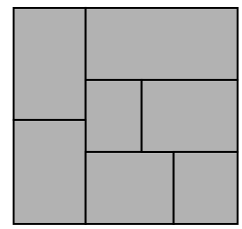


- Minimizing area is the same as minimizing deadspace.
- Dead space percentage is computed as

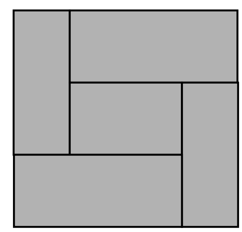
$$(A - \Sigma_i A_i) / A \times 100\%$$

Slicing and Non-Slicing Floorplans

- Slicing Floorplan:
  A Floorplan that may be obtained by repetitively subdividing (slicing) rectangles, horizontally or vertically.



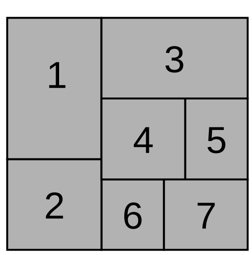
- Non-Slicing Floorplan:
  A Floorplan that may NOT be obtained by repetitively subdividing alone



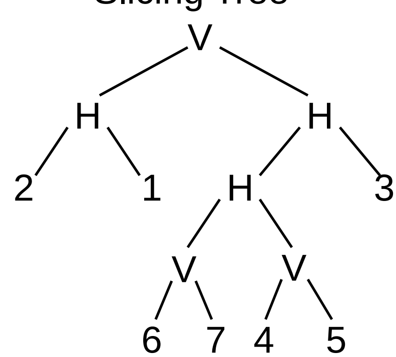
- Otten (LSSS-82) first pointed out that slicing floorplans are much easier to handle

Slicing Floorplan Representation

Slicing Floorplan

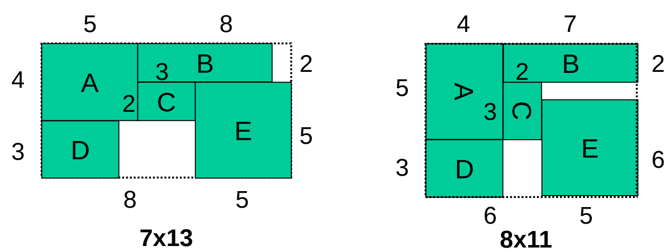


Slicing Tree

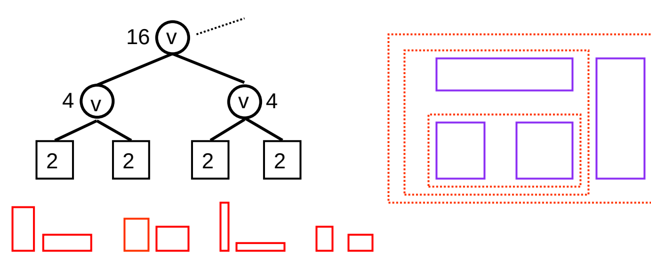


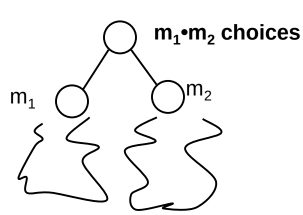
Optimal Hard Macro Floorplan

- Given slicing structure and a set of module shapes (or shape list)
- How to orient these modules such that the total area is smallest?
  - Optimal Orientation of Cells in Slicing floorplan Designs" L. Stockmeyer, Information and Control 57(1983), 91-101



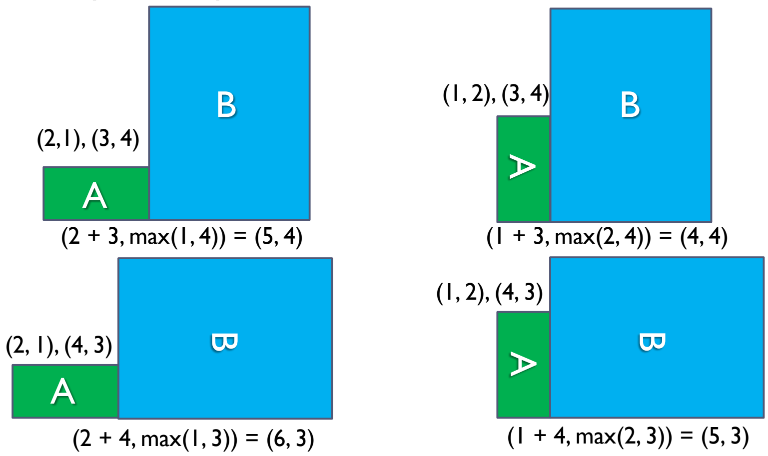
Optimal Hard Macro Floorplan – Choices





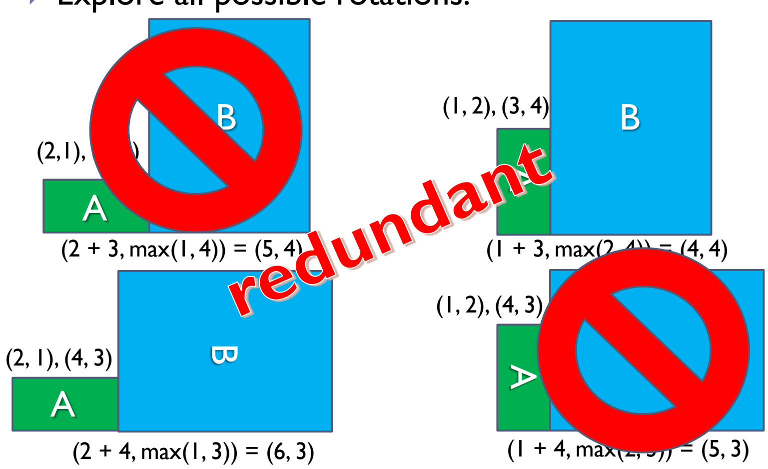
Example of Block Merging

- Block A = (2, 1), Block B = (3, 4)
- Explore all possible rotations:



Example of Block Merging

- Block A = (2, 1), Block B = (3, 4)
- Explore all possible rotations:



Stockmeyer Algorithm

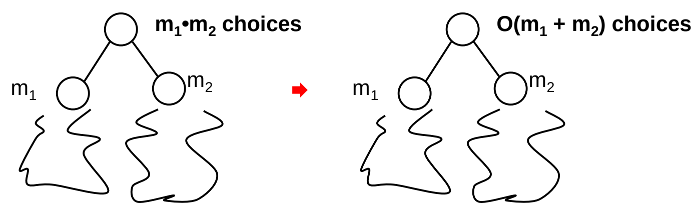
- Sort LHS, RHS children dimensions in
  - **increasing width**, **decreasing height**
- Block A = (2, 1), Block B = (3, 4) becomes
  - L = {(1, 2), (2, 1)}, R = {(3, 4), (4, 3)}
- **First pair, (l1, r1) represents lowest w, largest h**:
  - (l1, r1): (1 + 3, max(2, 4)) = (4, 4)
- *Identify maximum dimension and rotate*: r1 ➔ r2
  - (l1, r2): (1 + 4, max(2, 3)) = (5, 3)
- *Identify maximum dimension*: no new permutation
- **Terminate if maximum dimension cannot be reduced by a new permutation**

Stockmeyer Algorithm Idea in more detail

- Dynamic programming
  - Compute a set of irredundant solutions at each sub-tree
    - rooted from the list of irredundant solutions for its two child subtrees
  - Pick the best solution from the list of irredundant solutions at the root
- For a V node
  - Sort LHS, RHS children dimensions in **increasing width**, **decreasing height**
    - First Resulting dimension is (`wl` + `wr`, `hnew` = `max`(`hl`, `hr`)), for sorted L, R pairs
    - **Repeat:** *identify maximum dimension and rotate:*
      - `if` (`hl` > `hr`), join **next L** pair (maximum from L)
      - `if` (`hl` < `hr`), join **next R** pair (maximum from R)
      - `if` (`hl` = `hr`), join both **next L and next R** pairs (identical maximum)
    - **Until relevant next pairs do not exist**
- For an H node
  - Identical Algorithm, but sort LHS, RHS children dimensions in **decreasing width**, **increasing height**
- **Terminate if maximum dimension cannot be reduced by a new permutation**

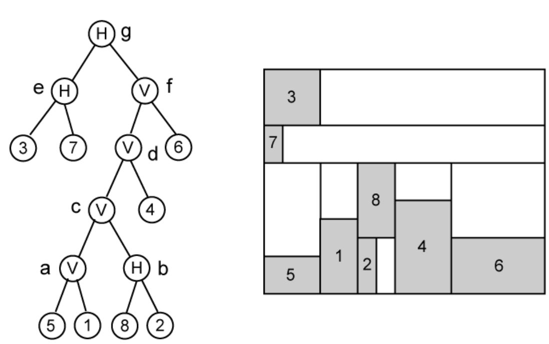
Stockmeyer Algorithm Idea

- Complexity is **O(L + R), instead of O(L x R)** based on Stockmeyer algorithm
  - Does not enumerate the L x R Cartesian Product



Stockmeyer Algorithm Example

- Determine optimal orientation of the blocks
  - Internal nodes in the slicing tree: top-H-bottom, left-V-right
  - lower-left corner of the block → lower-left corner of its room
  - Block dimension: (2,4), (1,3), (3,3), (3,5), (3,2), (5,3), (1,2), (2,4)



Stockmeyer Algorithm Example

- Bottom-up Tree Traversal

visit node a: Since the cut orientation is vertical;

$$L = \{(2,3),(3,2)\}$$
$$R = \{(2,4),(4,2)\}$$

i join $l_1 = (2,3)$ and $r_1 = (2,4)$: we get $(2+2, \max\{3,4\}) = (4,4)$. Since the maximum is from R, we join l_1 and r_2 next.

ii join $l_1 = (2,3)$ and $r_2 = (4,2)$: we get $(2+4, \max\{3,2\}) = (6,3)$. Since the maximum is from L, we join l_2 and r_2 next.

iii join $l_2 = (3,2)$ and $r_2 = (4,2)$: we get $(3+4, \max\{2,2\}) = (7,2)$.

Thus, the resulting dimensions are $\{(4,4),(6,3),(7,2)\}$.

Stockmeyer Algorithm Example

- Bottom-up Tree Traversal

visit node b: Since the cut orientation is horizontal;

$$L = \{(4,2),(2,4)\}$$
$$R = \{(3,1),(1,3)\}$$

i join $l_1 = (4,2)$ and $r_1 = (3,1)$: we get $(\max\{4,3\}, 2+1) = (4,3)$. Since the maximum is from L, we join l_2 and r_1 next.

ii join $l_2 = (2,4)$ and $r_1 = (3,1)$: we get $(\max\{2,3\}, 4+1) = (3,5)$. Since the maximum is from R, we join l_2 and r_2 next.

iii join $l_2 = (2,4)$ and $r_2 = (1,3)$: we get $(\max\{2,1\}, 4+3) = (2,7)$.

Thus, the resulting dimensions are $\{(4,3),(3,5),(2,7)\}$.

Stockmeyer Algorithm Example

- Top Node

visit node g: Since the cut orientation is horizontal;

$$L = \{(3,4)\}$$
$$R = \{(20,3),(18,4),(13,5),(12,7)\}$$

i join $l_1 = (3,4)$ and $r_1 = (20,3)$: we get $(\max\{3,20\}, 4+3) = (20,7)$. Since the maximum is from R, we join l_1 and r_2 next.

ii join $l_1 = (3,4)$ and $r_2 = (18,4)$: we get $(\max\{3,18\}, 4+4) = (18,8)$. Since the maximum is from R, we join l_1 and r_3 next.

iii join $l_1 = (3,4)$ and $r_3 = (13,5)$: we get $(\max\{3,13\}, 4+5) = (13,9)$. Since the maximum is from R, we join l_1 and r_4 next.

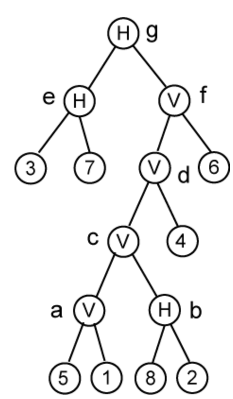
iv join $l_1 = (3,4)$ and $r_4 = (12,7)$: we get $(\max\{3,12\}, 4+7) = (12,11)$.

Thus, the resulting dimensions are $\{(20,7),(18,8),(13,9),(12,11)\}$. The minimum area floorplan is $13 \times 9 = 117$.

Stockmeyer Algorithm Example

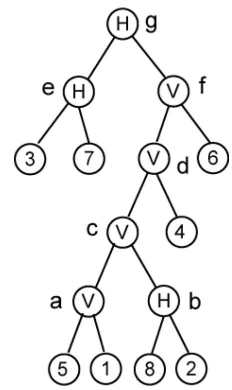
- Top-Node Tree Traversal

node	dir	dimensions
g	hor	$L = \{(\mathbf{3},\mathbf{4})\}$
		$R = \{(20,3),(18,4),(\mathbf{13},\mathbf{5}),(12,7)\}$
		$D = \{(20,7),(18,8),(\mathbf{13},\mathbf{9}),(12,11)\}$
e	hor	$L = \{(\mathbf{3},\mathbf{3})\}$
		$R = \{(\mathbf{2},\mathbf{1}),(1,2)\}$
		$D = \{(\mathbf{3},\mathbf{4})\}$
f	ver	$L = \{(9,7),(\mathbf{10},\mathbf{5}),(13,4),(15,3)\}$
		$R = \{(\mathbf{3},\mathbf{5}),(5,3)\}$
		$D = \{(12,7),(\mathbf{13},\mathbf{5}),(18,4),(20,3)\}$
d	ver	$L = \{(6,7),(\mathbf{7},\mathbf{5}),(8,4),(10,3)\}$
		$R = \{(\mathbf{3},\mathbf{5}),(5,3)\}$
		$D = \{(9,7),(\mathbf{10},\mathbf{5}),(13,4),(15,3)\}$



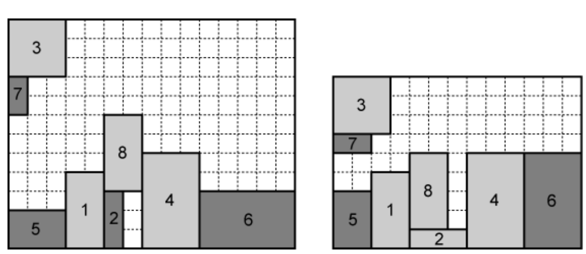
Stockmeyer Algorithm Example

node	dir	dimensions
c	ver	$L = \{(\mathbf{4}, \mathbf{4}), (6, 3), (7, 2)\}$
		$R = \{(2, 7), (\mathbf{3}, \mathbf{5}), (4, 3)\}$
		$D = \{(6, 7), (\mathbf{7}, \mathbf{5}), (8, 4), (10, 3)\}$
b	hor	$L = \{(4, 2), (\mathbf{2}, \mathbf{4})\}$
		$R = \{(\mathbf{3}, \mathbf{1}), (1, 3)\}$
		$D = \{(4, 3), (\mathbf{3}, \mathbf{5}), (2, 7)\}$
a	ver	$L = \{(\mathbf{2}, \mathbf{3}), (3, 2)\}$
		$R = \{(\mathbf{2}, \mathbf{4}), (4, 2)\}$
		$D = \{(\mathbf{4}, \mathbf{4}), (6, 3), (7, 2)\}$



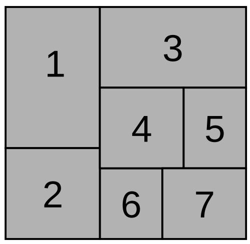
Stockmeyer Algorithm Example

- Final Floorplan
- 4 blocks are rotated
  - Area reduced from 15 × 12 to 13 × 9

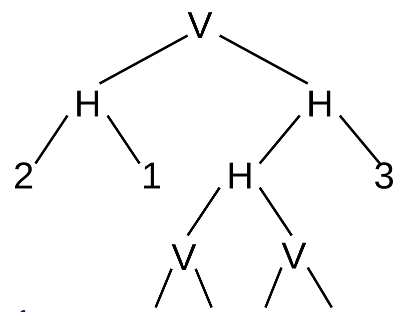


Slicing Floorplan Representation

Slicing Floorplan

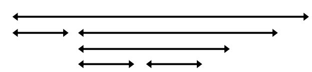


Slicing Tree



- D.F. Wong and C.L. Liu, "A New Algorithm for Floorplan Design" DAC, 1986, pages 101-107.

Polish Expression
(postorder traversal
of slicing tree)



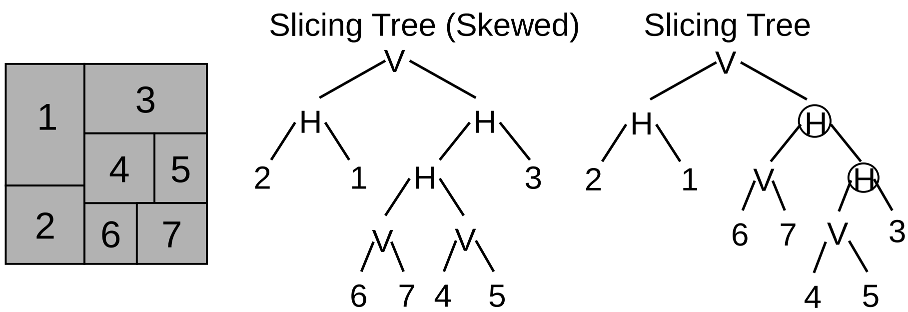
21H67V45VH3HV

RP Floorplan Representation

- Succinct representation of slicing floorplan
  - roughly specifying relative positions of blocks
- Postorder traversal of slicing tree
  - 1. Postorder traversal of left sub-tree
  - 2. Postorder traversal of right sub-tree
  - 3. The label of the current root
- For n blocks, a Polish Expression contains n operands (blocks) and (n – 1) operators (H,V).
- However, for a given slicing floorplan, the corresponding slicing tree (and hence polish expression) is not unique
  - Therefore, there is some redundancy in the representation.

Skewed Slicing Tree and Normalized RP

- Skewed Slicing Tree:
  - **no node and its right son are the same**
- Normalized Polish Expression:
  - **no consecutive H's or V's**



Polish Expression 21H67V45VH3HV 21H67V45V3HHV

Normalized RP

- There is a 1-1 correspondence between Slicing Floorplan, Skewed Slicing Tree, and Normalized Polish Expression
- Normalized Polish Expression is typically used to represent slicing floorplans, or its Binary Tree
  - What is a valid NPE?
- Can be formulated as a state space search problem
  - Chain: HVHVH.... or VHVHV....

16H35V2HV74HV

Chains

RP Neighborhood Structure

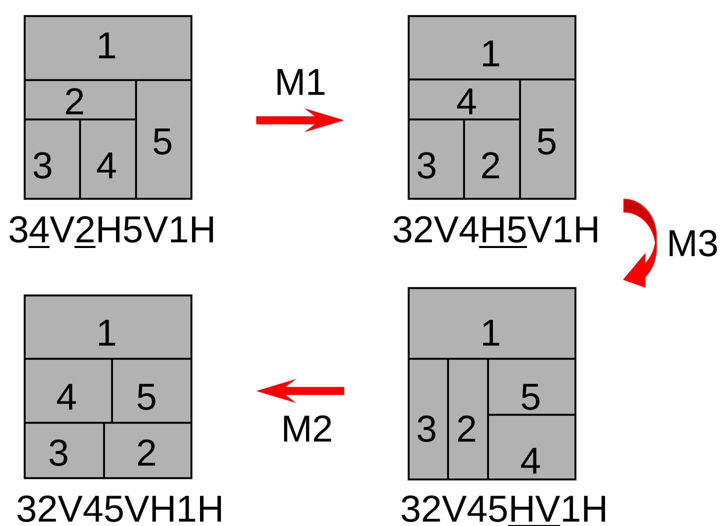
- Chain: HVHVH.... or VHVHV....

16H35V2HV74HV

Chains

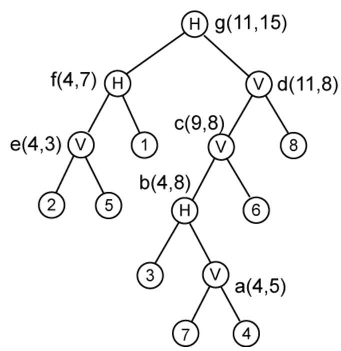
- The moves:
  - **M1: Swap adjacent operands** (ignoring chains)
  - **M2: Complement a chain**
  - **M3: Swap an adjacent operand and an operator**
    - M3 may produce an invalid NPE, thus checking NPE validity is necessary
- It can be proven that every pair of valid NPE are connected through these moves

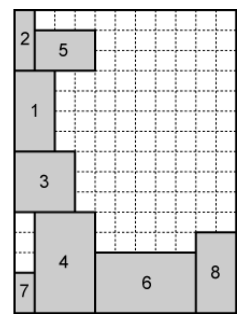
Example of NPE Moves



Normalized Polish Expression Example

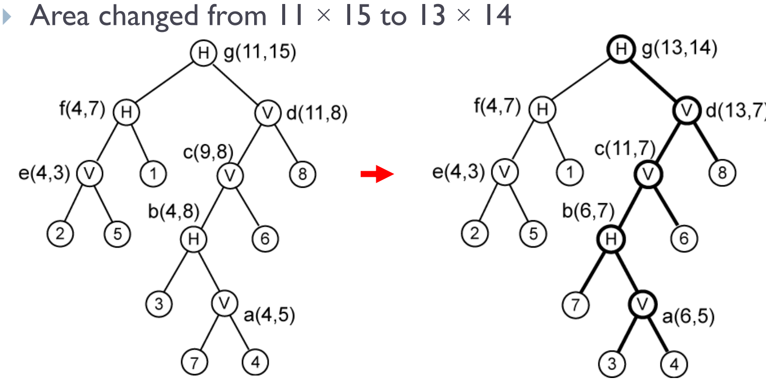
- Draw slicing floorplan based on:
  - Initial PE: P_1 = 25V1H374VH6V8VH
  - Dimensions: (2,4), (1,3), (3,3), (3,5), (3,2), (5,3), (1,2), (2,4)





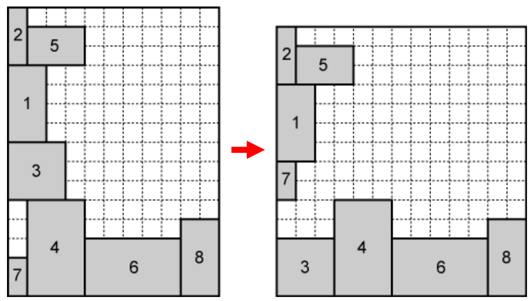
Normalized Polish Expression Example

- M1 Move
- Swap module 3 and 7 in P_1 = 25V1H<u>37</u>4VH6V8VH
  - We get: P_2 = 25V1H<u>73</u>4VH6V8VH
  - Area changed from 11 × 15 to 13 × 14



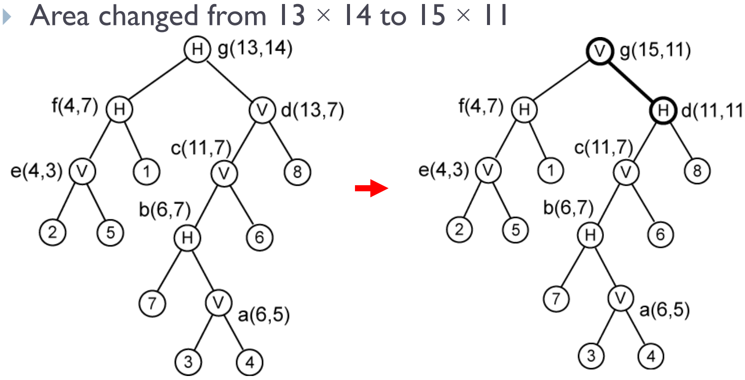
Normalized Polish Expression Example

- Floorplan Change



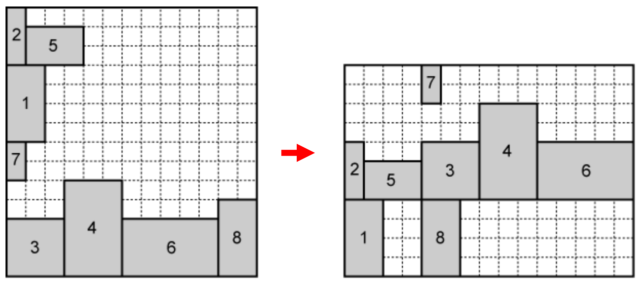
Normalized Polish Expression Example

- M2 Move
- Complement last chain in P_2 = 25V1H734VH6V8VH
  - We get: P_3 = 25V1H734VH6V8HV
  - Area changed from 13 × 14 to 15 × 11



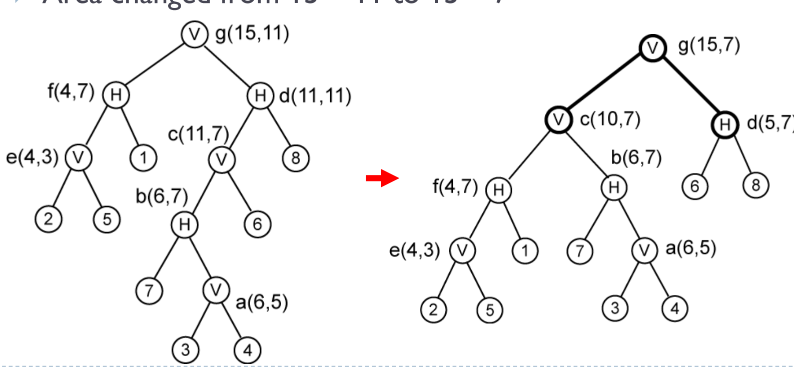
Normalized Polish Expression Example

- Floorplan Change



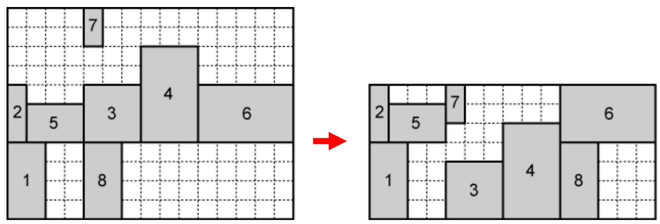
Normalized Polish Expression Example

- M3 Move
- Swaps 6 and V in P_3 = 25V1H734VH6V8HV
  - We get: P_4 = 25V1H734VHV68HV
  - Area changed from 15 × 11 to 15 × 7



Normalized Polish Expression Example

- Floorplan Change



Normalized Polish Expression Example

- What is average change on cost function?
- Initial temperature with acceptance probability 0.9?

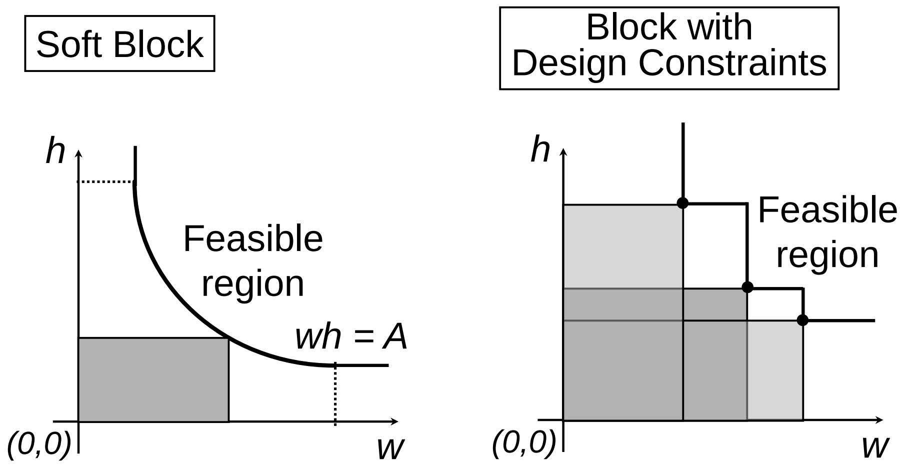
The area changed from 11×15 to 13×14 to 15×11 to 15×7. Thus, the average area change is

$$\Delta_{ave} = \frac{|165 - 182| + |182 - 165| + |165 - 105|}{3} = 31.33$$

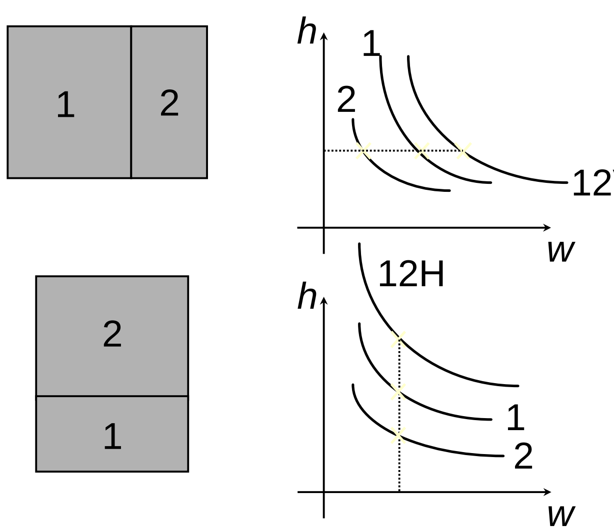
Thus,

$$T_0 = \frac{-\Delta_{ave}}{ln(0.9)} = 297.39$$

Floorplan Block Shape Curves

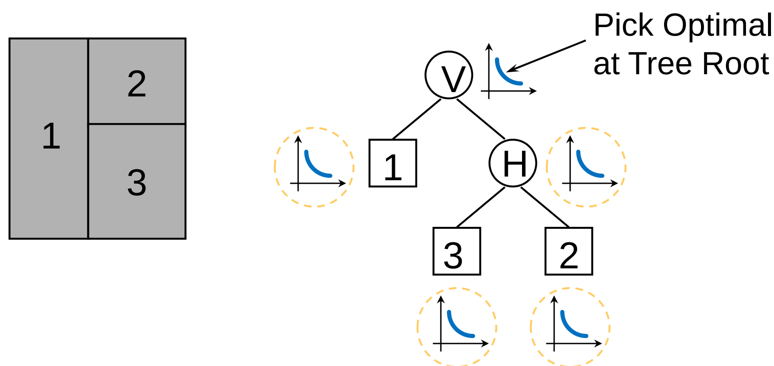


Combining Shape Curves



Identifying the Optimal Area for an NPE

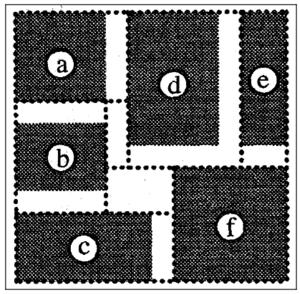
- Recursively combine shape curves

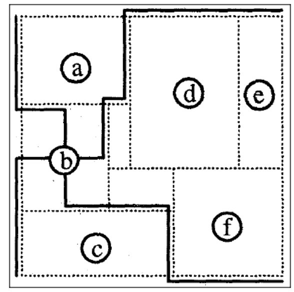


Incremental Shape Curve Update

- If keeping k points for each shape curve, time for shape curve computation for each NPE is O(kn)
- After each move, only a small change occurs to the floorplan:
  - no need to re-compute shape curve computation from scratch
  - Shape curves may be updated incrementally after each Floorplan move
- Reduces Run time to ~ O(k log n)

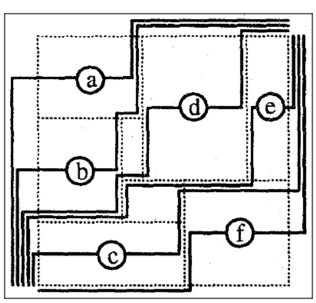
Sequence Pair Representation



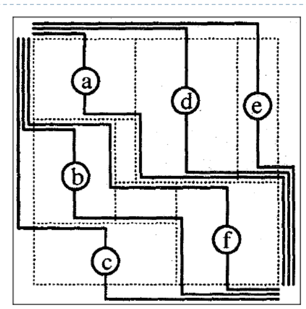


- Centre of each module may be mapped to a locus
  - Right-up and Left-down are defined as + Locus
  - Down-right and Up-left are defined as – Locus
- Loci of b above show +ve and –ve Loci
- Theorem: No Positive/Negative Loci cross each other

Sequence Pair Representation



Positive Loci



Negative Loci

- Sequence Pair Ordering = (abcdef, cbfade)
  - from Top-left, Bottom-right respectively

Sequence Pair Representation

- Sequence Pair Ordering = (abcdef, cbfade) = (G+, G-)
  - For a pair of modules x, x' there are four possible ordering cases in SP
    - (i) x' is after x, in both G+ and G- [Maa]
    - (ii) x' is before x, in both G+ and G- [Mbb]
    - (iii) x' is before x in G+, after x in G- [Mba]
    - (iv) x' is after x in G+, before x in G- [Mab]
- Maa(b) = {d,e,f}, Mbb(b) = 8, Mba(b) = **{*a*}**, and Mab(b) = **{*c*}**
- Theorem
  - Maa → right of, Mbb → left to, Mba → above, Mab → below
  - In terms of the positions in a SP Floorplan

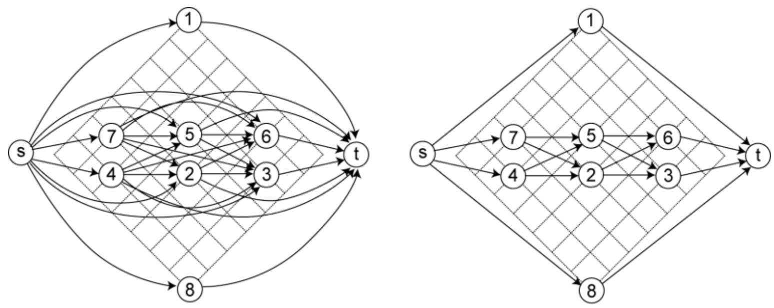
Sequence Pair Example

- Initial SP: SP_1 = (17452638, 84725361)
  - Dimensions: (2,4), (1,3), (3,3), (3,5), (3,2), (5,3), (1,2), (2,4)
  - Based on SP_1 we build the following table:

module	right-of	left-of	above	below
1	$\emptyset$	$\emptyset$	$\emptyset$	$\{2,3,4,5,6,7,8\}$
2	$\{3,6\}$	$\{4,7\}$	$\{1,5\}$	$\{8\}$
3	$\emptyset$	$\{2,4,5,7\}$	$\{1,6\}$	$\{8\}$
4	$\{2,3,5,6\}$	$\emptyset$	$\{1,7\}$	$\{8\}$
5	$\{3,6\}$	$\{4,7\}$	$\{1\}$	$\{2,8\}$
6	$\emptyset$	$\{2,4,5,7\}$	$\{1\}$	$\{3,8\}$
7	$\{2,3,5,6\}$	$\emptyset$	$\{1\}$	$\{4,8\}$
8	$\emptyset$	$\emptyset$	$\{1,2,3,4,5,6,7\}$	$\emptyset$

Sequence Pair Example

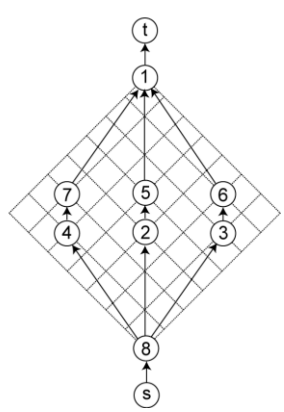
- Horizontal constraint graph (HCG)
  - Before and after removing transitive edges



(a) (b)

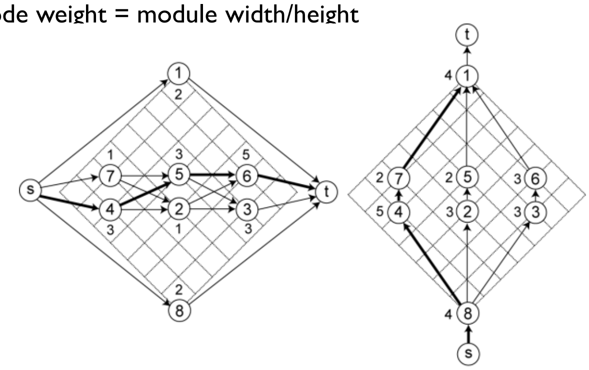
Sequence Pair Example

- Vertical constraint graph (VCG)



Sequence Pair Example

- Computing Chip Width and Height
- Longest source-sink path length in:
  - HCG = chip width, VCG = chip height
  - Node weight = module width/height

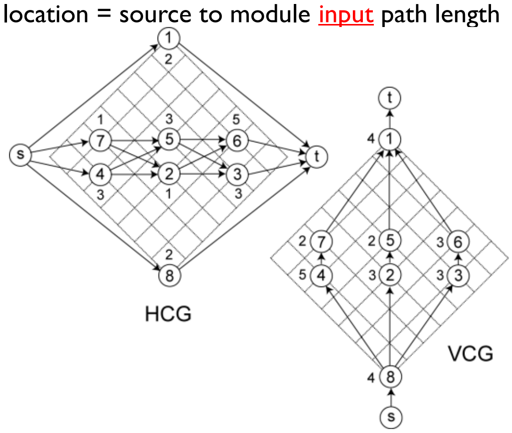


(a) (b)

Sequence Pair Example

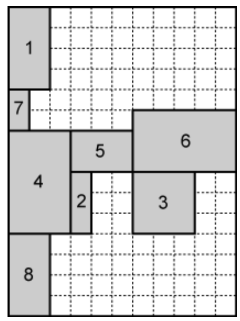
- Computing Module Location
  - Use longest source-module path length in HCG/VCG
    - Lower-left corner location = source to module input path length

module	HCV	VCG
1	0	11
2	3	4
3	6	4
4	0	4
5	3	7
6	6	7
7	0	9
8	0	0



Sequence Pair Example

- Final Floorplan
  - Dimension: 11 × 15



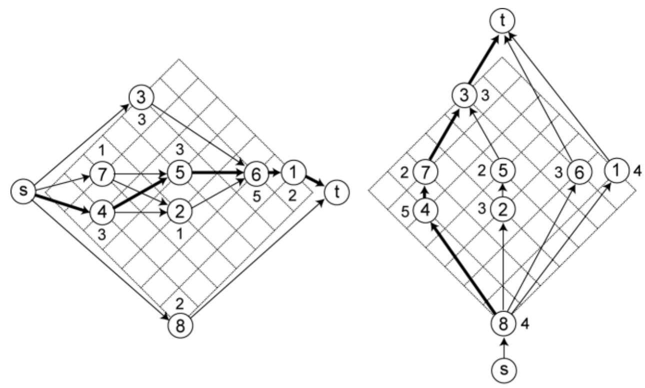
Sequence Pair Example

- Move I
- Swap 1 and 3 in positive sequence of SP_1
  - SP_1 = (17452638, 84725361)
  - SP_2 = (37452618, 84725361)

module	right-of	left-of	above	below
1	$\emptyset$	$\{2,3,4,5,6,7\}$	$\emptyset$	$\{8\}$
2	$\{1,6\}$	$\{4,7\}$	$\{3,5\}$	$\{8\}$
3	$\{1,6\}$	$\emptyset$	$\emptyset$	$\{2,4,5,7,8\}$
4	$\{1,2,5,6\}$	$\emptyset$	$\{3,7\}$	$\{8\}$
5	$\{1,6\}$	$\{4,7\}$	$\{3\}$	$\{2,8\}$
6	$\{1\}$	$\{2,3,4,5,7\}$	$\emptyset$	$\{8\}$
7	$\{1,2,5,6\}$	$\emptyset$	$\{3\}$	$\{4,8\}$
8	$\emptyset$	$\emptyset$	$\{1,2,3,4,5,6,7\}$	$\emptyset$

Sequence Pair Example

- Constraint Graphs

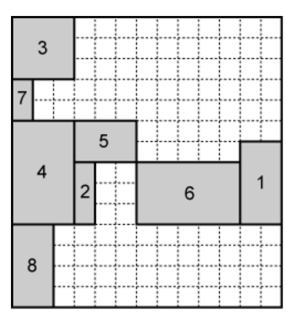


(a) (b)

Sequence Pair Example

- Constructing Floorplan
- Dimension: 13×14

module	HCV	VCG
1	11	4
2	3	4
3	0	11
4	0	4
5	3	7
6	6	4
7	0	9
8	0	0



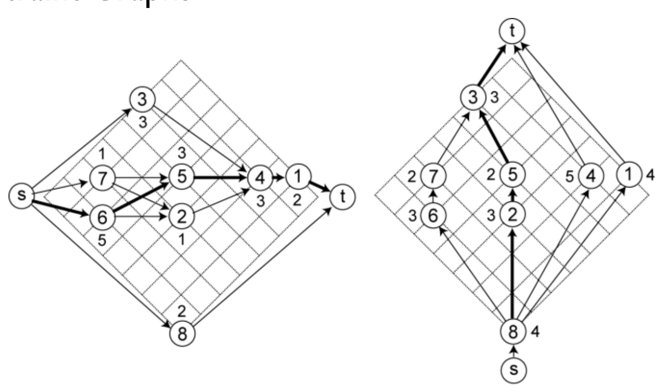
Sequence Pair Example

- Move II
- Swap 4 and 6 in both sequences of SP_2
  - SP_2 = (37452618, 84725361)
  - SP_3 = (37652418, 86725341)

module	right-of	left-of	above	below
1	∅	{2, 3, 4, 5, 6, 7}	∅	{8}
2	{1, 4}	{6, 7}	{3, 5}	{8}
3	{1, 4}	∅	∅	{2, 5, 6, 7, 8}
4	{1}	{2, 3, 5, 6, 7}	∅	{8}
5	{1, 4}	{6, 7}	{3}	{2, 8}
6	{1, 2, 4, 5}	∅	{3, 7}	{8}
7	{1, 2, 4, 5}	∅	{3}	{6, 8}
8	∅	∅	{1, 2, 3, 4, 5, 6, 7}	∅

Sequence Pair Example

- Constraint Graphs

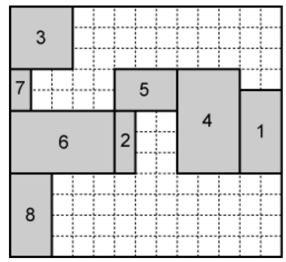


(a) (b)

Sequence Pair Example

- Constructing Floorplan
- Dimension: 13 × 12

module	HCV	VCG
1	11	4
2	3	4
3	0	11
4	0	4
5	3	7
6	6	4
7	0	9
8	0	0



Sequence Pair Example

- Summary
- Moves Impact:
  - Floorplan dimension changes from 11 × 15 to 13 × 14 to 13 × 12