Floorplanning

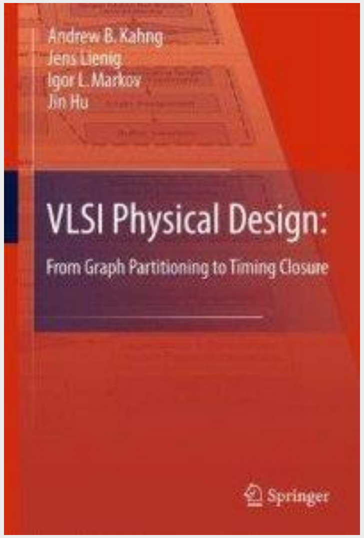
Mustafa Ozdal

SOME SLIDES ARE FROM THE BOOK:

VLSI Physical Design: From Graph Partitioning to Timing Closure

MODIFICATIONS WERE MADE ON THE ORIGINAL SLIDES

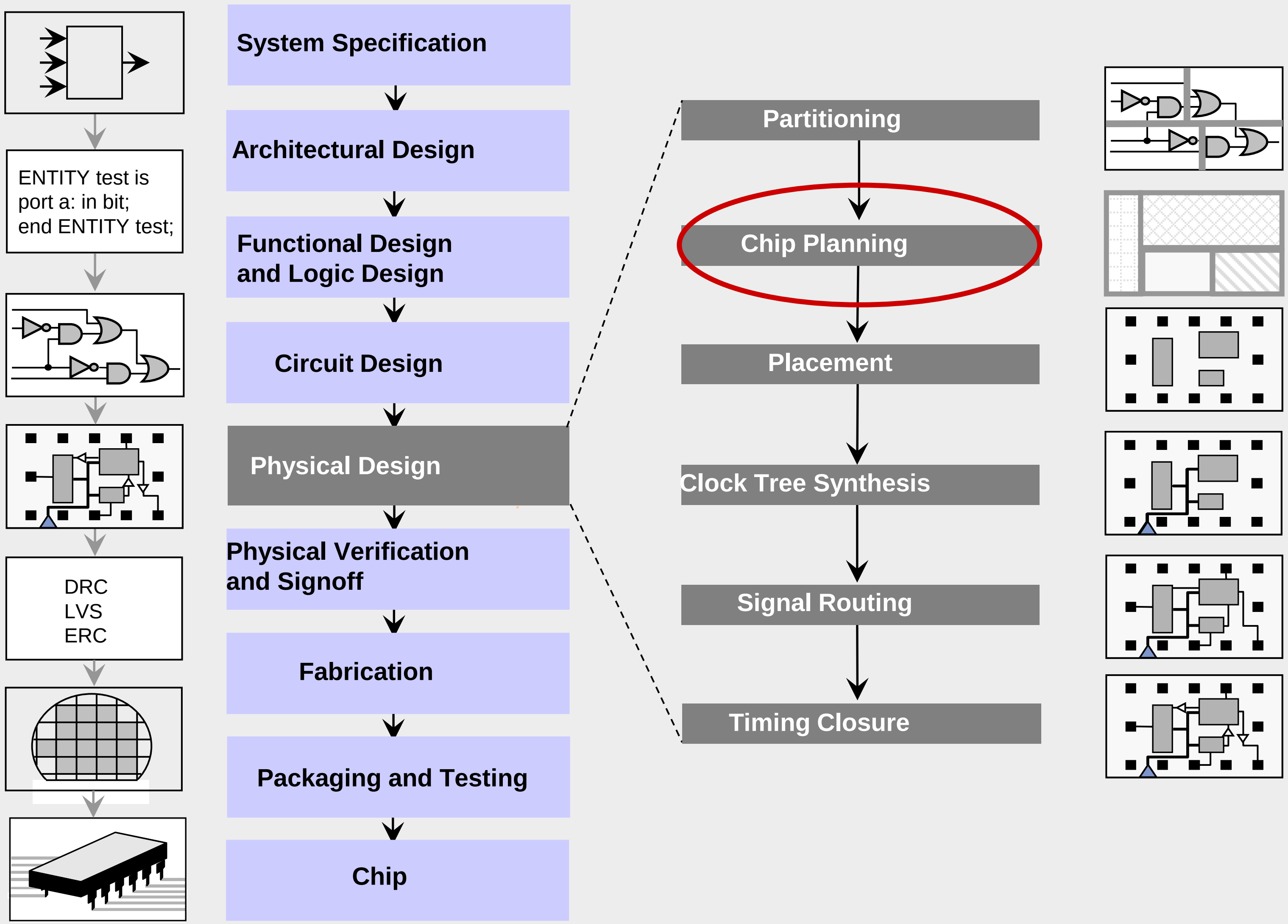
Chapter 2 – Netlist and System Partitioning

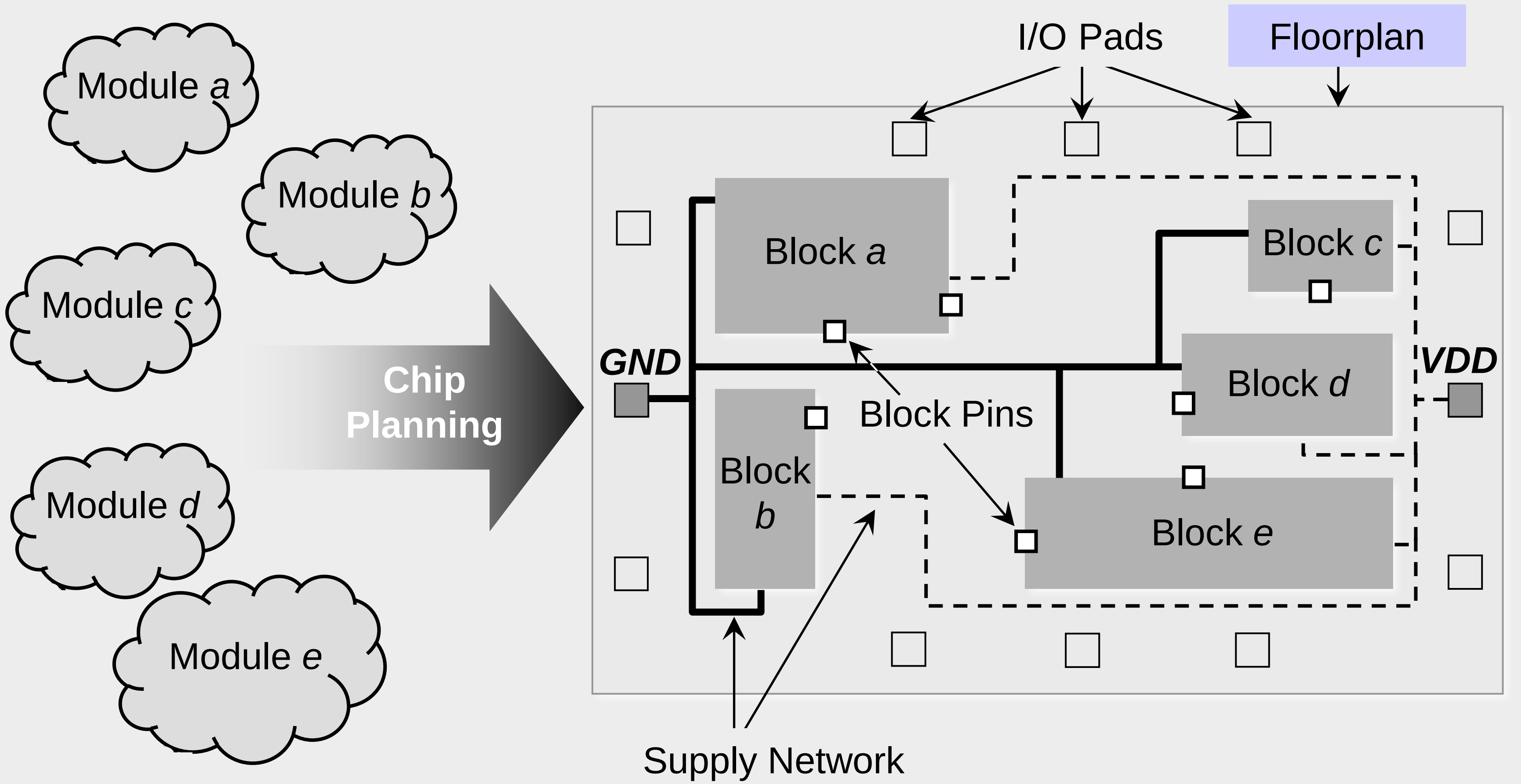


Original Authors:

Andrew B. Kahng, Jens Lienig, Igor L. Markov, Jin Hu

3.1 Introduction




Module a
Module b
Module c
Module d
Module e
Chip Planning
I/O Pads
Floorplan
Block a
Block b
Block c
Block d
Block e
GND
VDD
Block Pins
Supply Network

Floorplanning

- Circuit modules obtained through partitioning
  - either automatic or manual partitioning

- Floorplanning: Assign shapes and locations for all circuit modules.

3.1 Introduction

Example

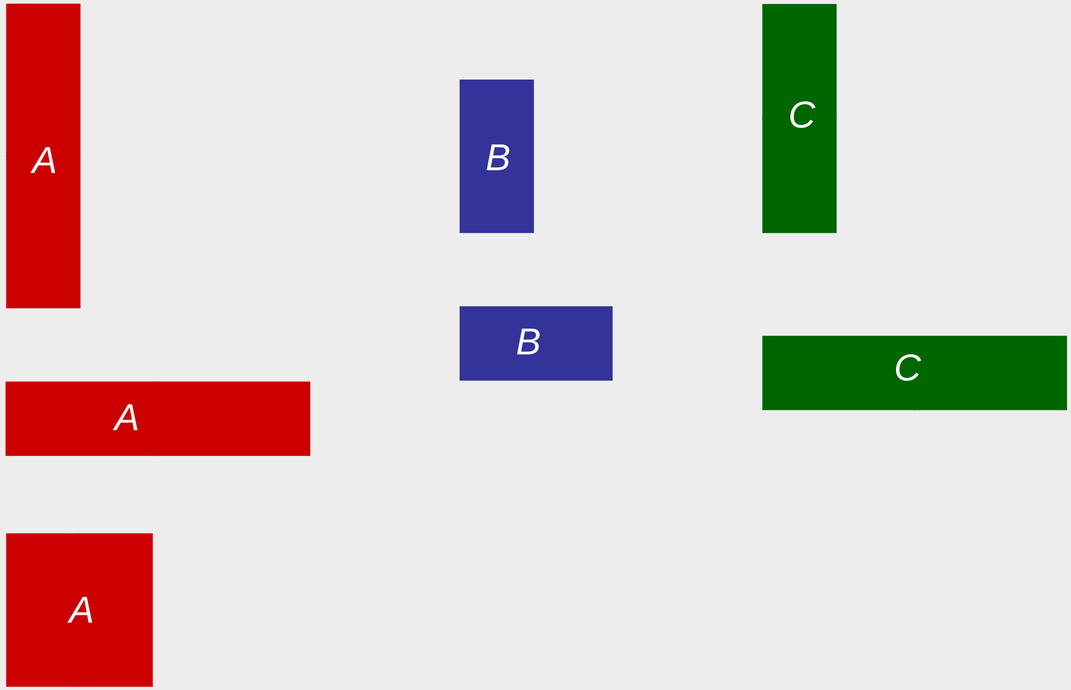
Given: Three blocks with the following potential widths and heights

Block A: w = 1, h = 4 or w = 4, h = 1 or w = 2, h = 2

Block B: w = 1, h = 2 or w = 2, h = 1

Block C: w = 1, h = 3 or w = 3, h = 1

Task: Floorplan with minimum total area enclosed



Example

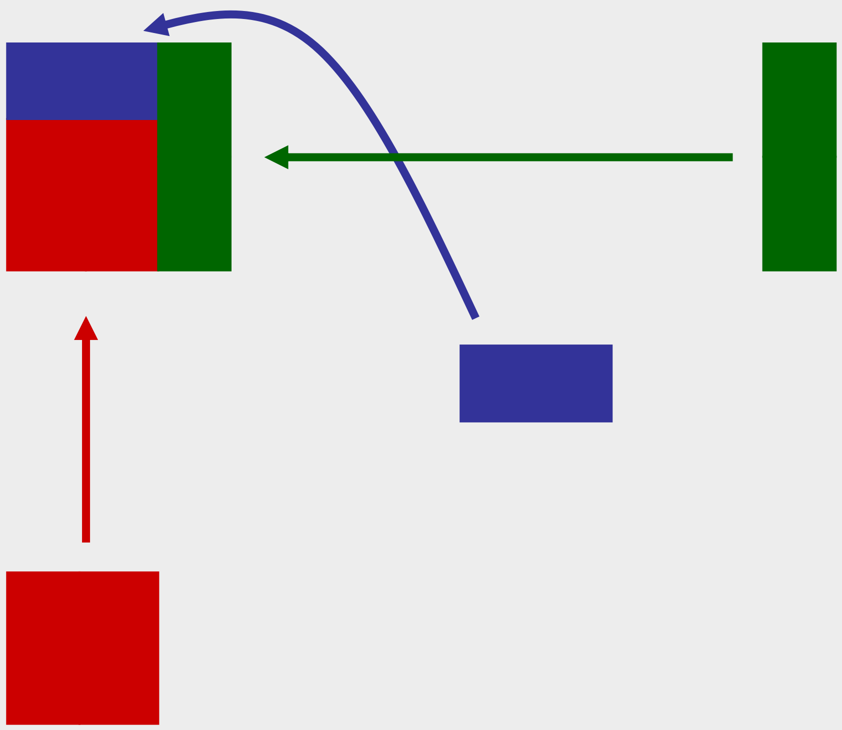
Given: Three blocks with the following potential widths and heights

Block A: $w = 1, h = 4$ or $w = 4, h = 1$ or $w = 2, h = 2$

Block B: $w = 1, h = 2$ or $w = 2, h = 1$

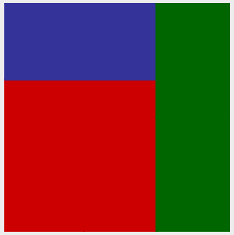
Block C: $w = 1, h = 3$ or $w = 3, h = 1$

Task: Floorplan with minimum total area enclosed



Example
Given: Three blocks with the following potential widths and heights
Block A: $w = 1, h = 4$ or $w = 4, h = 1$ or $w = 2, h = 2$
Block B: $w = 1, h = 2$ or $w = 2, h = 1$
Block C: $w = 1, h = 3$ or $w = 3, h = 1$

Task: Floorplan with minimum total area enclosed



Solution:
Aspect ratios
Block A with $w = 2, h = 2$; Block B with $w = 2, h = 1$; Block C with $w = 1, h = 3$

This floorplan has a global bounding box with minimum possible area (9 square units).

Optimization Objectives

- Minimize the area of the global bounding box
  - Aspect ratio constraints due to packaging and manufacturing limitations (e.g. a square chip)

- Minimize the total wirelength between blocks
  - Long connections increase signal delays (lower performance)
  - More wirelength can degrade routability
  - More wirelength increases power (due to wire capacitances)

Objective Function: Example

- Combination of $area(F)$ and total wirelength $L(F)$ of floorplan F

$$\text{Minimize } \alpha \cdot area(F) + (1 - \alpha) \cdot L(F)$$

where the parameter $0 \leq \alpha \leq 1$ gives the relative importance between $area(F)$ and $L(F)$

Floorplan Representations

- A floorplan can be represented based on the locations of the blocks

  ➔ Complicates generation of new overlap-free floorplans

- Typical floorplanning algorithms are iterative in nature
  - Local search and iterative improvement heavily used

- Topological representations based on relative block positions
  - The represented floorplan guaranteed to be overlap free
  - Easy to evaluate and make incremental changes

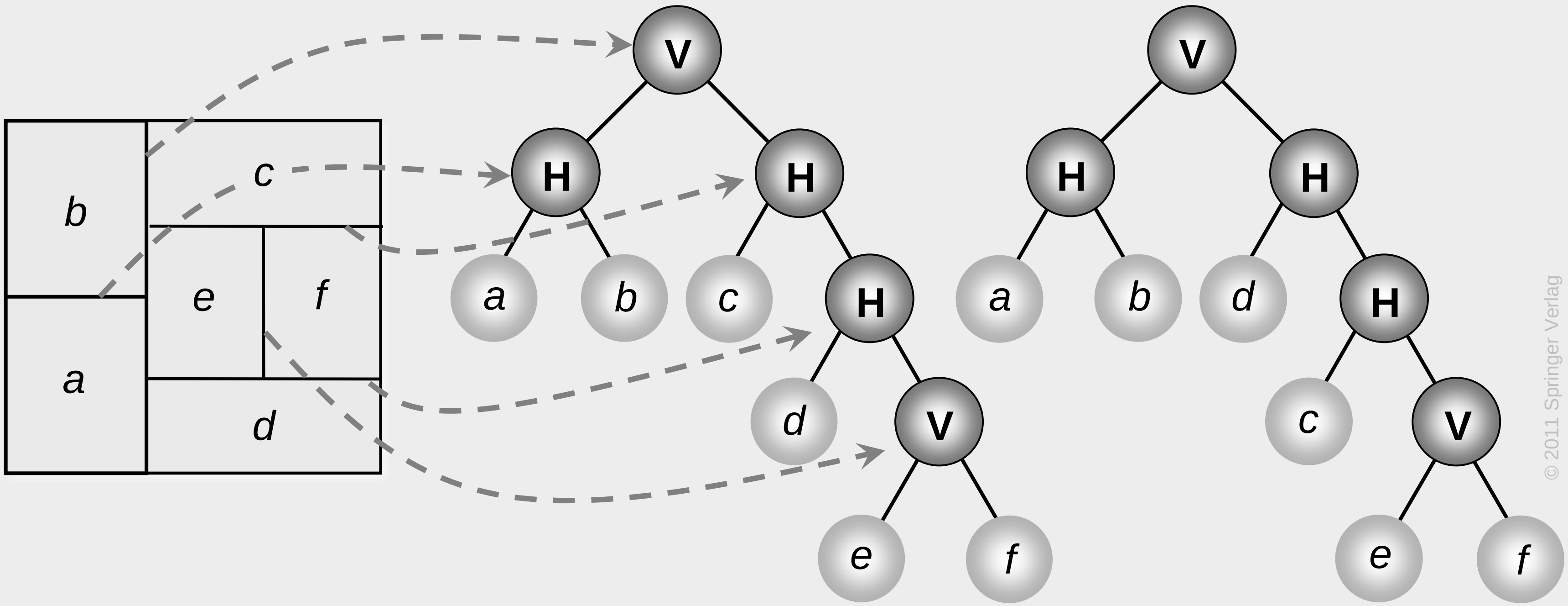
3.3 Terminology

- A rectangular dissection is a division of the chip area into a set of *blocks* or non-overlapping rectangles.

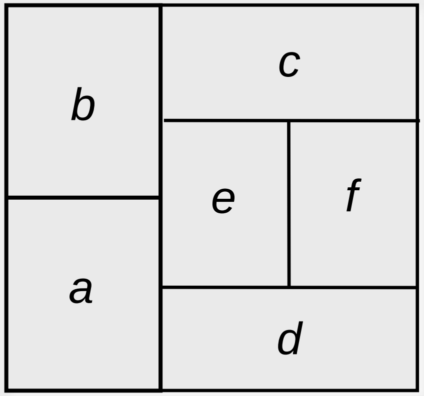
- A slicing floorplan is a rectangular dissection
  - Obtained by repeatedly dividing each rectangle, starting with the entire chip area, into two smaller rectangles
  - Horizontal or vertical cut line.

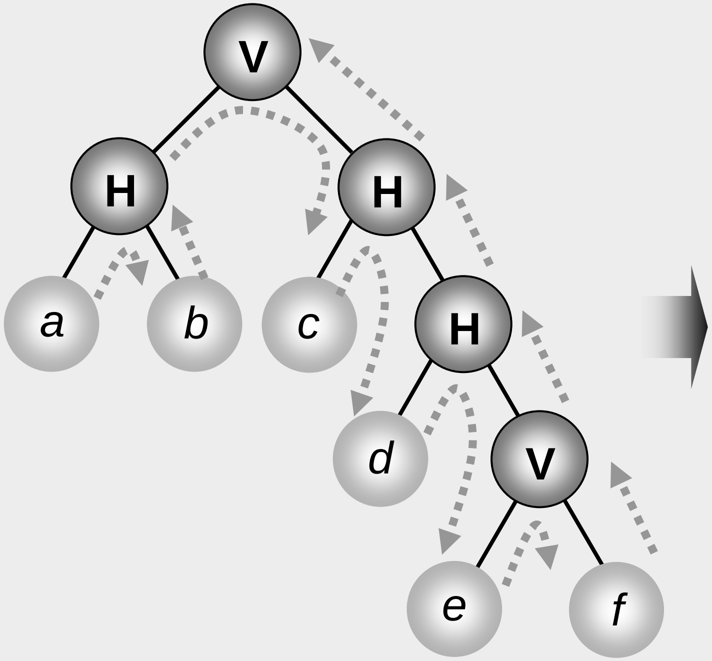
- A slicing tree or slicing floorplan tree is a binary tree with k leaves and k – 1 internal nodes
  - Each leaf represents a block
  - Each internal node represents a horizontal or vertical cut line.

Slicing floorplan and two possible corresponding slicing trees



Polish expression

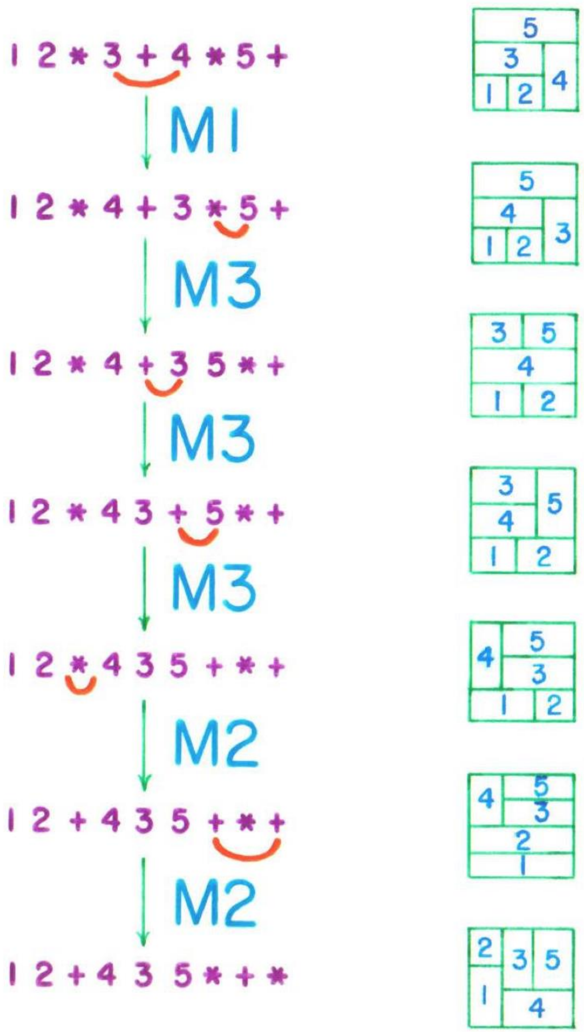




AB+CDEF*++*

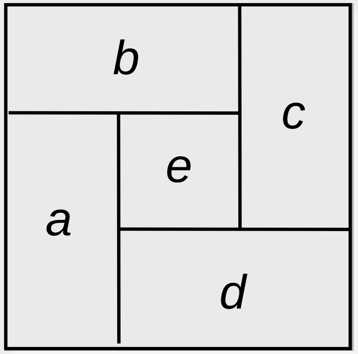
- Bottom up: **V** → * and **H** → +
- Length 2*n*-1 (*n* = Number of leaves of the slicing tree)

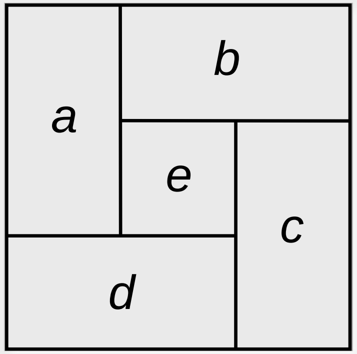
Algorithm



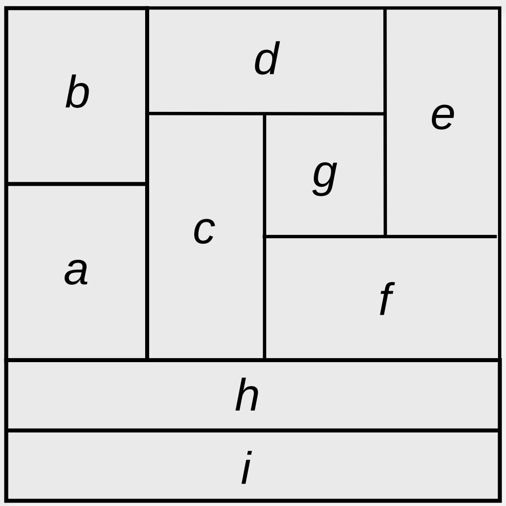
slide from *M. D. F. Wong, "On Simulated Annealing in EDA", ISPD 2012*

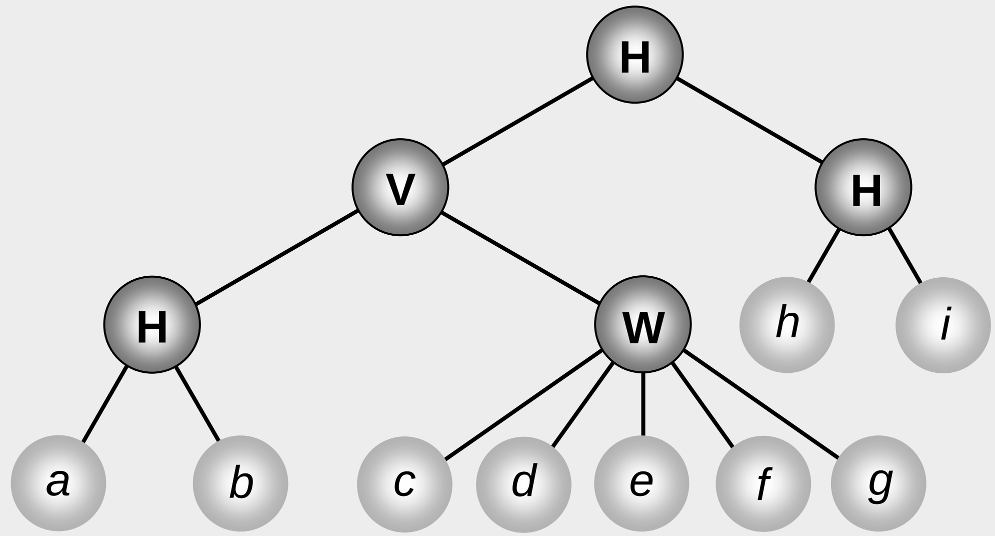
Non-slicing floorplans (wheels)





Floorplan tree: Tree that represents a hierarchical floorplan





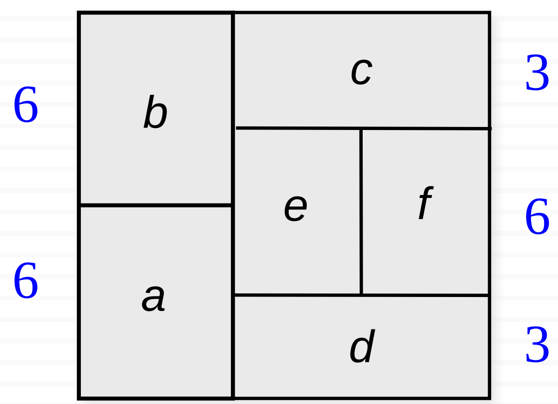
H Horizontal division (objects to the top and bottom)

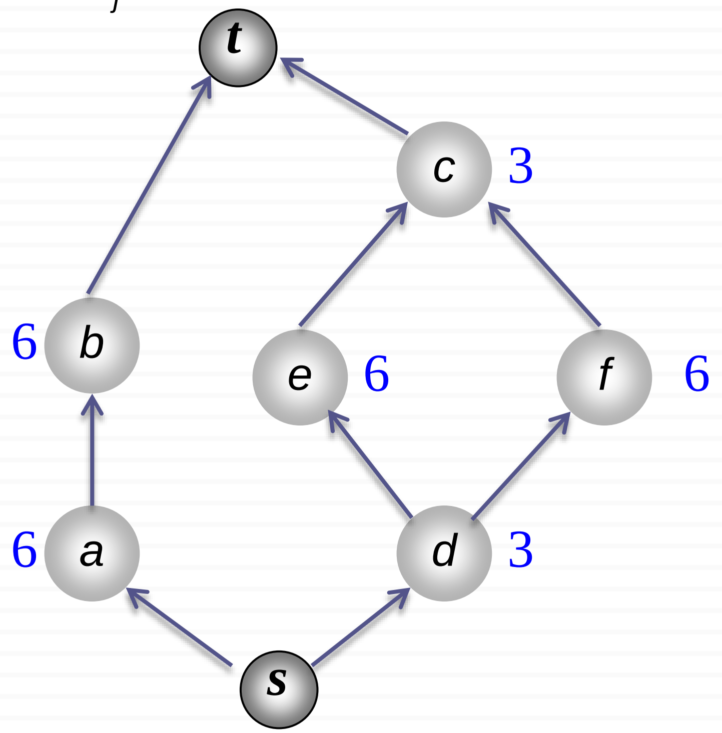
V Vertical division (objects to the left and right)

W Wheel (4 objects cycled around a center object)

Terminology: Vertical Constraint Graph

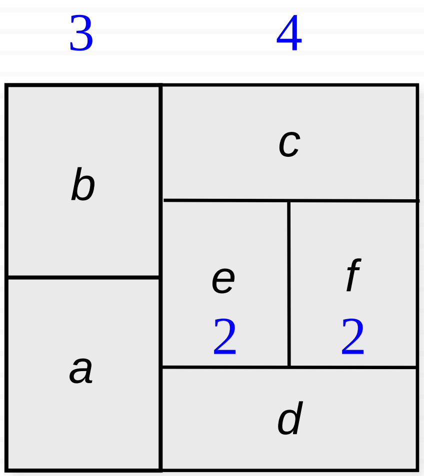
- In a vertical constraint graph (VCG), node weights represent the heights of the corresponding blocks.
  - Two nodes v_i and v_j, with corresponding blocks m_i and m_j, are connected with a directed edge from v_i to v_j if m_i is below m_j.

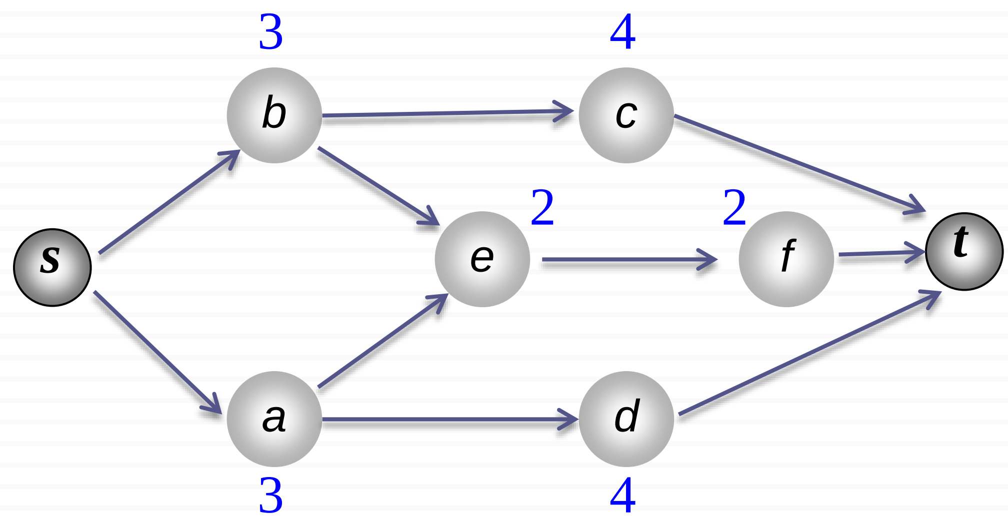




Terminology: Horizontal Constraint Graph

- In a horizontal constraint graph (HCG), node weights represent the widths of the corresponding blocks.
  - Two nodes v_i and v_j, with corresponding blocks m_i and m_j, are connected with a directed edge from v_i to v_j if m_i is to the left of m_j.

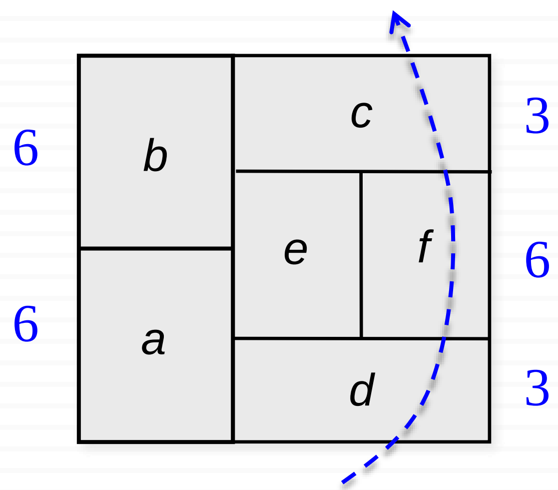


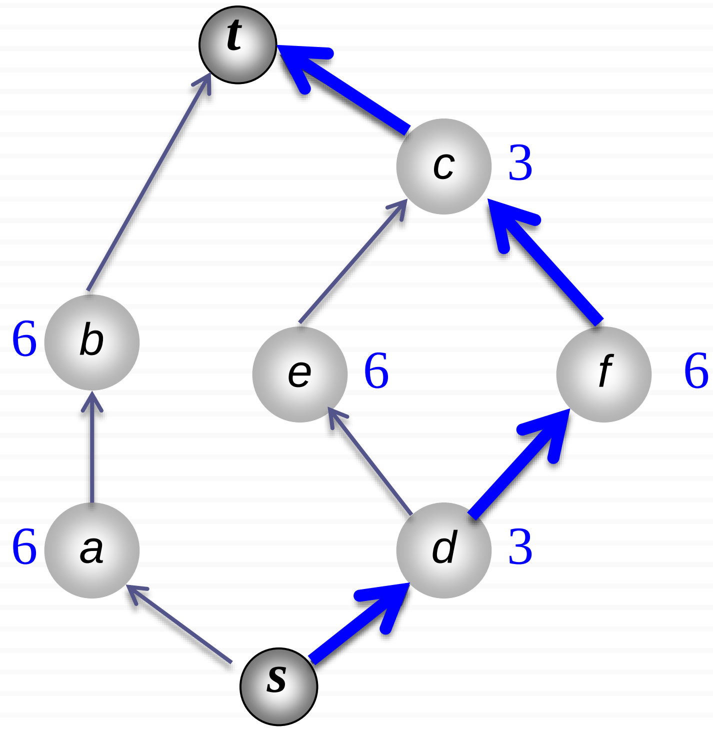


Longest Path in a VCG

- What does the longest path in the VCG correspond to?

  ➔ The minimum required floorplan height

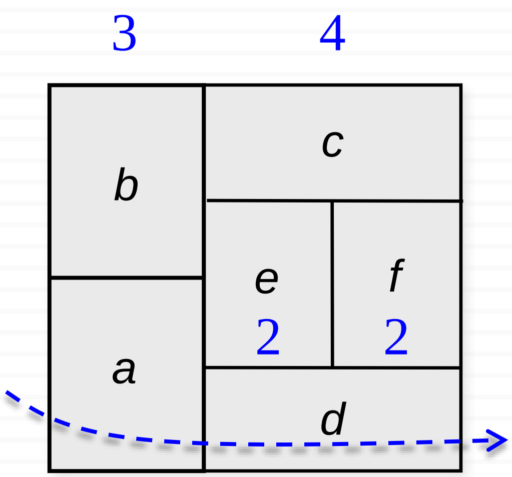


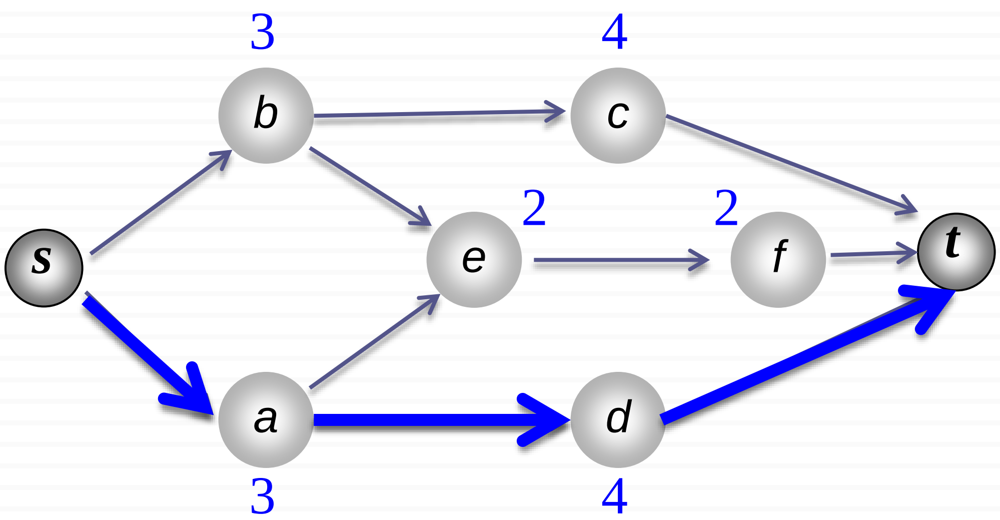


Longest Path in HCG

- What does the longest path in the HCG correspond to?

  ➔ The minimum required floorplan width





- In a vertical constraint graph (VCG), node weights represent the heights of the corresponding blocks.
  - Two nodes v_i and v_j, with corresponding blocks m_i and m_j, are connected with a directed edge from v_i to v_j if m_i is below m_j.

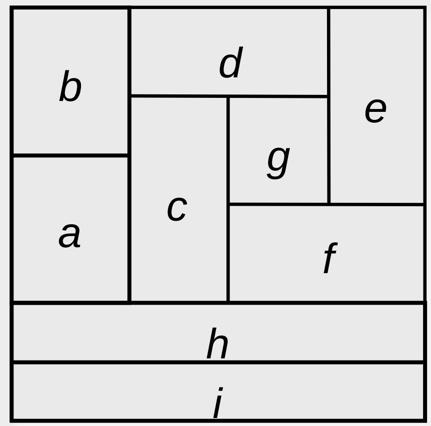
- In a horizontal constraint graph (HCG), node weights represent the widths of the corresponding blocks.
  - Two nodes v_i and v_j, with corresponding blocks m_i and m_j, are connected with a directed edge from v_i to v_j if m_i is to the left of m_j.

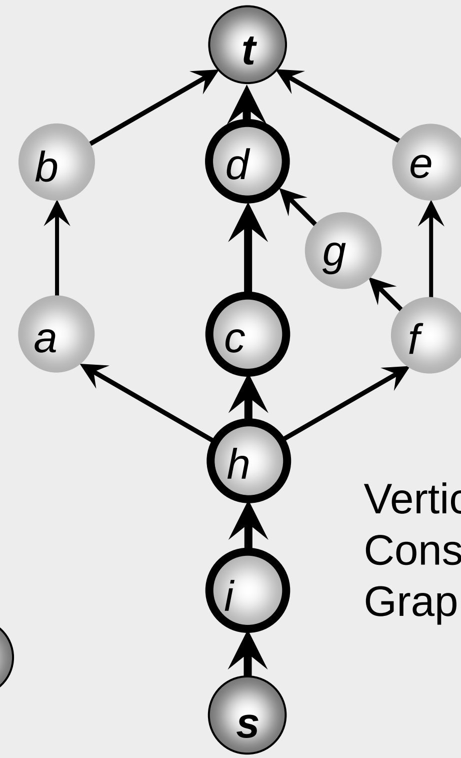
- The longest path(s) in the VCG / HCG correspond(s) to the minimum vertical / horizontal floorplan span required to pack the blocks (floorplan height / width).

- A constraint-graph pair is a floorplan representation that consists of two directed graphs – *vertical constraint graph* and *horizontal constraint graph* – which capture the relations between block positions.

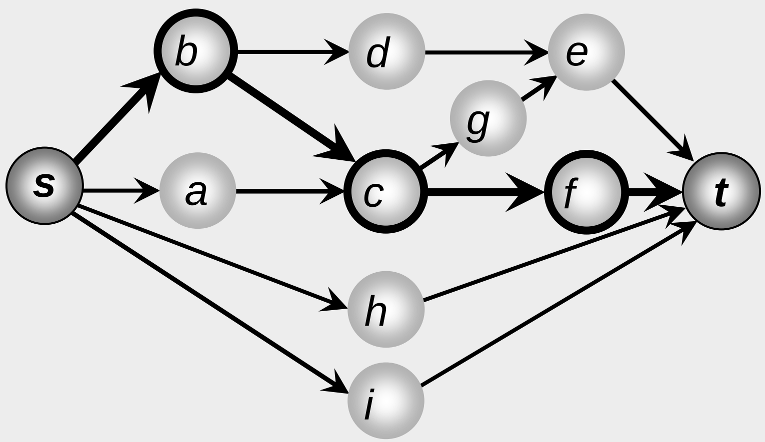
3.3 Terminology

Constraint graphs





Vertical Constraint Graph

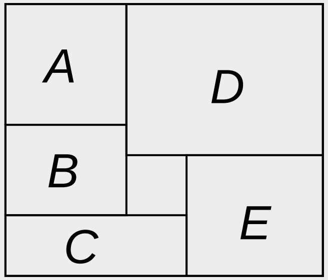


Horizontal Constraint Graph

Sequence pair

- Two permutations represent geometric relations between every pair of blocks

- Example: (*ABDCE*, *CBAED*)



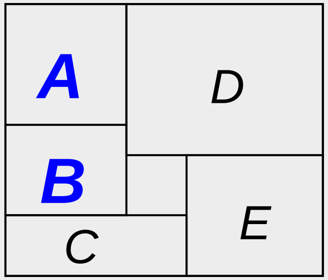
- Horizontal and vertical relations between blocks *A* and *B*:

(… *A* … *B* … **,** … *A* … *B* …) → *A* is left of *B*
(… *B* … *A* … **,** … *A* … *B* …) → *A* is below *B*
(… *A* … *B* … **,** … *B* … *A* …) → *A* is above *B*
(… *B* … *A* … **,** … *B* … *A* …) → *A* is right of *B*

Sequence pair

- Two permutations represent geometric relations between every pair of blocks

- Example: (*AB*DCE, C*BA*ED)



➔ A is above B

- Horizontal and vertical relations between blocks *A* and *B*:

(… *A* … *B* … , … *A* … *B* …) → *A* is left of *B*
(… *B* … *A* … , … *A* … *B* …) → *A* is below *B*
(… *A* … *B* … , … *B* … *A* …) → *A* is above *B*
(… *B* … *A* … , … *B* … *A* …) → *A* is right of *B*

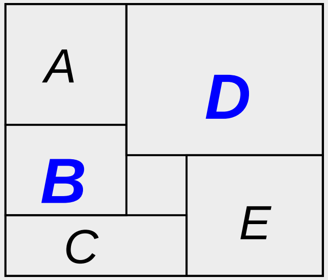
3.3 Terminology

Sequence pair

- Two permutations represent geometric relations between every pair of blocks

- Example: (*ABDCE*, *CBAED*)

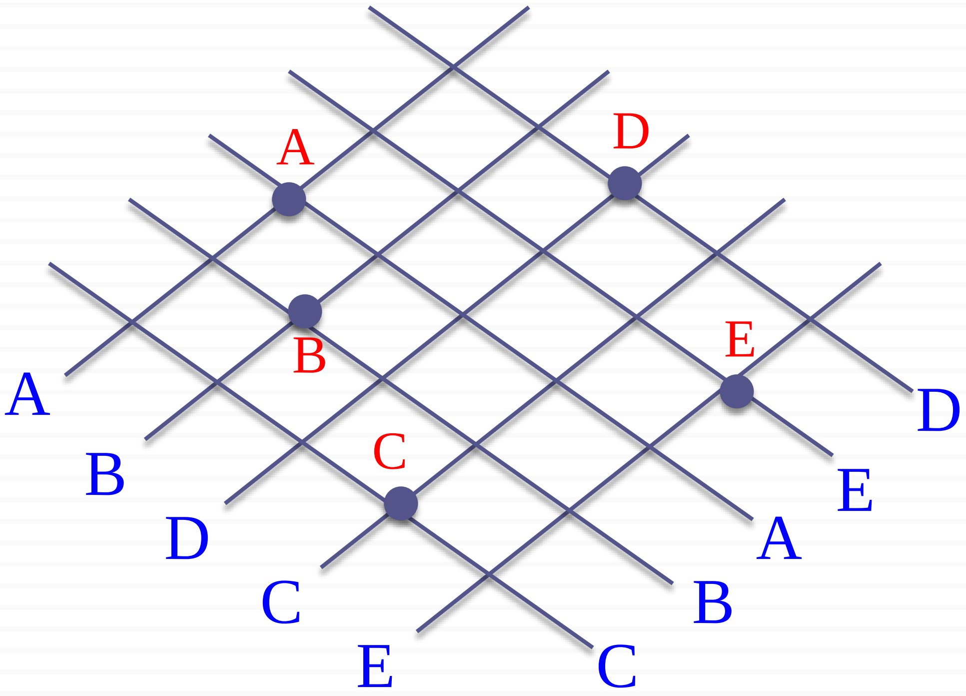
  ➔ B is left of D

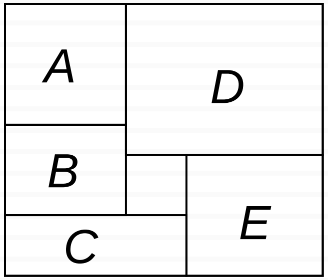


- Horizontal and vertical relations between blocks *A* and *B*:

  (… *A* … *B* … , … *A* … *B* …) → *A* is left of *B*
  (… *A* … *B* … , … *B* … *A* …) → *A* is above *B*
  (… *B* … *A* … , … *A* … *B* …) → *A* is below *B*
  (… *B* … *A* … , … *B* … *A* …) → *A* is right of *B*

Sequence Pair: Intuition

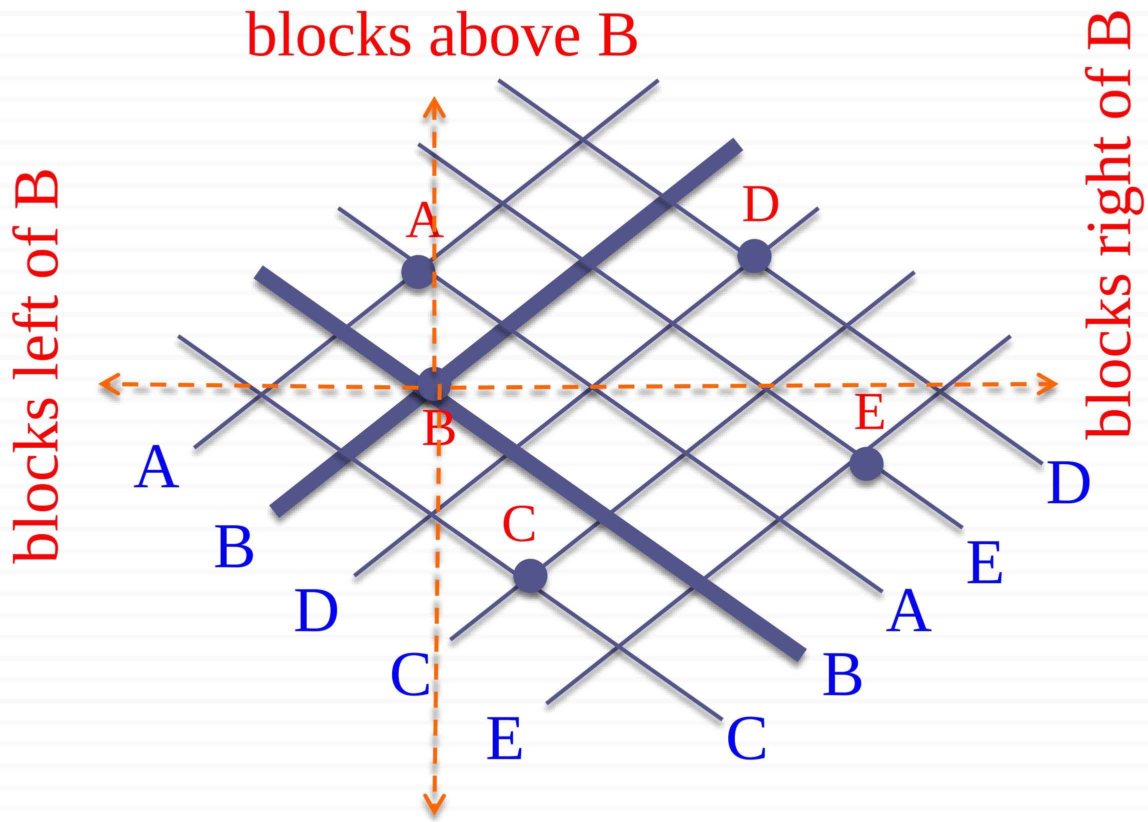


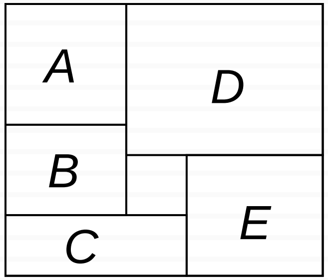


Sequence 1: ABDCE

Sequence 2: CBAED

Sequence Pair: Intuition

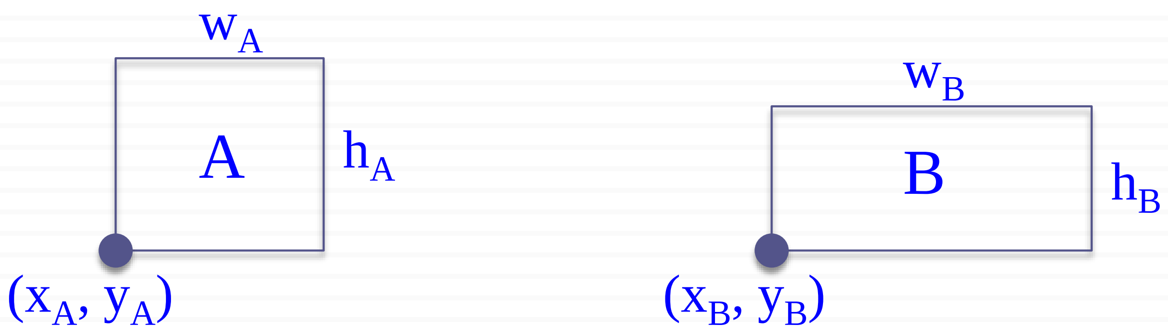




Sequence 1: ABDCE

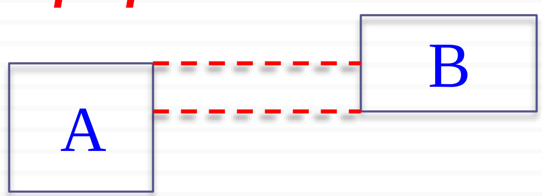
Sequence 2: CBAED

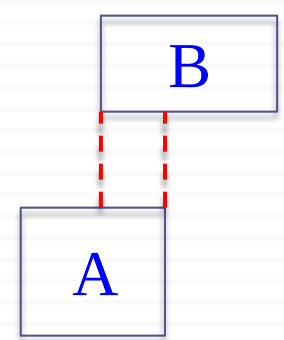
Horizontal and Vertical Constraints



If $x_A + w_A \leq x_B$ and $!\,(y_A + h_A \leq y_B \;\text{ or }\; y_B + h_B \leq y_A)$

➔ ***A is left of B***



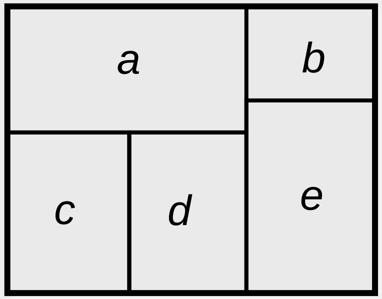


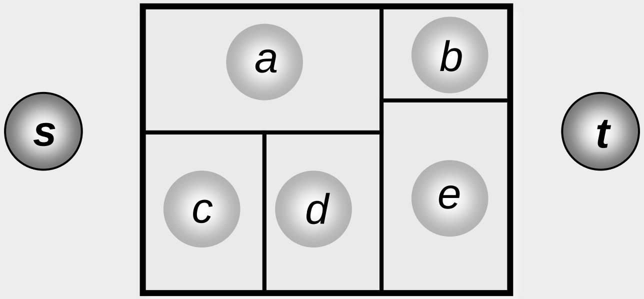
If $y_A + h_A \leq y_B$ and $!\,(x_A + w_A \leq x_B \;\text{ or }\; x_B + w_B \leq x_A)$

➔ ***A is below B***

3.4.1 Floorplan to a Constraint-Graph Pair

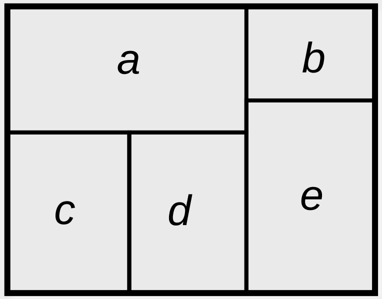
- Create nodes for every block. In addition, create a source node and a sink one.

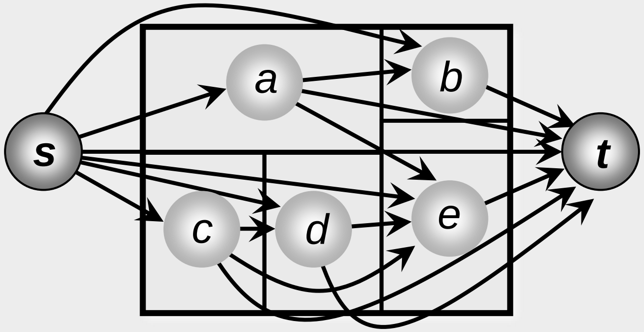




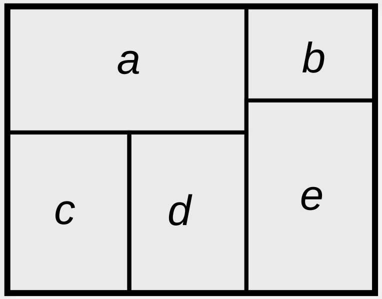
3.4.1 Floorplan to a Constraint-Graph Pair

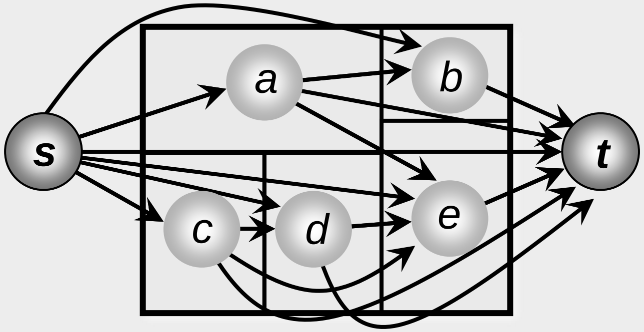
- Create nodes for every block. In addition, create a source node and a sink one.
- Add a directed edge (*A*,*B*) if Block *A* is to the left of Block *B*. (HCG)





- Create nodes for every block. In addition, create a source node and a sink one.
- Add a directed edge (*A*,*B*) if Block *A* is to the left of Block *B*. (HCG)
- Remove the redundant edges that cannot be derived from other edges by transitivity.

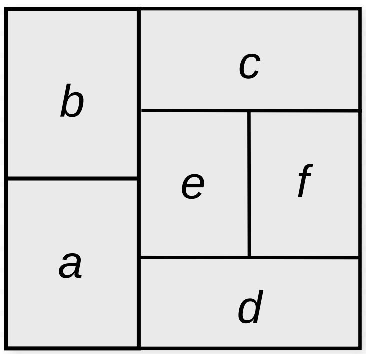




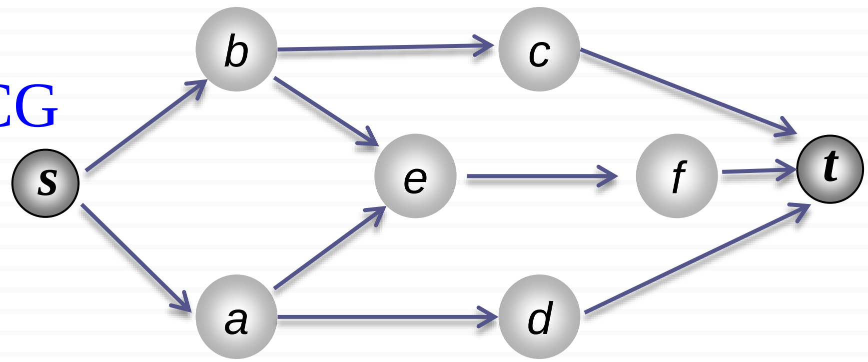
Floorplan to a Sequence Pair

Step 1: Consider the constraints related to HCG

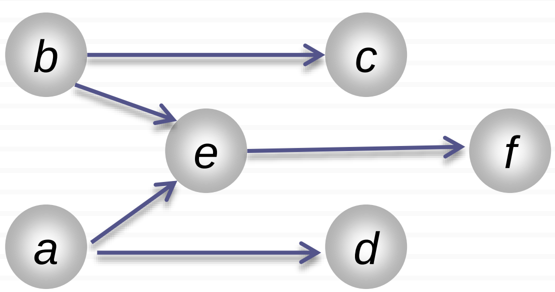
$(\ldots A \ldots B \ldots, \ldots A \ldots B \ldots) \rightarrow A$ is left of B



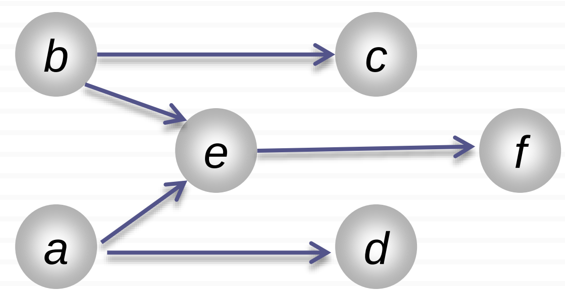
HCG



Constraints for SP 1

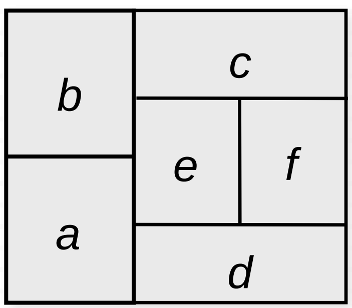


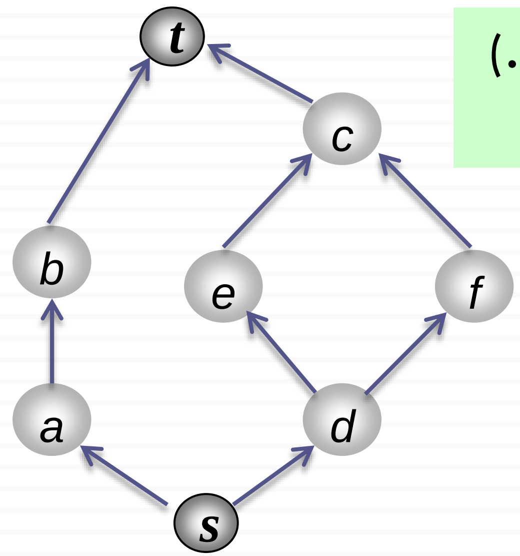
Constraints for SP 2



Floorplan to a Sequence Pair

Step 2: Consider the constraints related to VCG

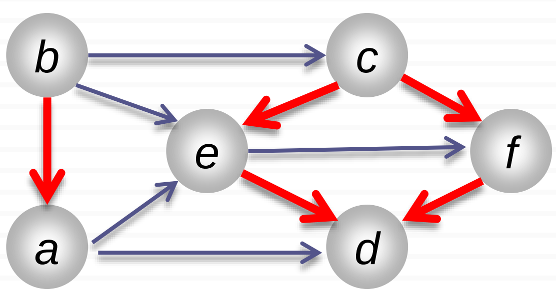




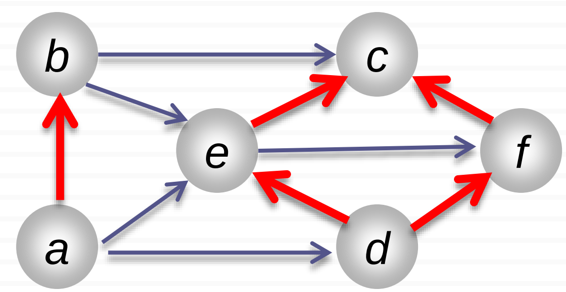
(… *B* … *A* … , … *A* … *B* …)

→ *A* is below *B*

Constraints for SP 1

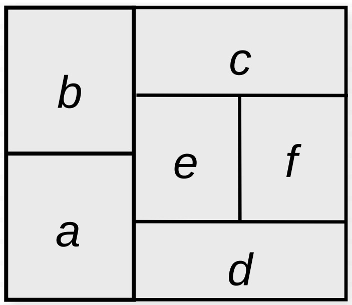


Constraints for SP 2



Floorplan to a Sequence Pair

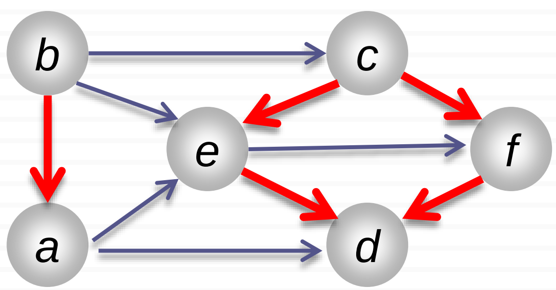
Step 3: Create the sequence pairs based on the constraints



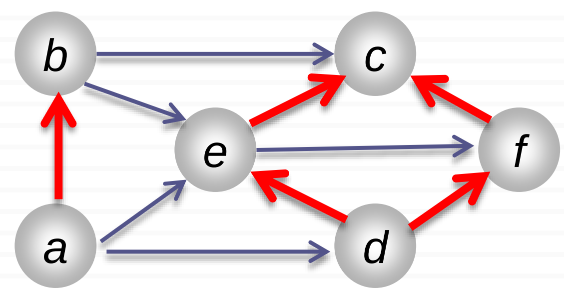
(… A … B … , … A … B …) → A is left of B

(… B … A … , … A … B …) → A is below B

Constraints for SP 1



Constraints for SP 2



Sequence 1: bacefd

Sequence 2: abdefc

Sequence Pair to a Floorplan

- Method 1 (simpler):

  1. Create constraint graphs: HCG and VCG
  2. Pack the blocks based on HCG and VCG (next slides)

  Complexity: $O(n^2)$

- Method 2

  Pack the blocks based on the sequence pair directly

  Complexity: O(nlgn)

Constraint Graph Pair to a Floorplan

- Given an HCG and a VCG, we can compute a packing solution that satisfies all the constraints.

- Basic idea:
  - Compute the longest path on HCG
  - The coordinate computed for each vertex will be the x-coordinate of the corresponding block in the packed floorplan.
  - Compute the longest path on VCG
  - The coordinate computed for each vertex will be the y-coordinate of the corresponding block in the packed floorplan.

Reminder: Longest Path Algorithm

LONGEST-PATH (G)

for each vertex u in G

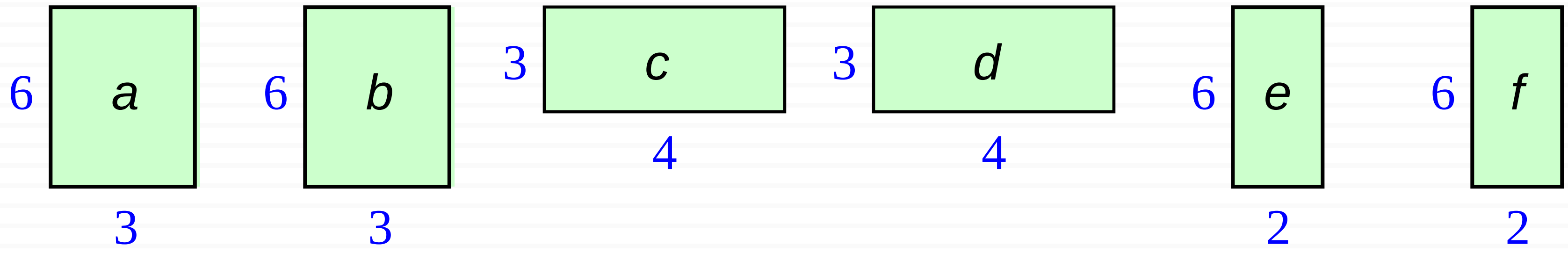
coord[u] = 0

for each vertex u in G in *topological order*

for each edge (u→v) in G do

coord[v] = **max** (coord[v], coord[u]+wt(u))

Example: Sequence Pair to a Floorplan



Compute HCG and VCG for the sequence pair:

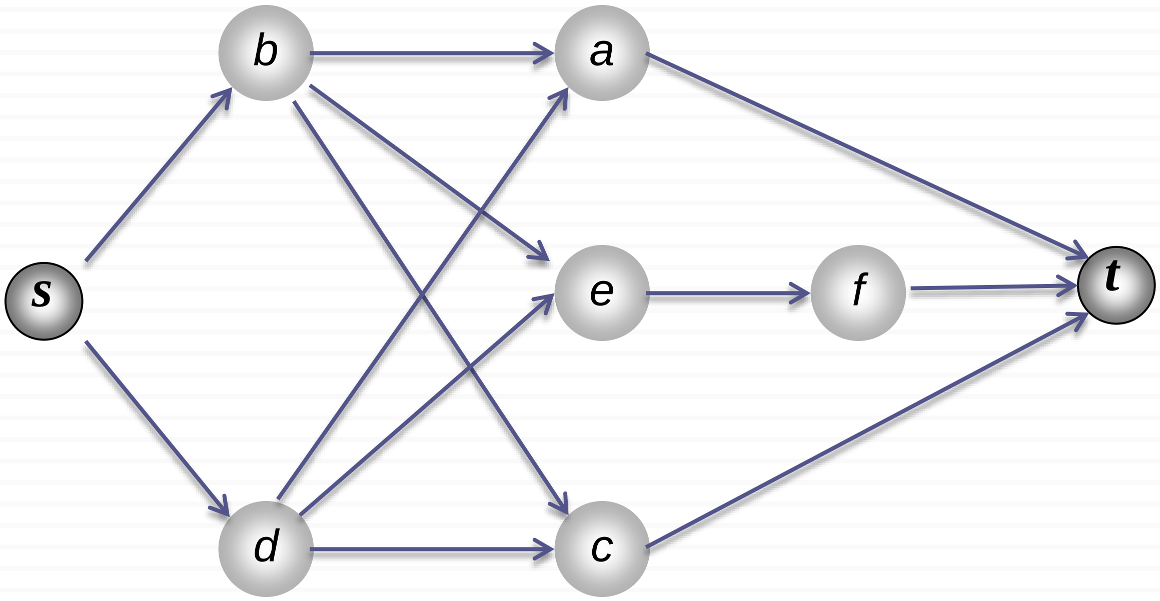
S1 = bdcefa S2 = dbaefc

(… *A* … *B* … , … *A* … *B* …) → *A* is left of *B*

(… *B* … *A* … , … *A* … *B* …) → *A* is below *B*

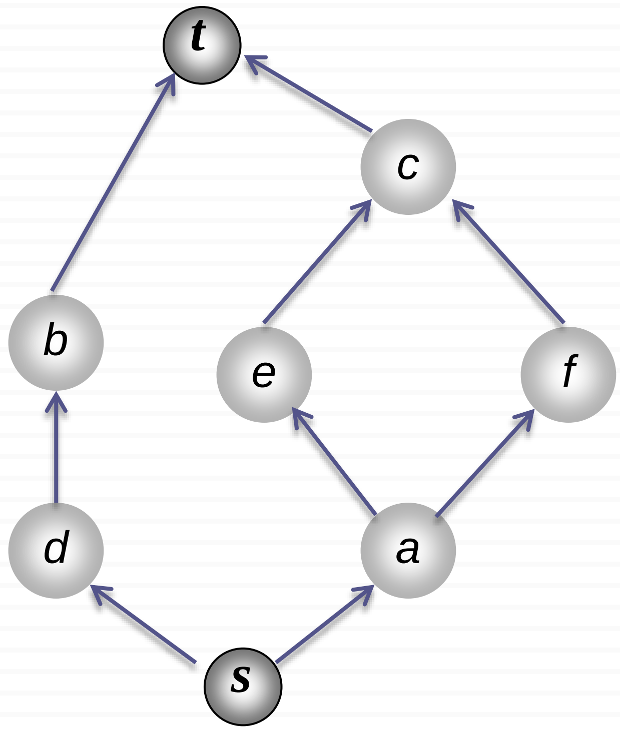
Example: HCG for sequence pair

S1 = bdcefa   S2 = dbaefc

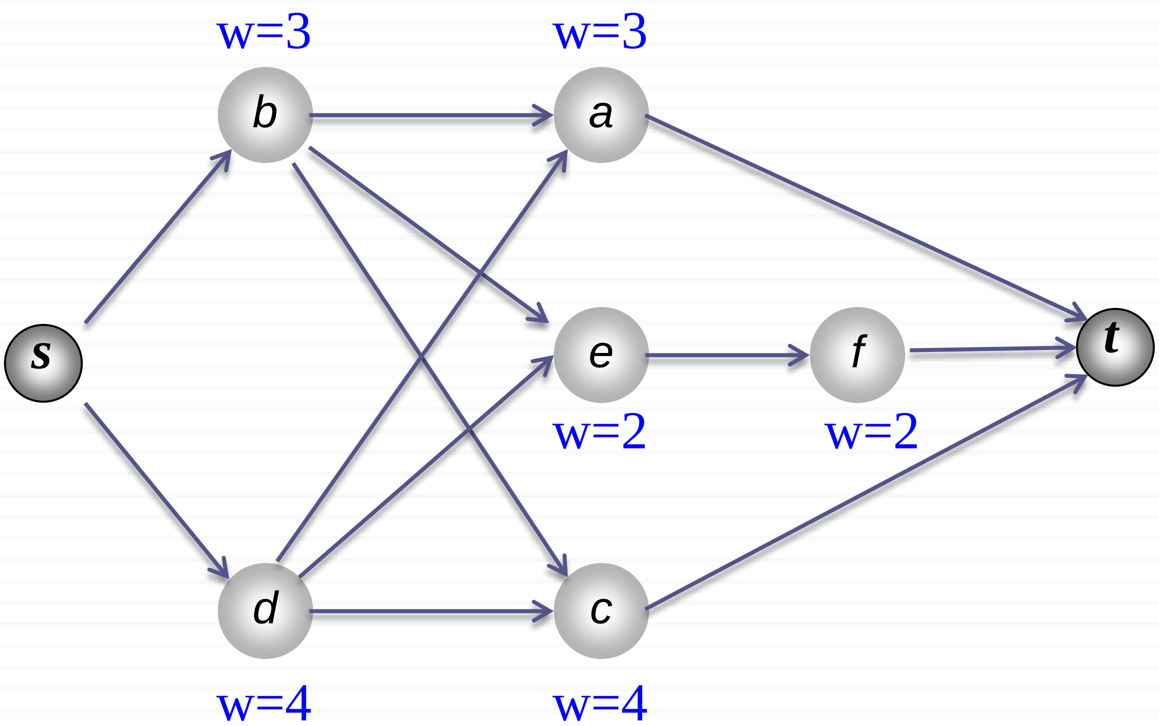


Example: VCG for Sequence Pair

S1 = bdcefa S2 = dbaefc

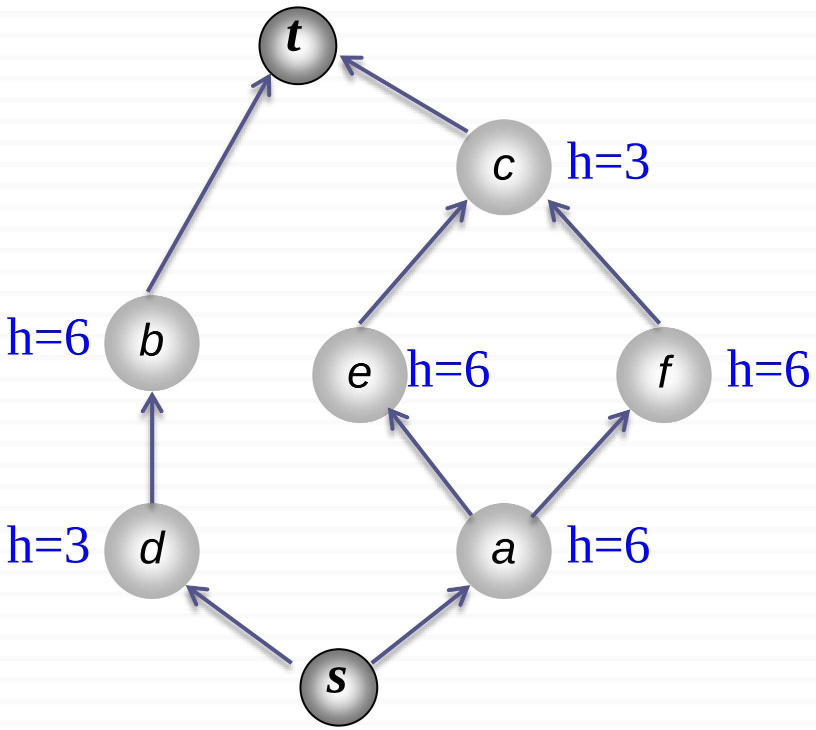


Example: Longest Path in HCG



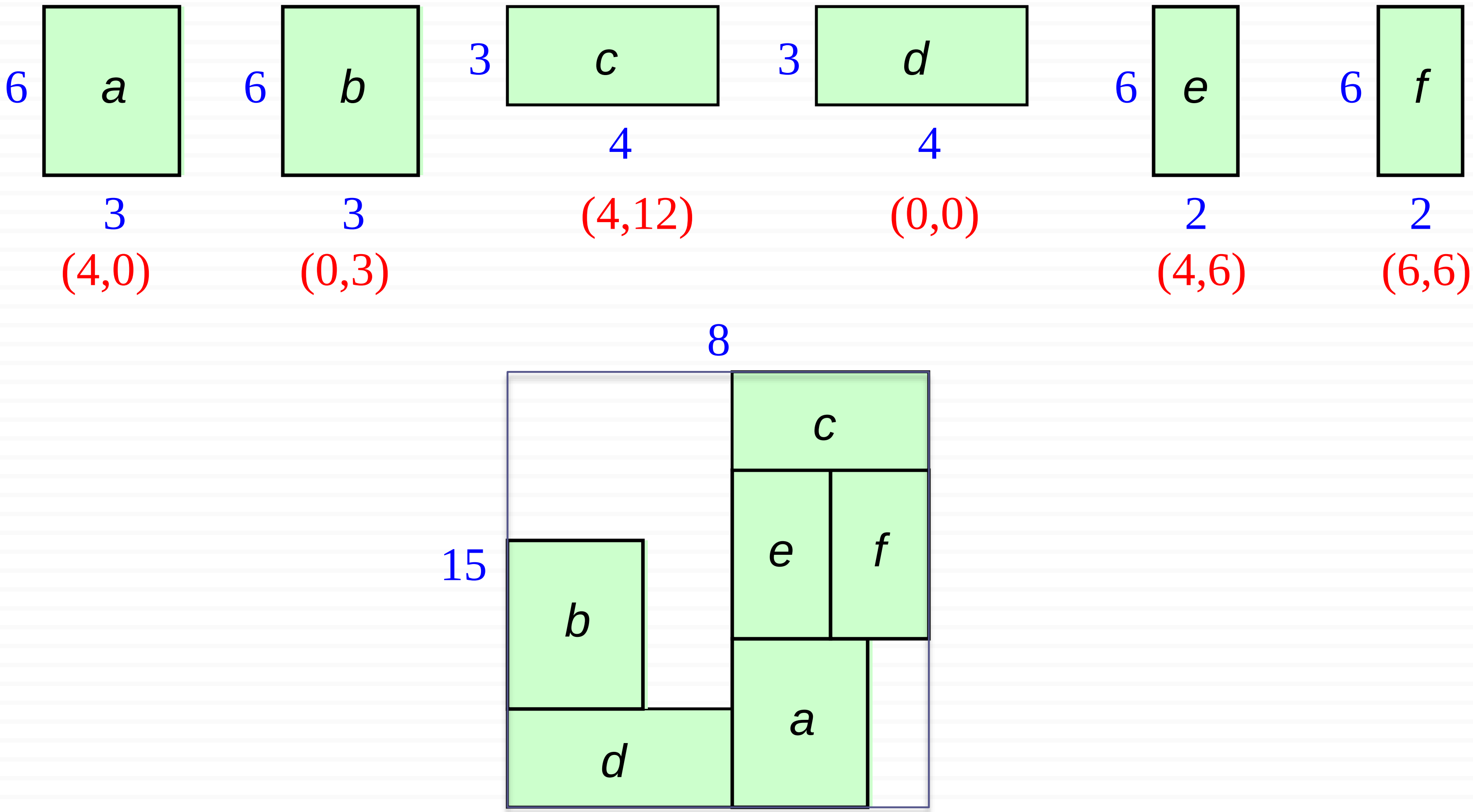
x(s) = 0
x(b) = 0
x(d) = 0
x(e) = 4
x(c) = 4
x(a) = 4
x(f) = 6
x(t) = 8

Example: Longest Path in VCG

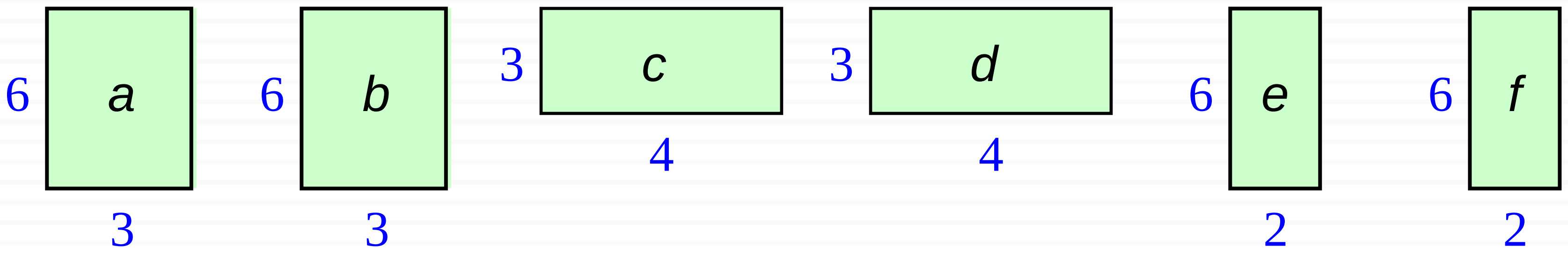


$y(s) = 0$
$y(a) = 0$
$y(d) = 0$
$y(b) = 3$
$y(e) = 6$
$y(f) = 6$
$y(c) = 12$
$y(t) = 15$

Example: Packing



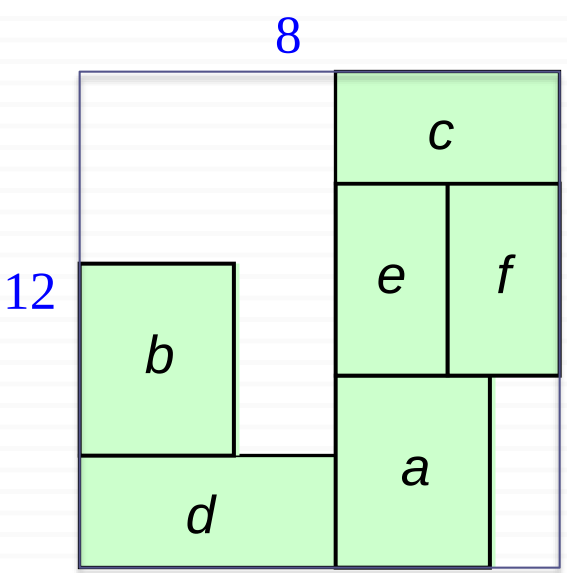
Example: Summary



The sequence pair:

S1 = bdcefa   S2 = dbaefc

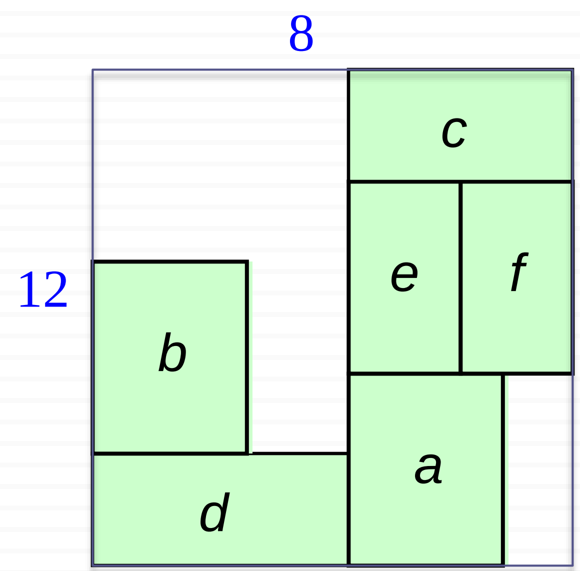
corresponds to the packed floorplan:



Example: Perturbation

The original sequence pair:

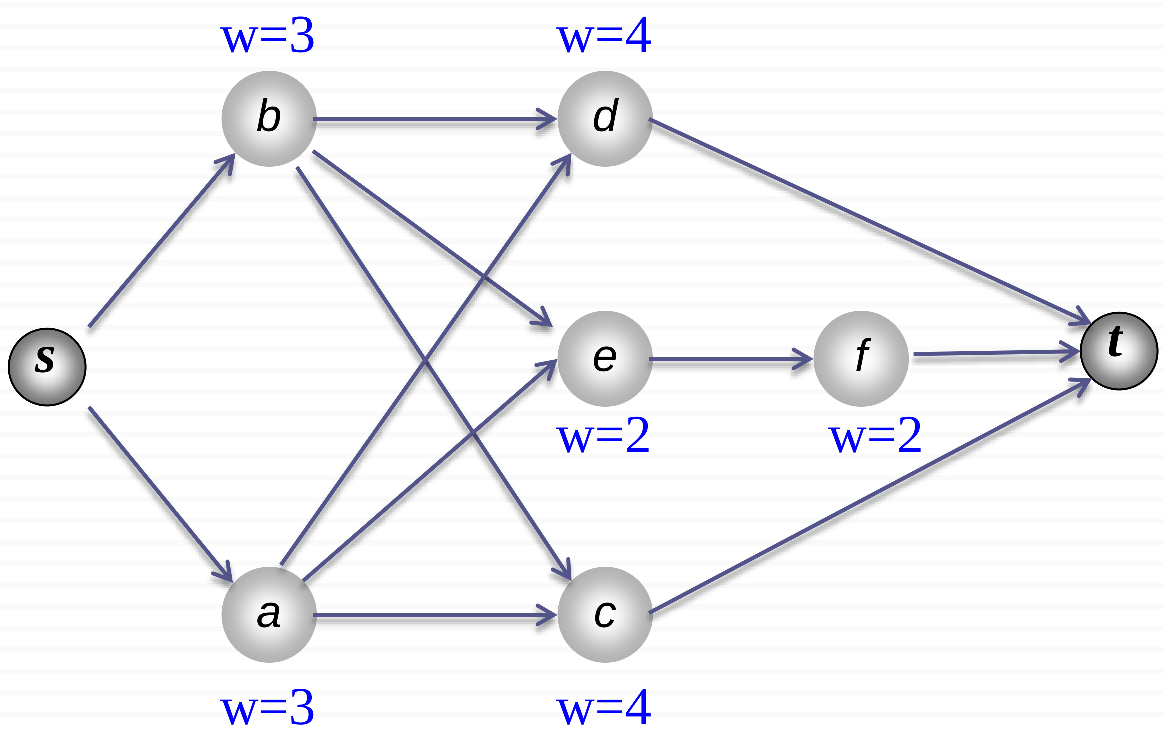
S1 = b**d**cef**a** S2 = **d**b**a**efc



What happens if we swap the positions of a and d in both sequences?

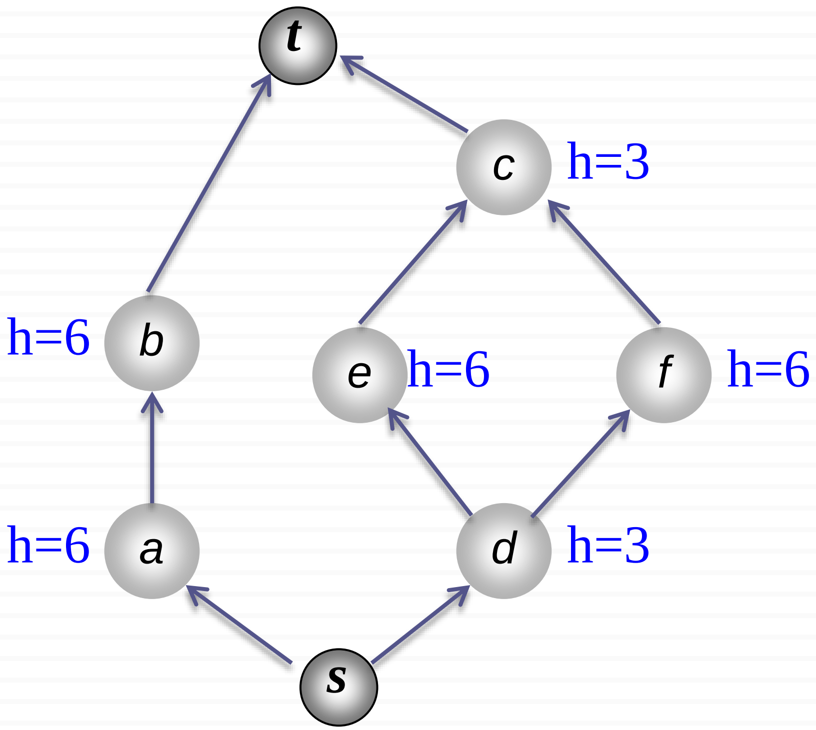
i.e. S1 = b**a**cef**d** S2 = **a**b**d**efc

Example: Longest Path in HCG after Perturbation



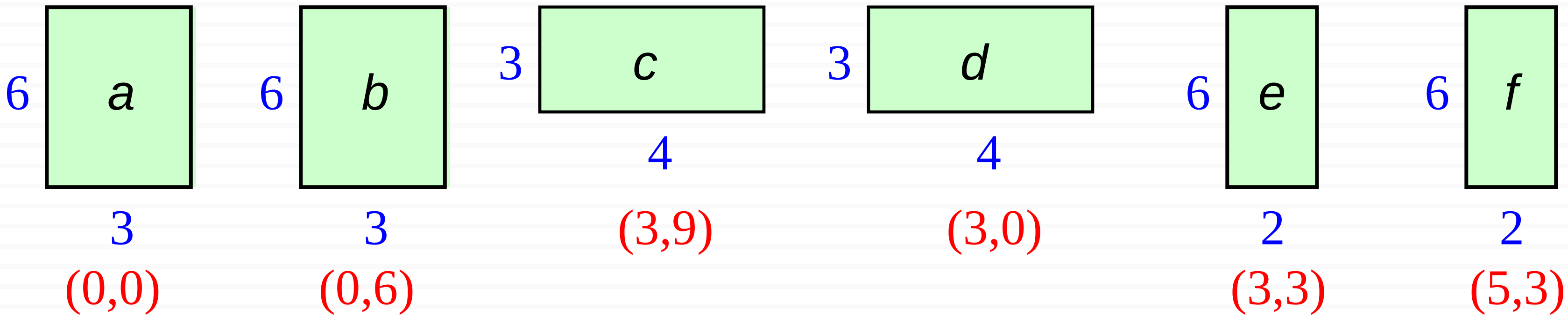
x(s) = 0
x(b) = 0
x(a) = 0
x(e) = 3
x(c) = 3
x(d) = 3
x(f) = 5
x(t) = 7

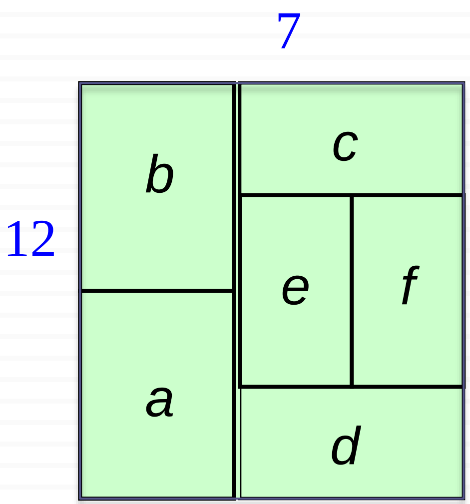
Example: Longest Path in VCG



y(s) = 0
y(a) = 0
y(d) = 0
y(b) = 6
y(e) = 3
y(f) = 3
y(c) = 9
y(t) = 12

Example: Packing

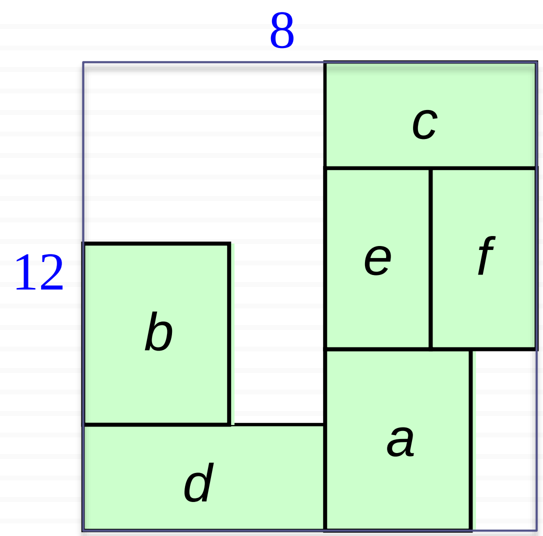




Example: Summary

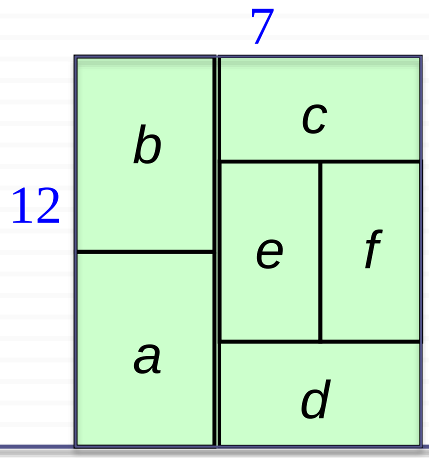
The original sequence pair:

S1 = bdcefa   S2 = dbaefc



Swap positions of a and d in both sequences:

S1 = bacefd   S2 = abdefc



Sequence Pair to a Floorplan

- Method 1 (simpler):

  1. Create constraint graphs: HCG and VCG
  2. Pack the blocks based on HCG and VCG (next slides)

  Complexity: $O(n^2)$

- Method 2

  Pack the blocks based on the sequence pair directly

  Complexity: $O(nlgn)$

Reminder: A Common Subsequence

- Given two sequences X and Y:

  Z is a ***common subsequence*** of X and Y if Z is a subsequence of both X and Y.

- Example:

  X = bdcefa   Y = dbaefc

  Z = **bef**   (a common subsequence of X and Y)

  because X = **b**dc**ef**a   Y = d**b**a**ef**c

Reminder: Longest Common Subsequence (LCS)

- Each element in the sequence can have a weight defined
- Example:

Elements: a b c d e f

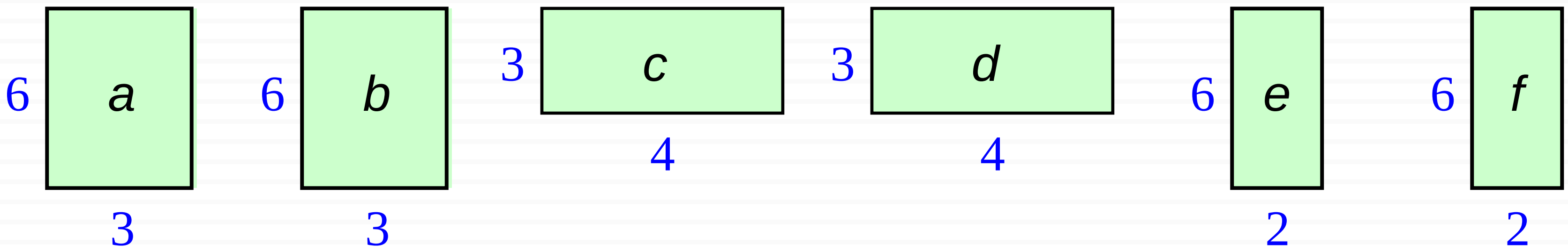
Weights: 3 3 4 4 2 2

- ***Longest common subsequence (LCS)*** of two sequences is the common sequence with the maximum weight

X = bdcefa   Y = dbaefc

LCS (X, Y) = def   *with weight = 4 + 2 + 2 = 8*

LCS of a Sequence Pair



(… *A* … *B* … , … *A* … *B* …) → *A* is left of *B*

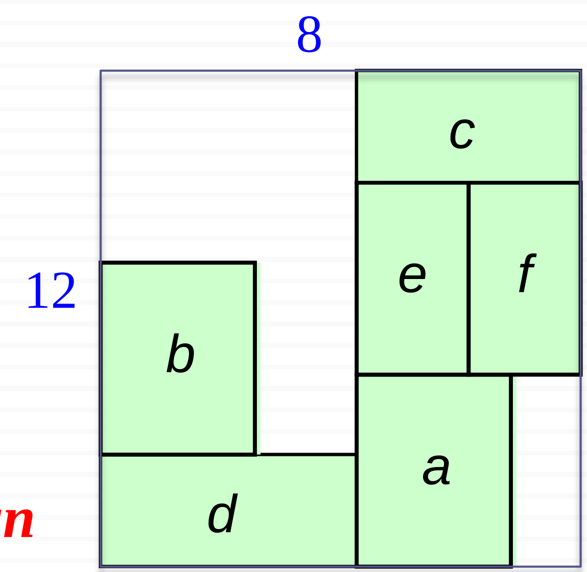
Sequence pair: X = bdcefa Y = dbaefc

Let the **weights** defined as the block **widths**

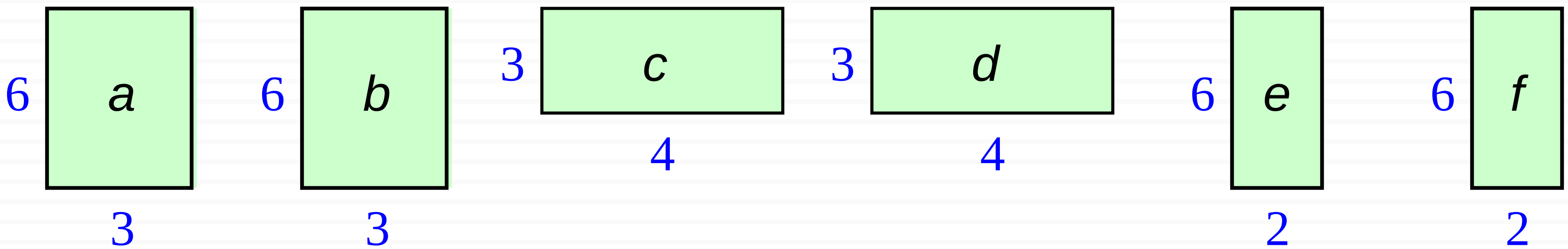
What does the **LCS(X, Y)** correspond to?

LCS(X, Y) = **def**

➔ ***the maximum horizontal span of the floorplan***



LCS of a Sequence Pair



(… B … A … , … A … B …) → A is below B

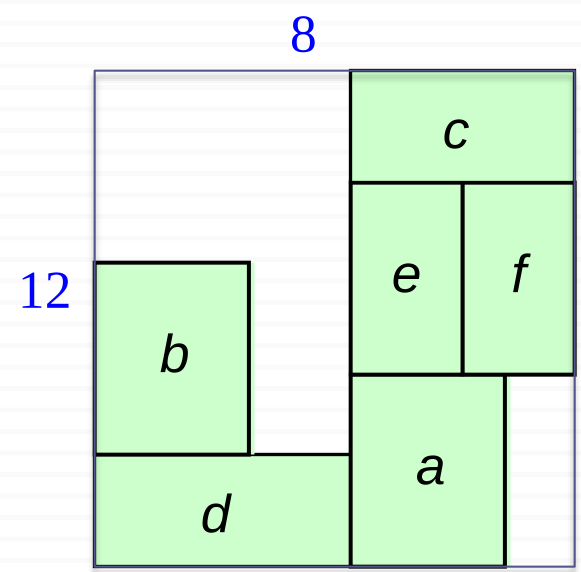
Sequence pair: X = bdcefa Y = dbaefc

Let the **weights** defined as the block **<u>heights</u>**

What does the **LCS(X^R, Y)** correspond to?

LCS(X^R, Y) = **aec**

➔ ***the maximum vertical span of the floorplan***



Sequence Pair to a Floorplan

- How to find the x-coordinate of block b?
  - Consider the location of b in the sequence pair (X,Y)

$$X = X_1 \, b \, X_2 \qquad\qquad Y = Y_1 \, b \, Y_2$$

  - What does LCS (X_1, Y_1) correspond to?
    - the max horizontal span of the blocks **left** of b
  - **x-coord (b) = LCS(X_1, Y_1)**

(… *A* … *B* … , … *A* … *B* …) → *A* is left of *B*

Sequence Pair to a Floorplan

- How to find the y-coordinate of block b?
  - Consider the location of b in the sequence pair (X,Y)

$$X = X_1\ b\ X_2 \qquad\qquad Y = Y_1\ b\ Y_2$$

$$X^R = X_2^R b\ X_1^R$$

  - What does LCS (X_2^R, Y_1) correspond to?
    - the max vertical span of the blocks **below** b
  - **y-coord (b) = LCS(X_2^R, Y_1)**

(… *B* … *A* … , … *A* … *B* …) → *A* is below *B*

Sequence Pair to a Floorplan using an LCS Algorithm

- **Find-LCS**: Given two sequences X and Y consisting of n blocks, return the length of the LCS before each block b

  i.e. Return length of $LCS(X_1, Y_1)$ for each block b for which $X= X_1\ b\ X_2$ and $Y = Y_1\ b\ Y_2$

Inputs:

Block	a	b	c	d	e	f
Weight	3	3	4	4	2	2

X = bdcefa

Y = dbaefc

Output:

LCS length before	4	0	4	0	4	6

Sequence Pair to a Floorplan using an LCS Algorithm

FIND-LCS is solvable in O(nlgn) time

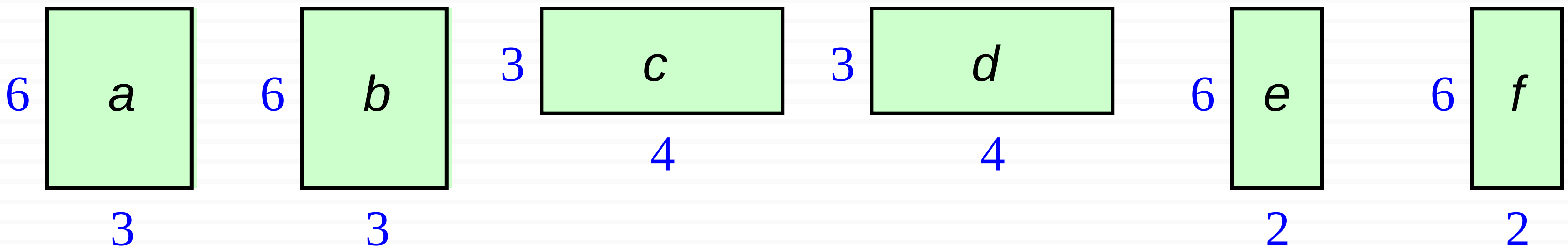
Tang, X. Tian, R. and Wong, D.F., "Fast Evaluations of Sequence Pair in Block Placement by Longest Common Subsequence Computations", DATE 2000

Sequence pair (X,Y) to a packed floorplan:

x-coords = FIND-LCS (X, Y, widths)

y-coords = FIND-LCS (X^R, Y, heights)

Example: Sequence Pair to a Floorplan

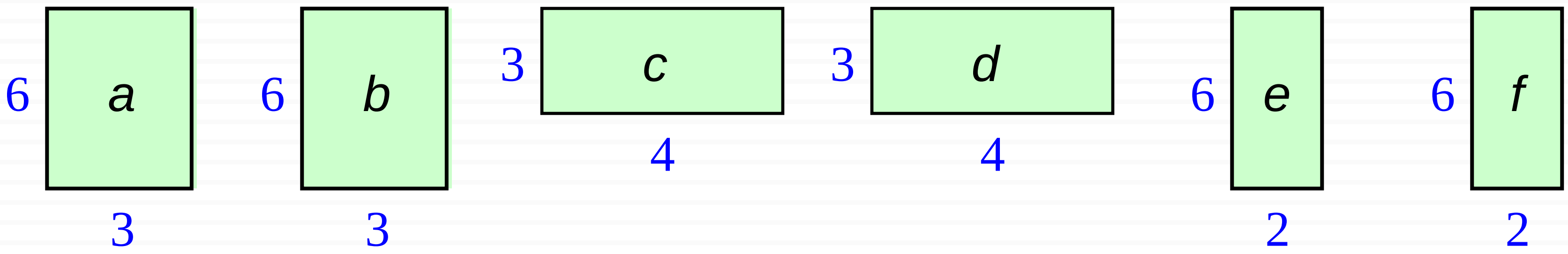


Sequence pair: X = bdcefa Y = dbaefc

x-coords ← **FIND-LCS** (bdcefa, dbaefc, widths)

x-coords = 4 0 4 0 4 6

Example: Sequence Pair to a Floorplan

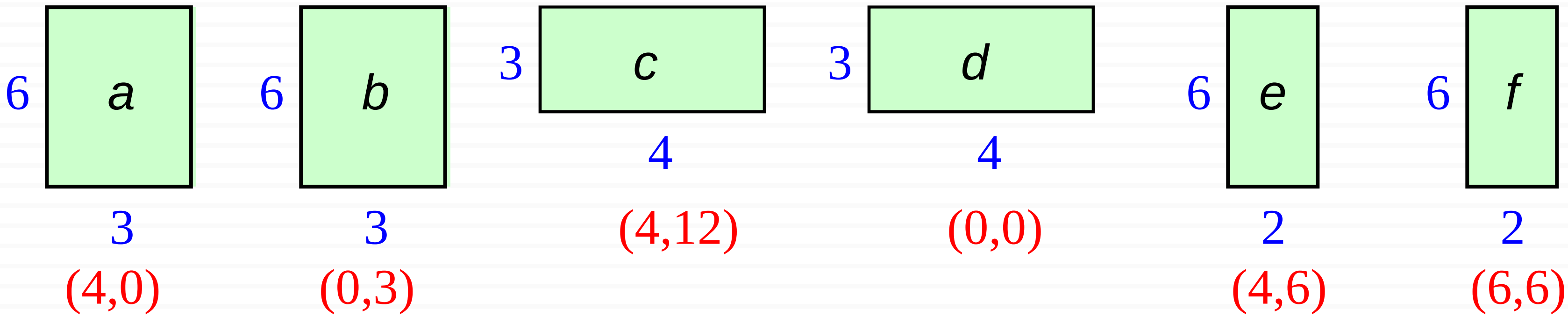


Sequence pair: X = bdcefa Y = dbaefc

y-coords ← **FIND-LCS** (afecdb, dbaefc, heights)

y-coords = 0 3 12 0 6 6

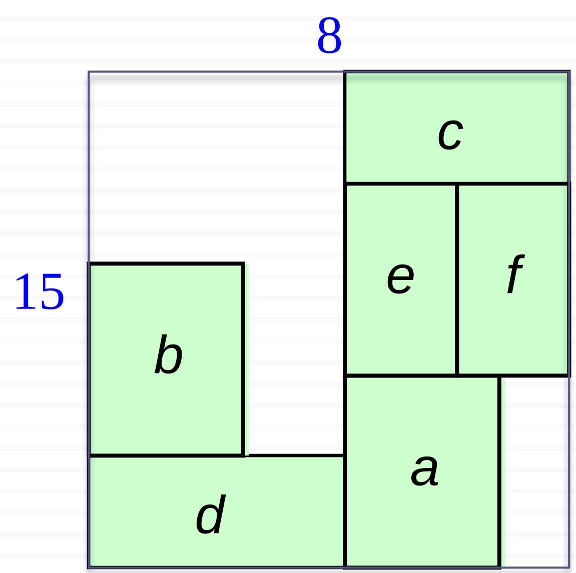
Example: Sequence Pair to a Floorplan



Sequence pair:

X = bdcefa

Y = dbaefc



How will the floorplan change if we swap a and d in sequence X?

3.5 Floorplanning Algorithms

Common Goals

- To minimize the total length of interconnect, subject to an upper bound on the floorplan area

  or

- To simultaneously optimize both wire length and area

Otten, R.: Efficient Floorplan Optimization. Int. Conf. on Computer Design, 499-502, 1983

Floorplan Sizing

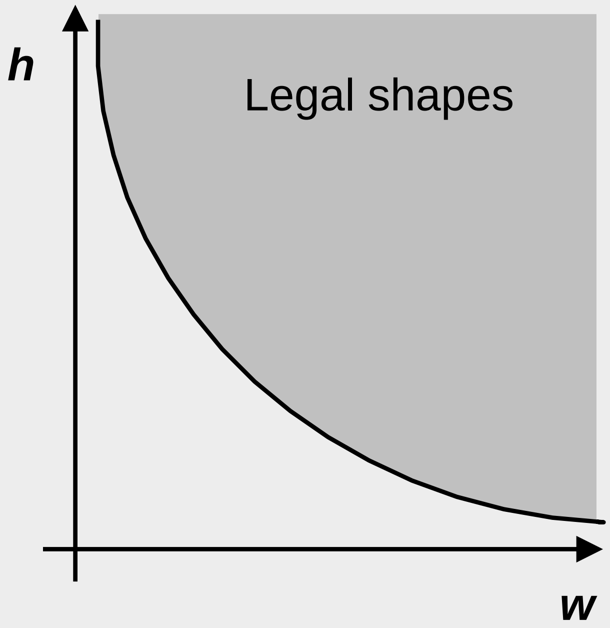
- Each block has the following constraints:
  - Area constraint: $w_{block} \cdot h_{block} \geq area_{block}$
  - Lower bound constraints: $w_{block} \geq w_{LB}$ and $h_{block} \geq h_{LB}$
  - Discrete w_{block} and h_{block} options

- *Min-area floorplan*: For a given slicing floorplan, compute the locations and shapes to obtain the min floorplan area.

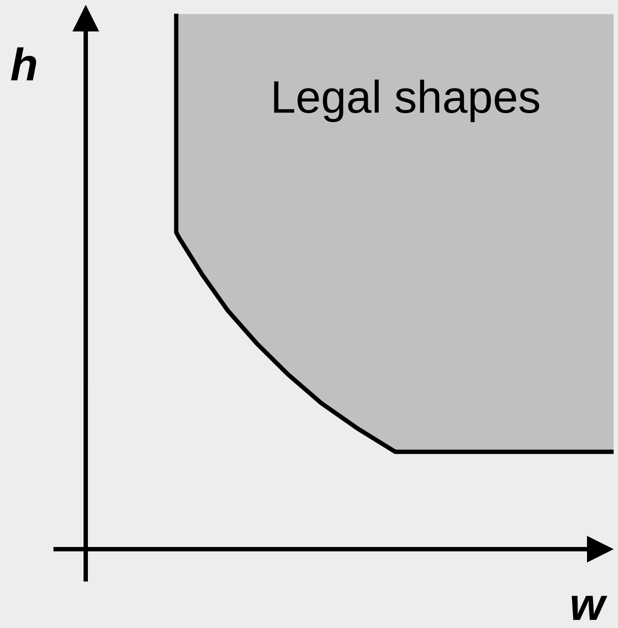
  Is this problem NP-hard?

  No, it's polynomial time solvable!

Shape functions



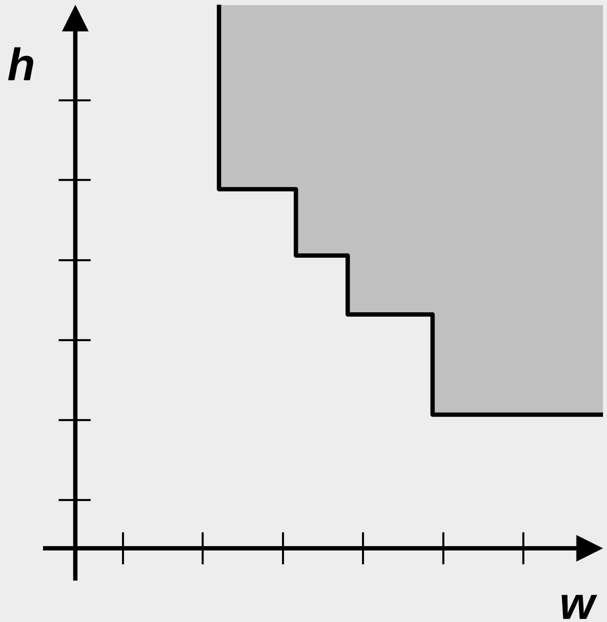
$h * w \geq A$



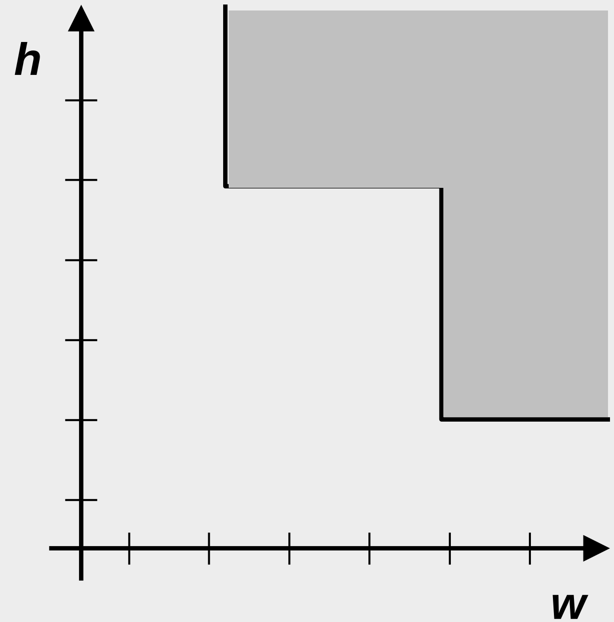
Block with minimum width and height restrictions

Otten, R.: Efficient Floorplan Optimization. Int. Conf. on Computer Design, 499-502, 1983

Shape functions



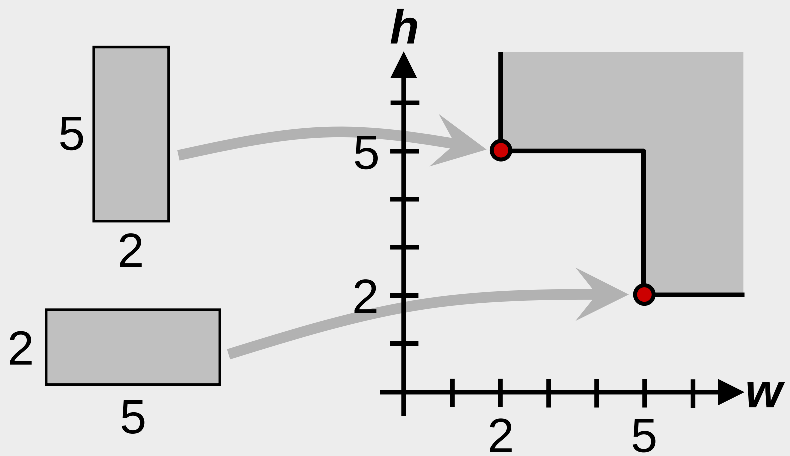
Discrete (*h*,*w*) values



Hard library block

Otten, R.: Efficient Floorplan Optimization. Int. Conf. on Computer Design, 499-502, 1983

Corner points



3.5.1 Floorplan Sizing

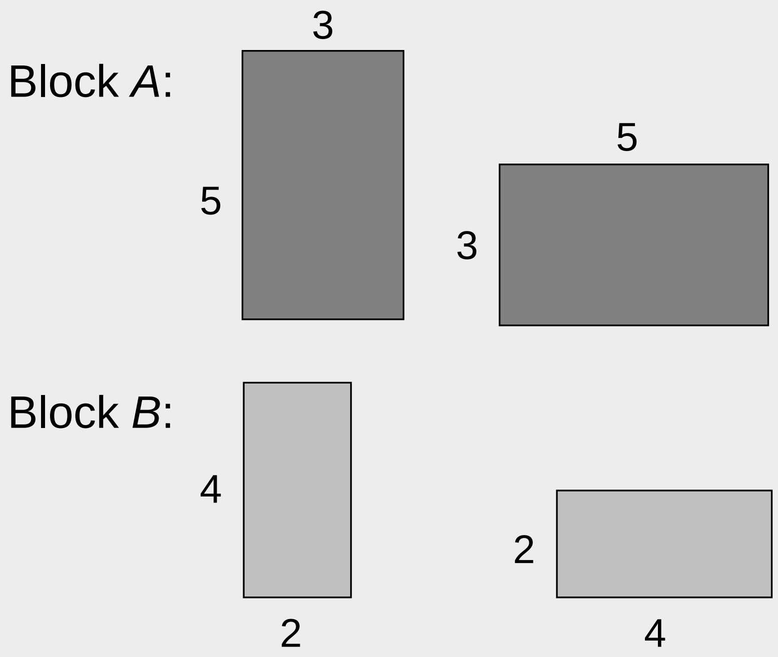
Algorithm

This algorithm finds the **minimum floorplan area** for a given slicing floorplan in polynomial time. For non-slicing floorplans, the problem is NP-hard.

- Construct the shape functions of all individual blocks
- Bottom up: Determine the shape function of the top-level floorplan from the shape functions of the individual blocks
- Top down: From the corner point that corresponds to the minimum top-level floorplan area, trace back to each block’s shape function to find that block’s dimensions and location.

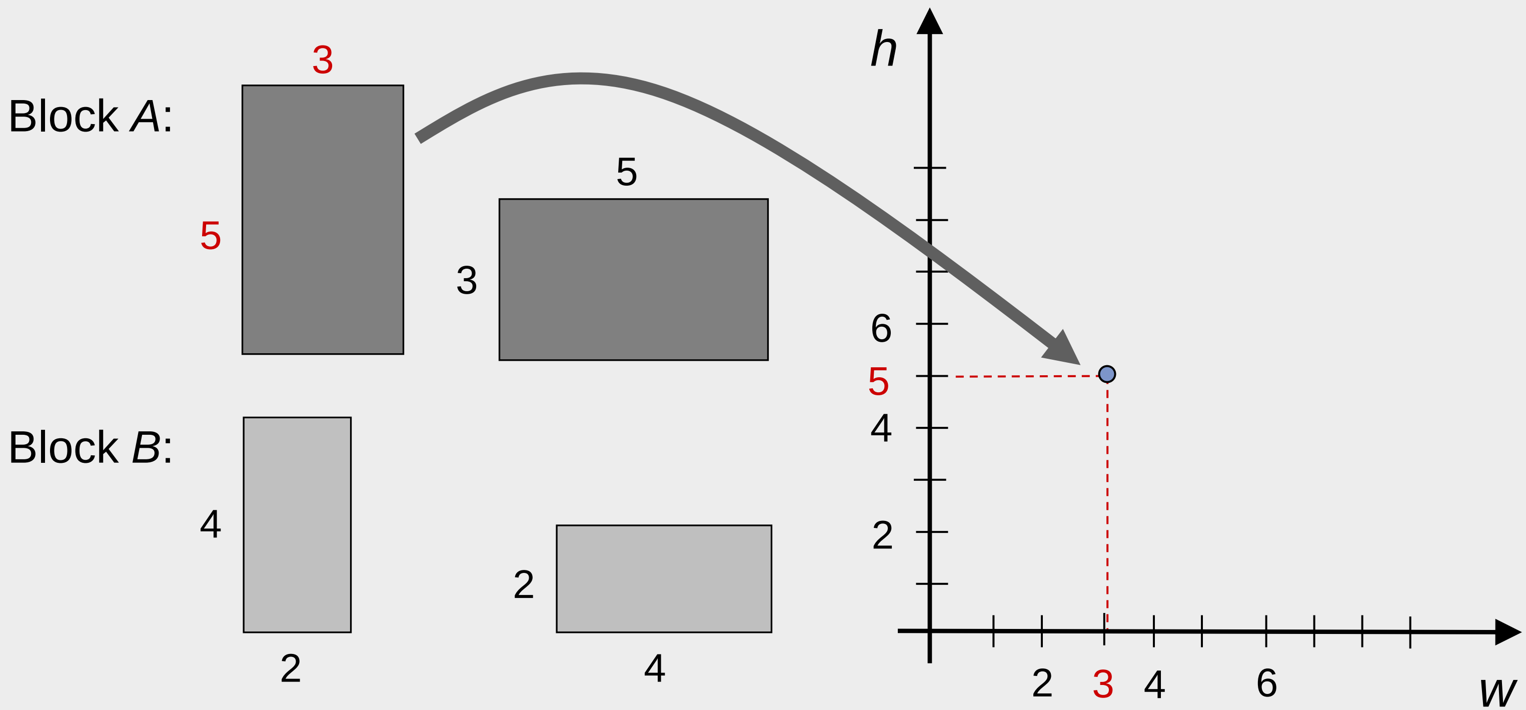
3.5.1 Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks

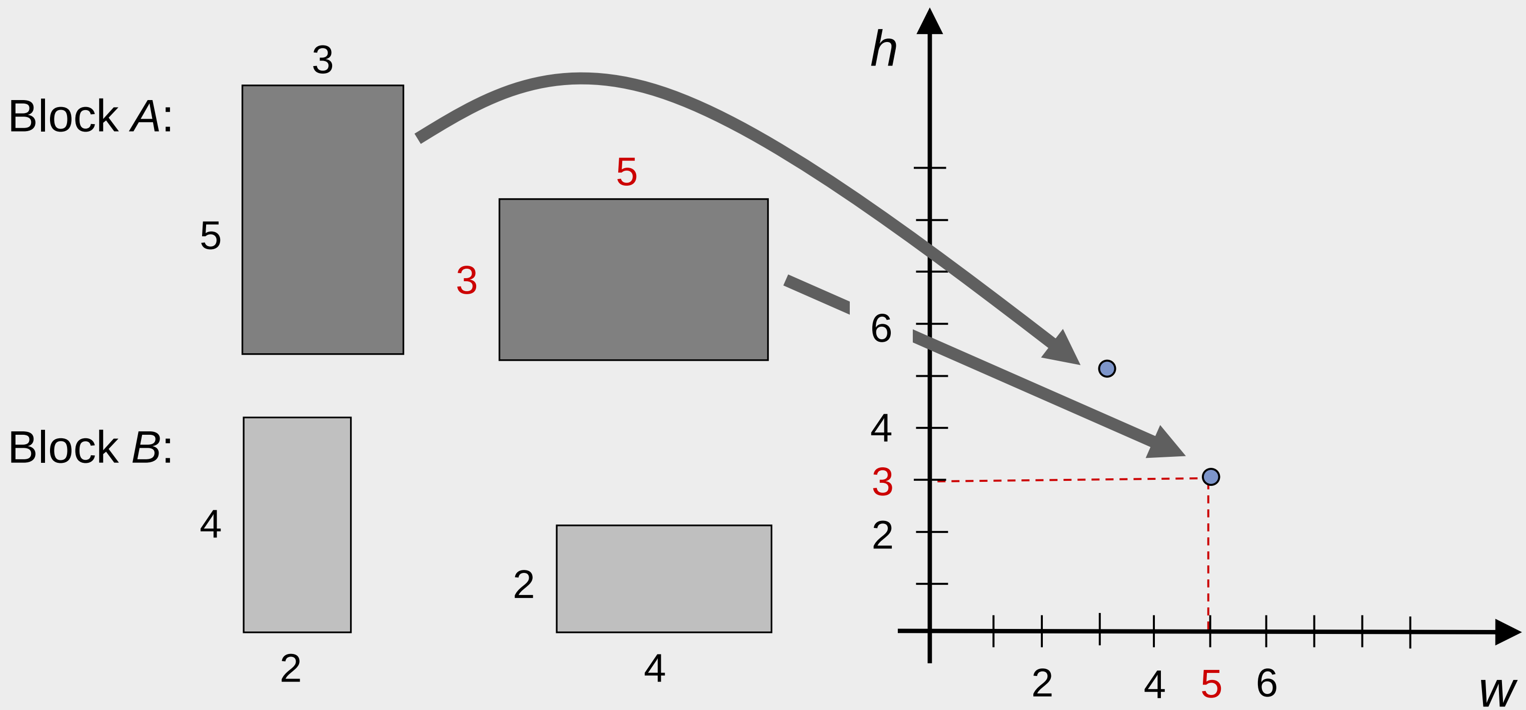


3.5.1 Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks

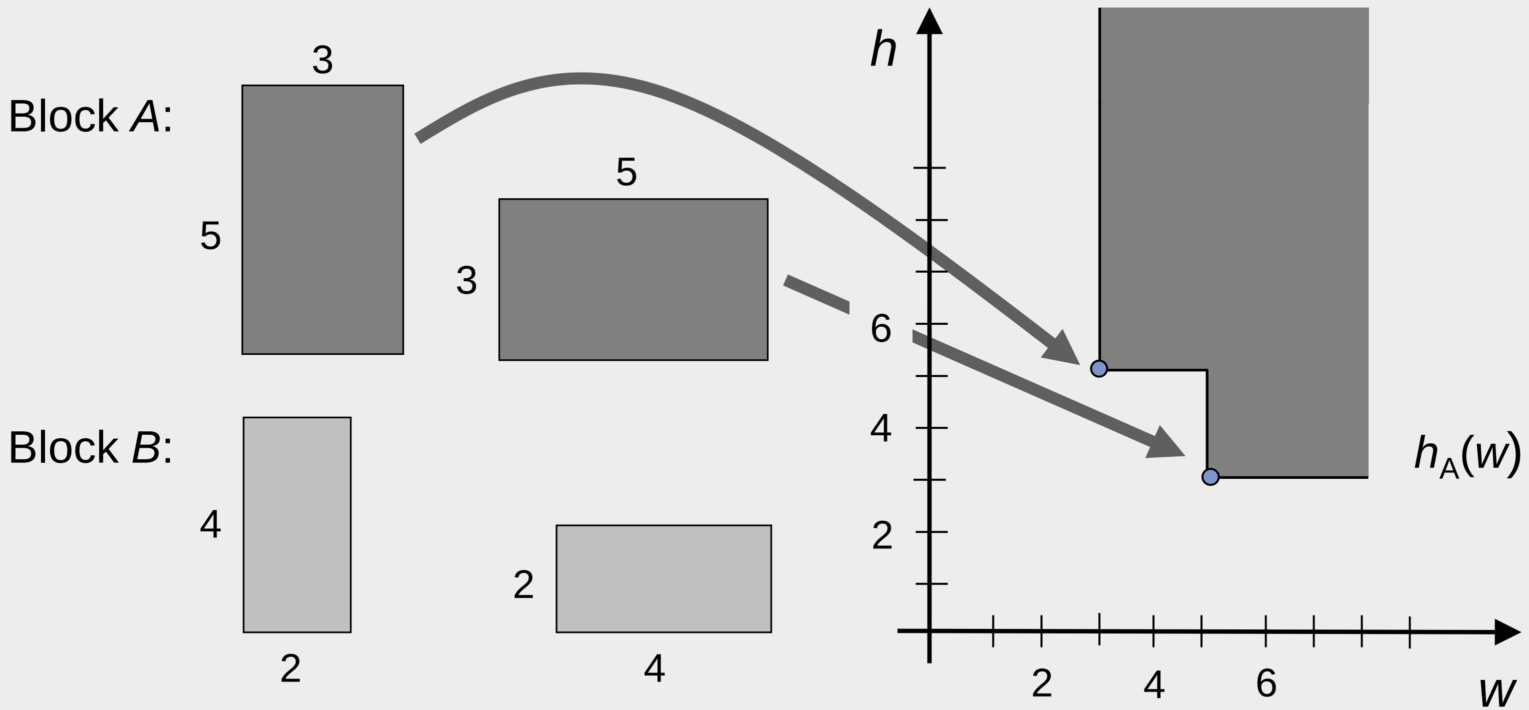


Step 1: Construct the shape functions of the blocks



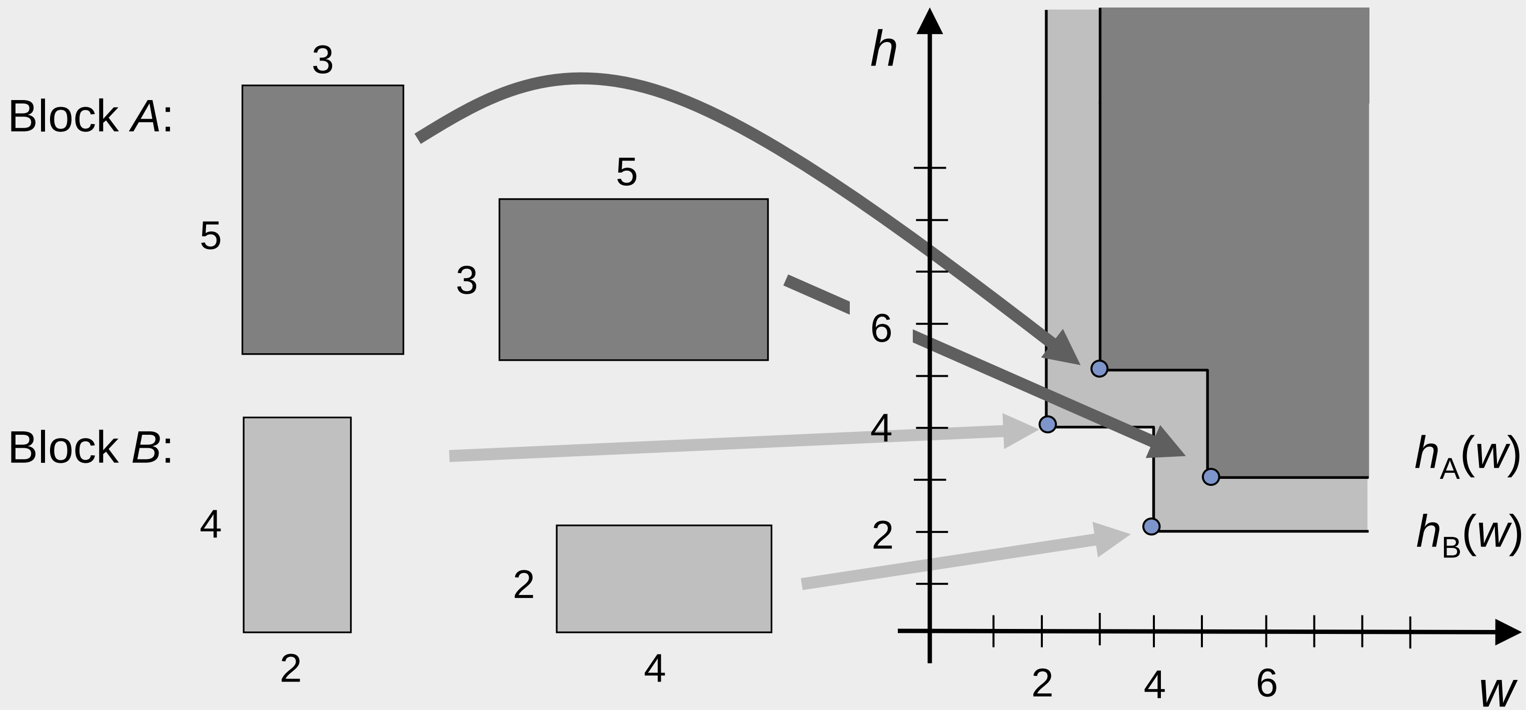
3.5.1 Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks



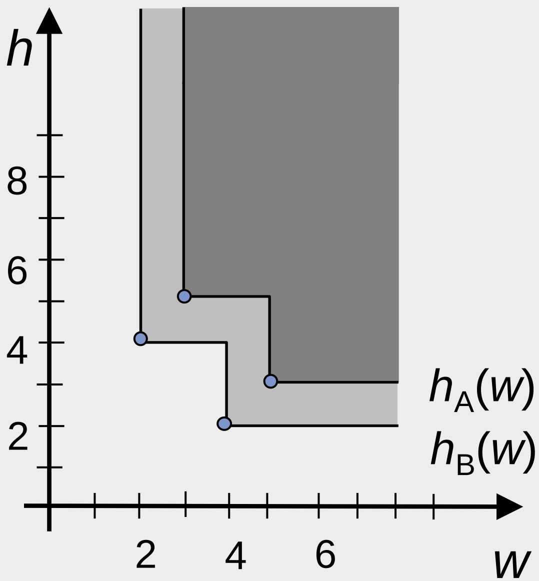
3.5.1 Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks

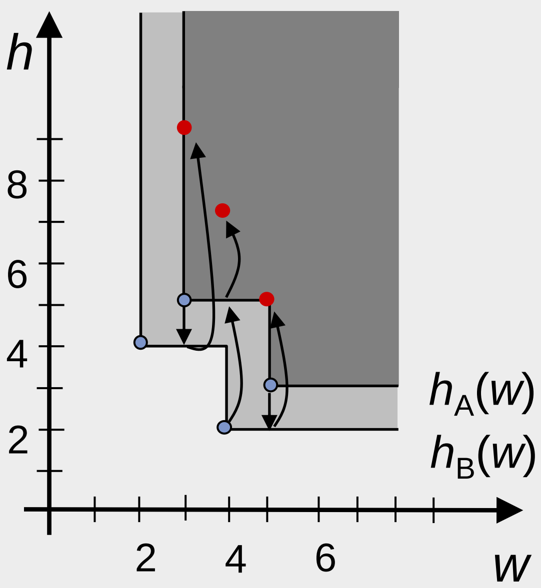


3.5.1 Floorplan Sizing – Example

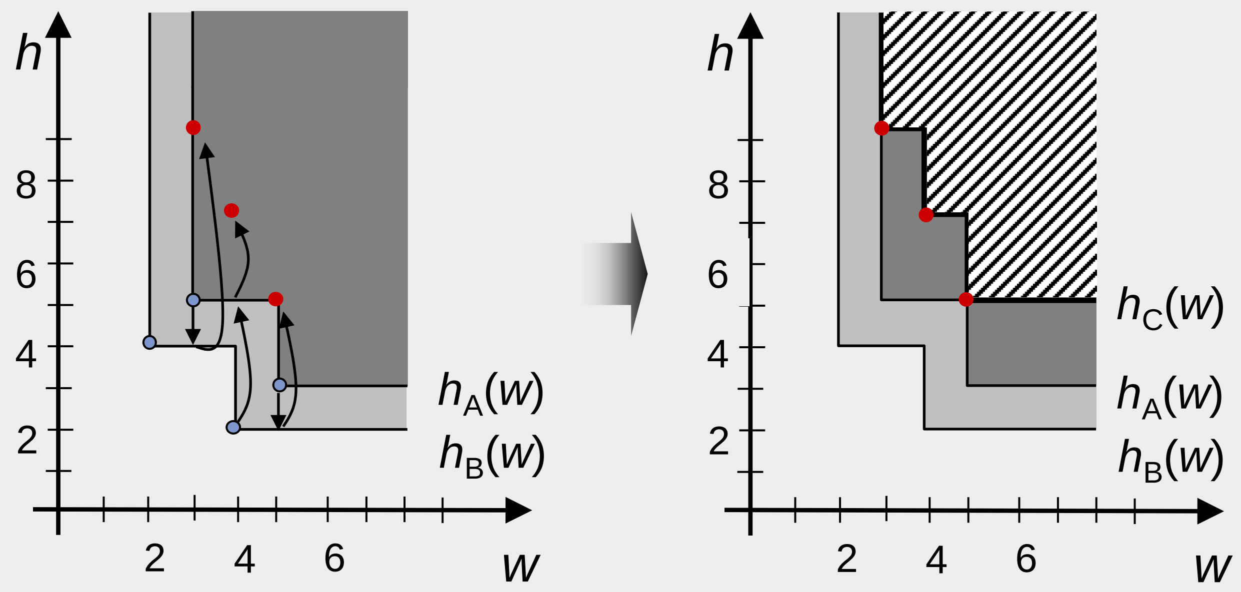
Step 2: Determine the shape function of the top-level floorplan (vertical)



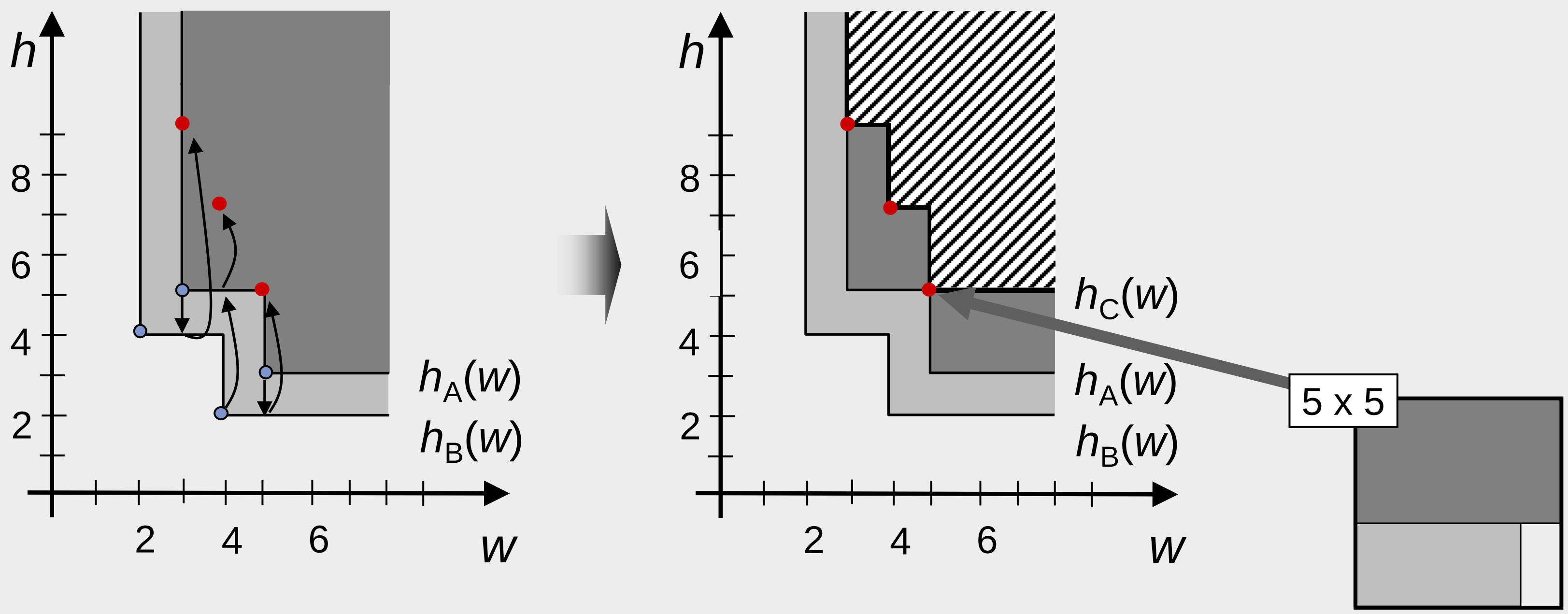
Step 2: Determine the shape function of the top-level floorplan (vertical)



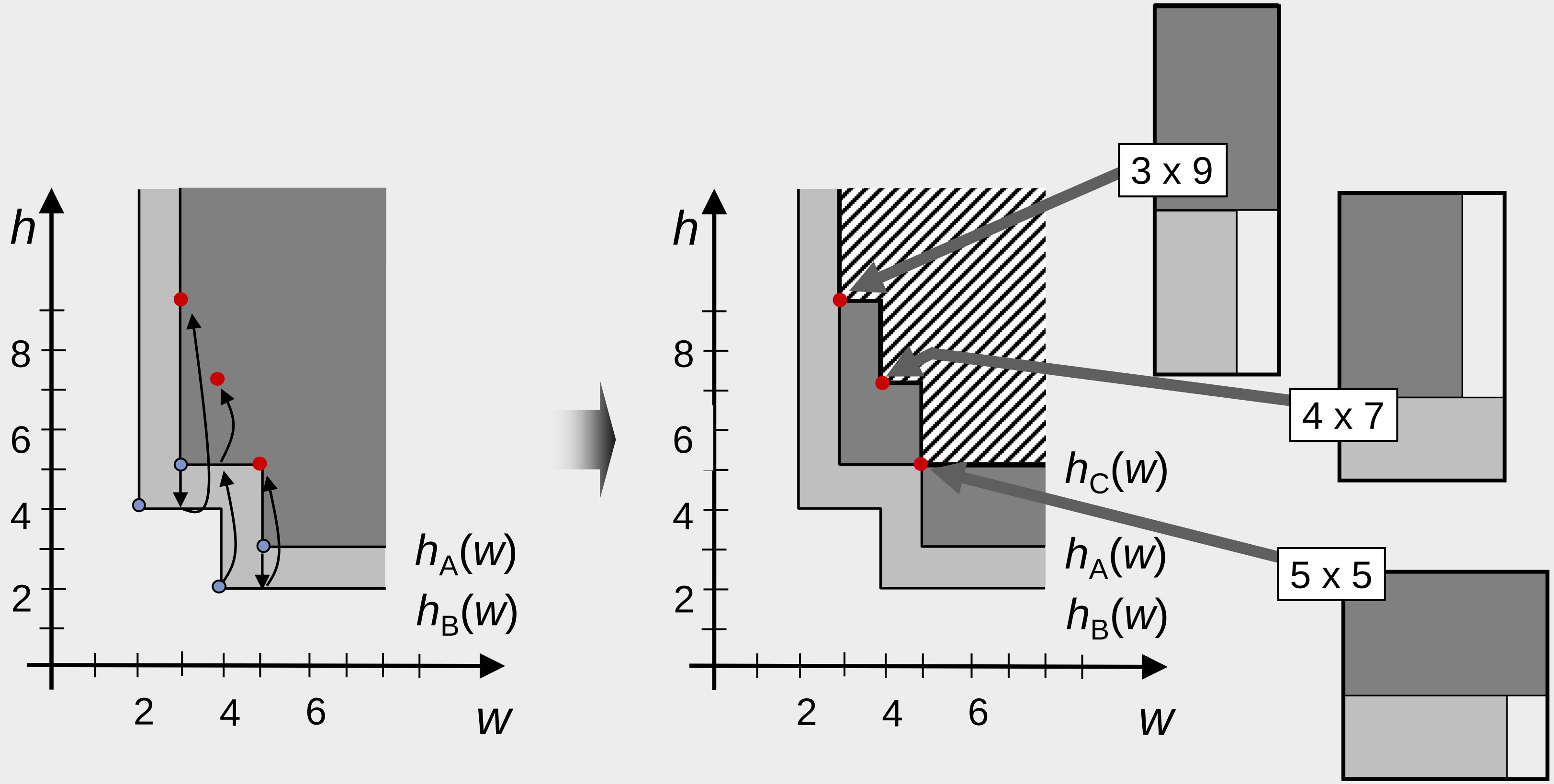
Step 2: Determine the shape function of the top-level floorplan (vertical)



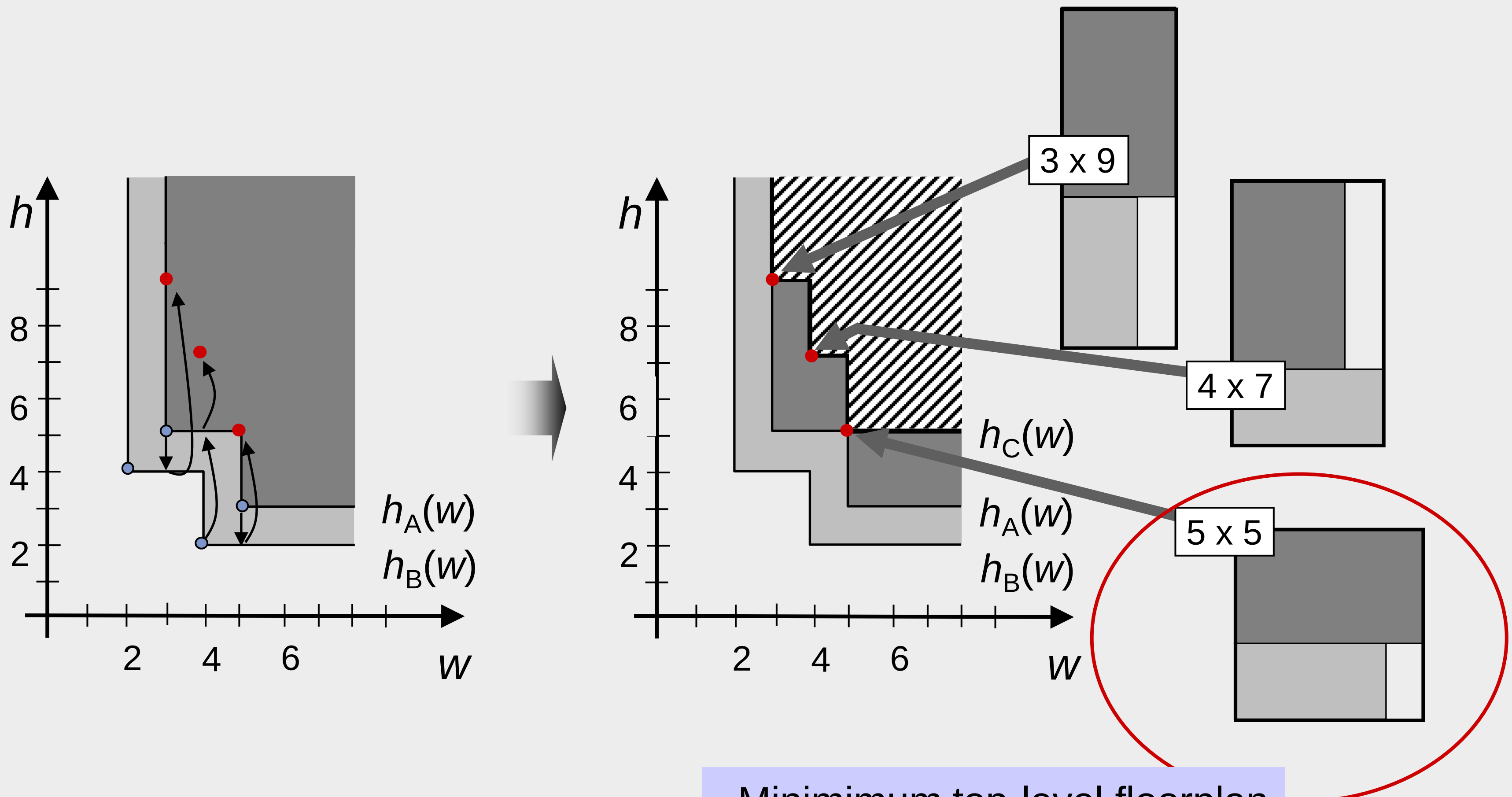
Step 2: Determine the shape function of the top-level floorplan (vertical)



Step 2: Determine the shape function of the top-level floorplan (vertical)

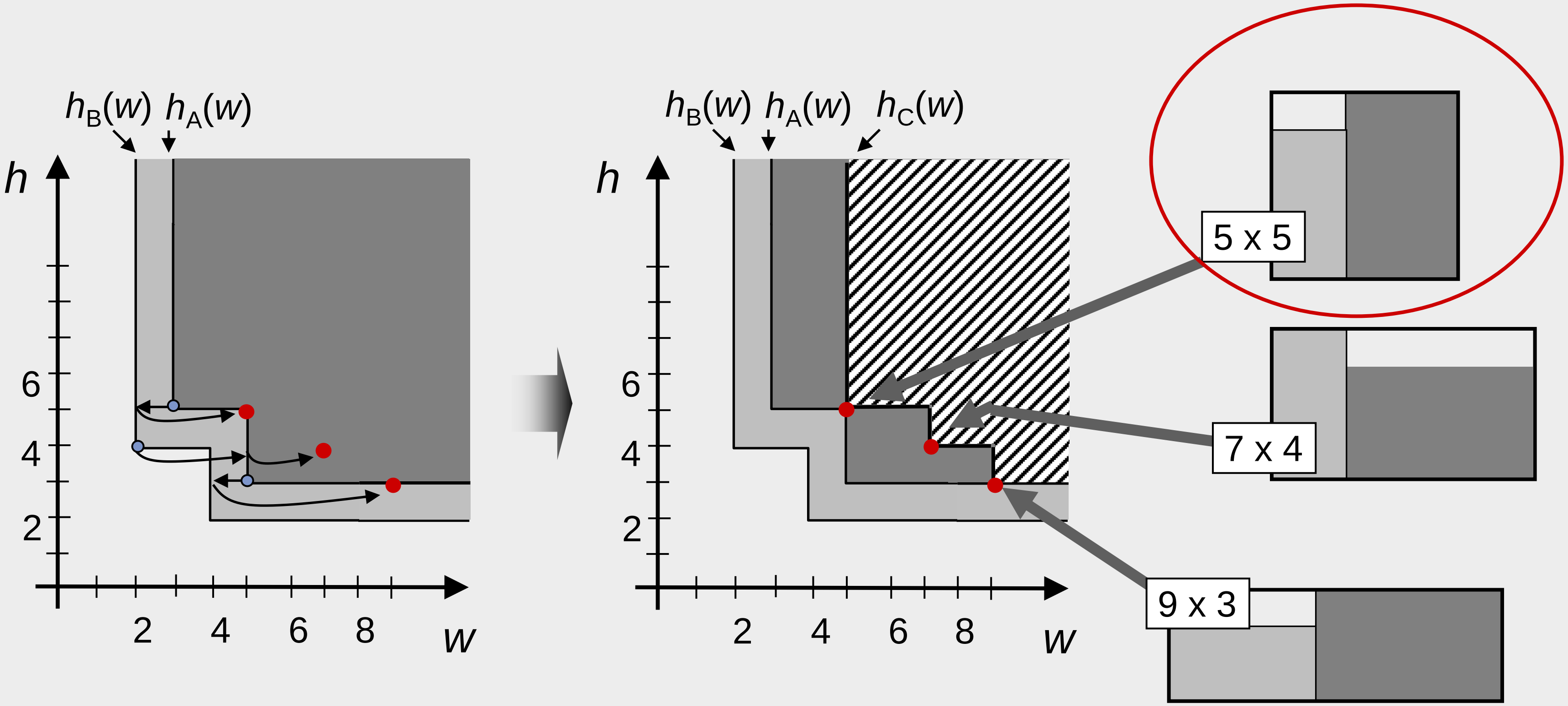


Step 2: Determine the shape function of the top-level floorplan (vertical)



Minimimum top-level floorplan with vertical composition

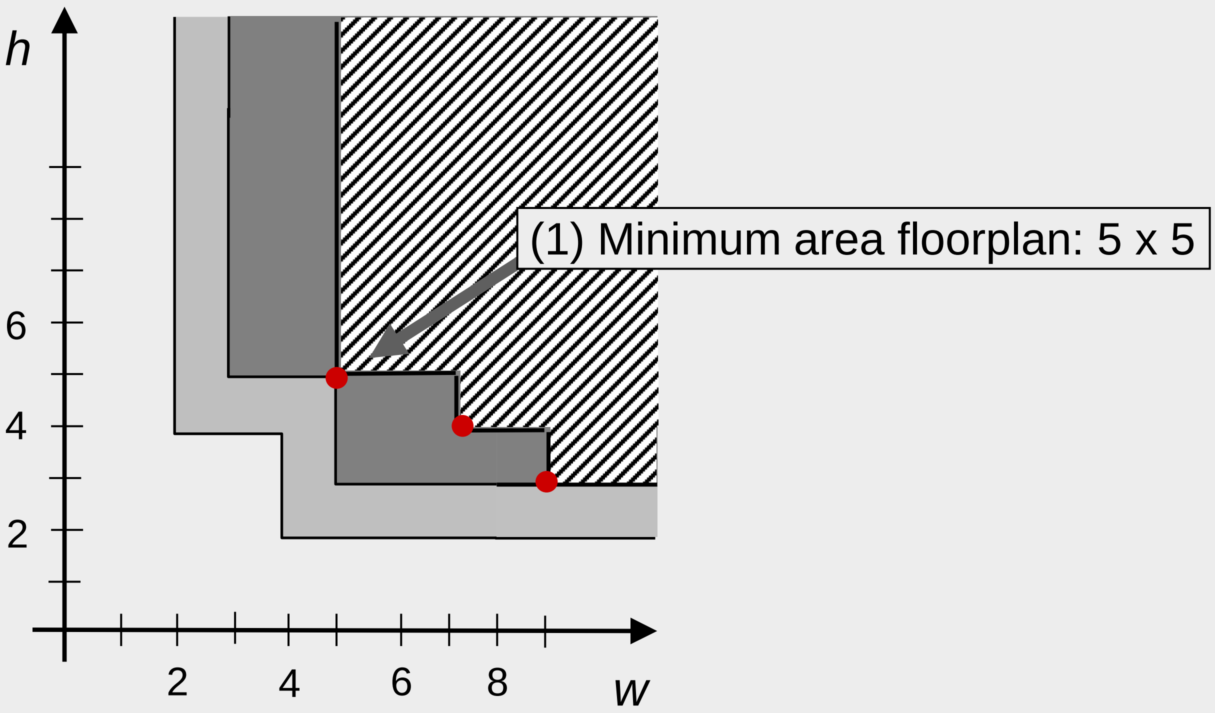
Step 2: Determine the shape function of the top-level floorplan (horizontal)



Minimimum top-level floorplan with horizontal composition

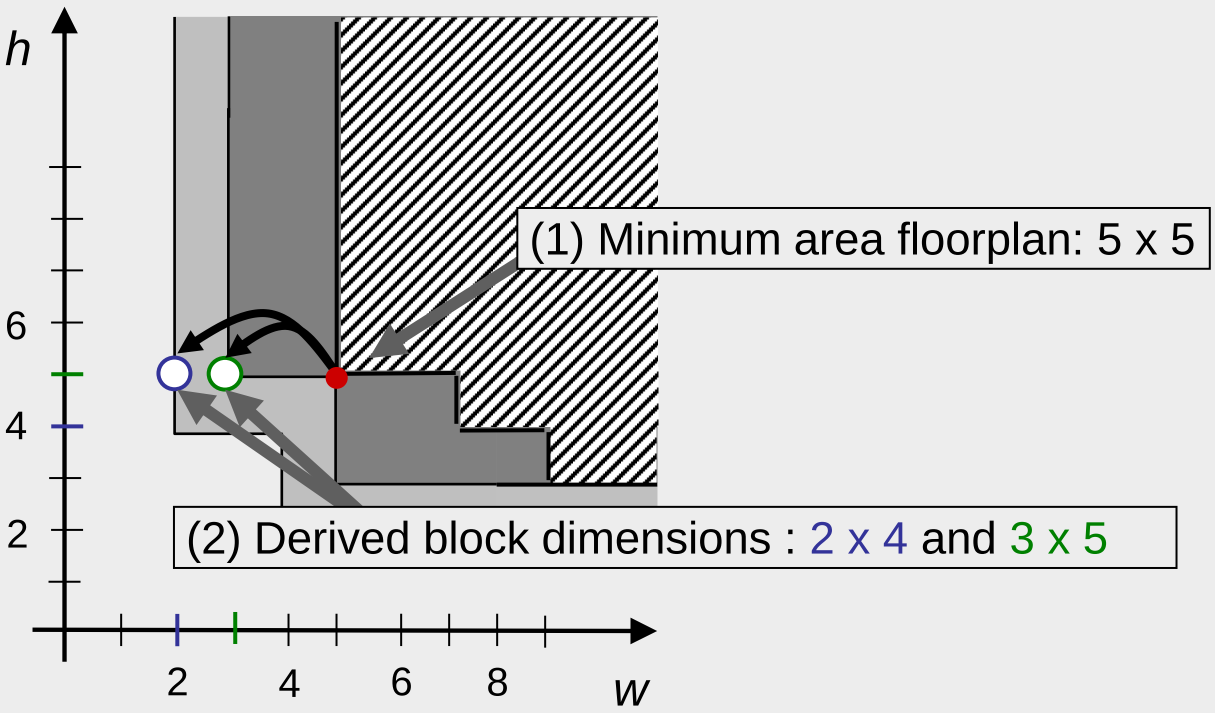
3.5.1 Floorplan Sizing – Example

Step 3: Find the individual blocks' dimensions and locations



Horizontal composition

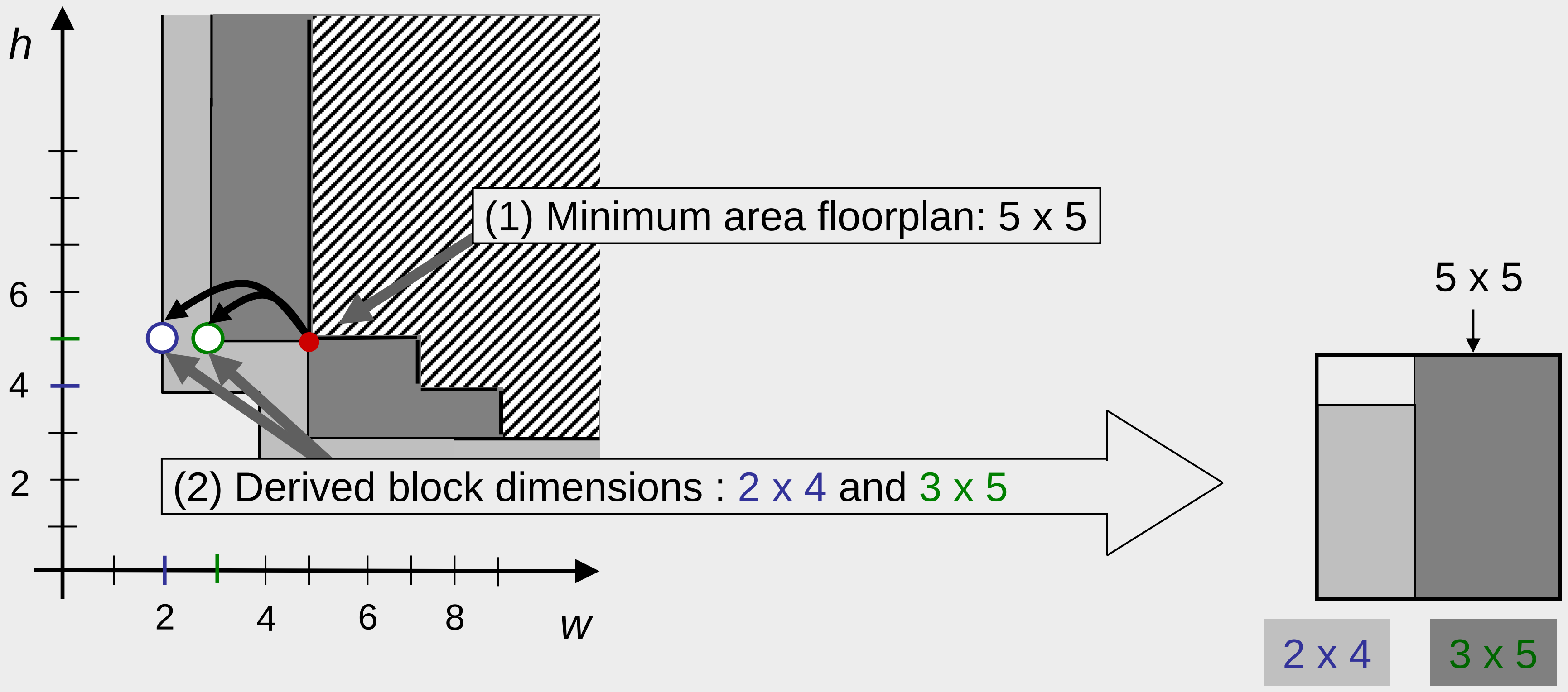
Step 3: Find the individual blocks' dimensions and locations



Horizontal composition

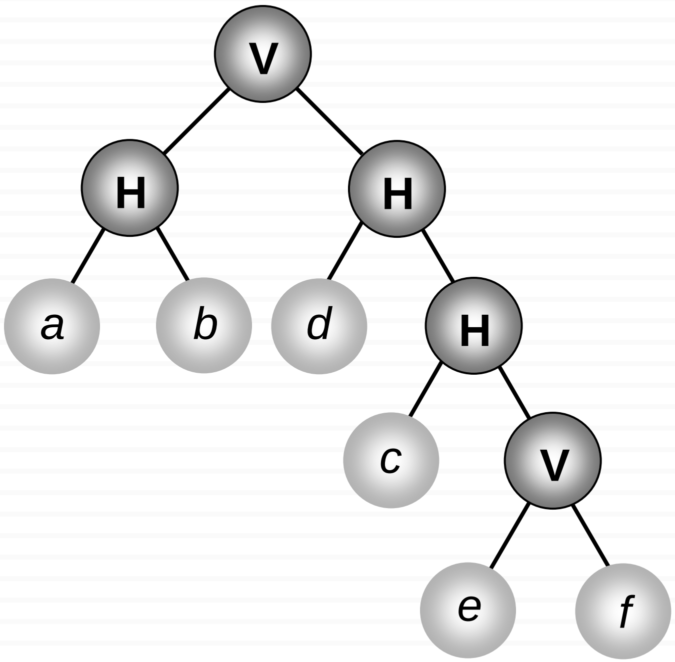
3.5.1 Floorplan Sizing – Example

Step 3: Find the individual blocks' dimensions and locations



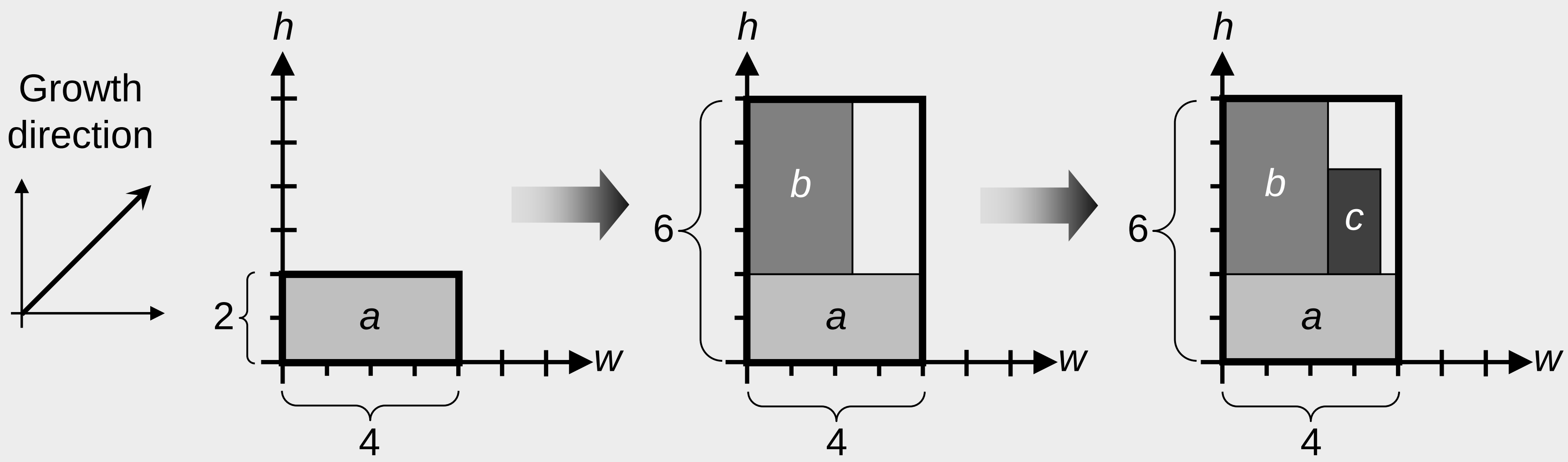
Horizontal composition

Floorplan Sizing



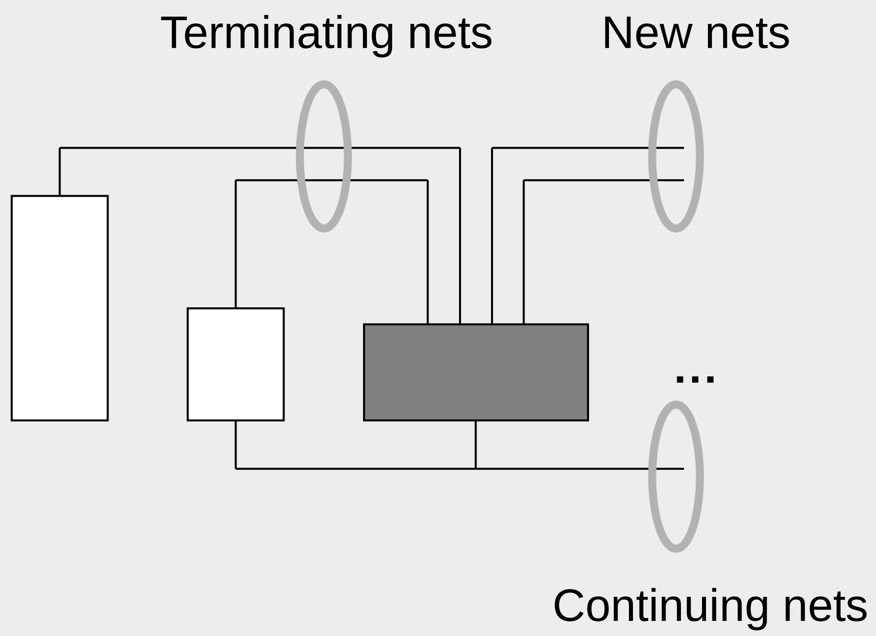
- Iteratively compose nodes in the tree bottom-up.
- At the root, choose the best solution.
- Backtrace the compositions

- Iteratively add blocks to the cluster until all blocks are assigned
- Only the different orientations of the blocks instead of the shape / aspect ratio are taken into account
- Linear ordering to minimize total wirelength of connections between blocks



3.5.2 Cluster Growth – Linear Ordering

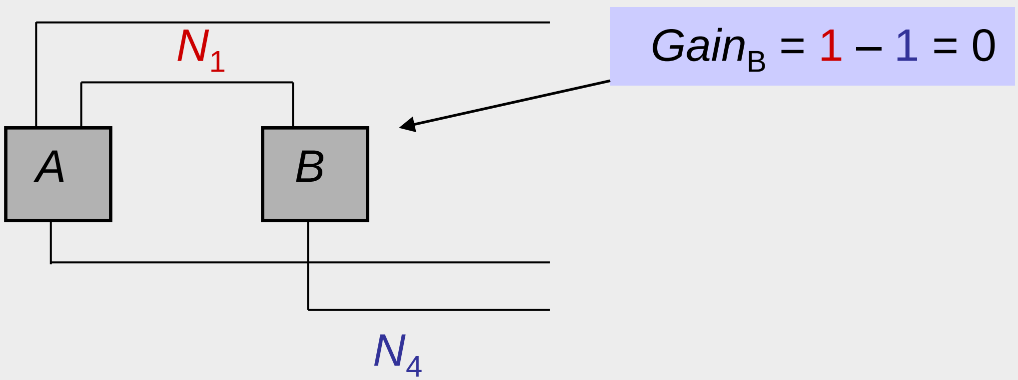
- New nets have no pins on any block from the partially-constructed ordering
- Terminating nets have no other incident blocks that are unplaced
- Continuing nets have at least one pin on a block from the partially-constructed ordering and at least one pin on an unordered block



3.5.2 Cluster Growth – Linear Ordering

- Gain of each block *m* is calculated:

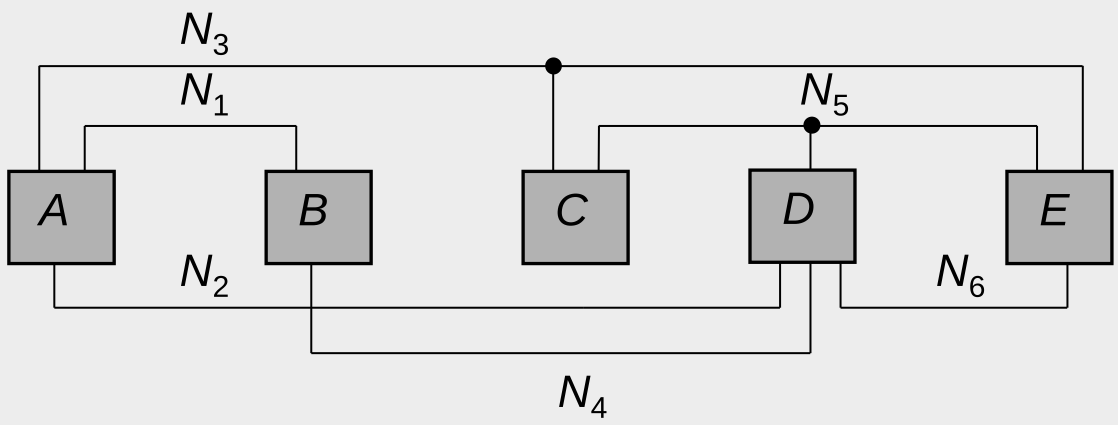
  $Gain_m$ = (Number of terminating nets of *m*) – (New nets of *m*)



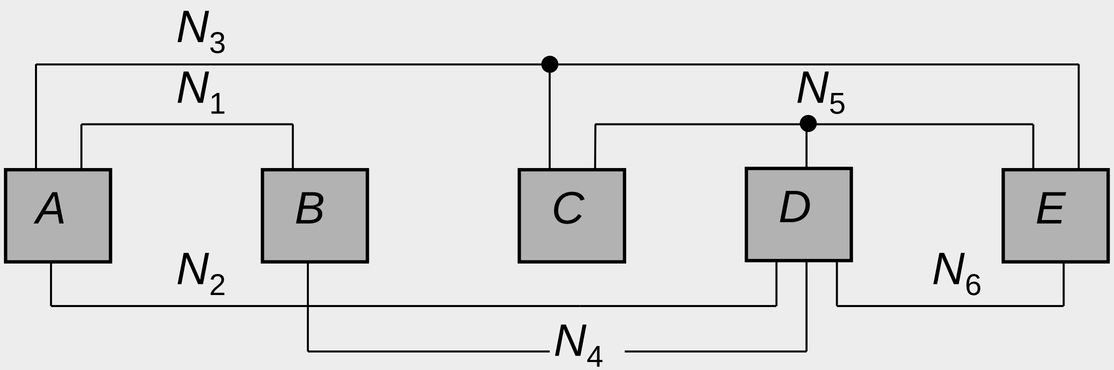
- The block with the maximum gain is selected to be placed next

Given:

– Netlist with five blocks *A, B, C, D, E* and six nets
  $N_1 = \{A, B\}$
  $N_2 = \{A, D\}$
  $N_3 = \{A, C, E\}$
  $N_4 = \{B, D\}$
  $N_5 = \{C, D, E\}$
  $N_6 = \{D, E\}$
– Initial block: *A*



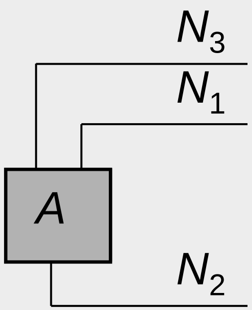
Task: Linear ordering with minimum netlength

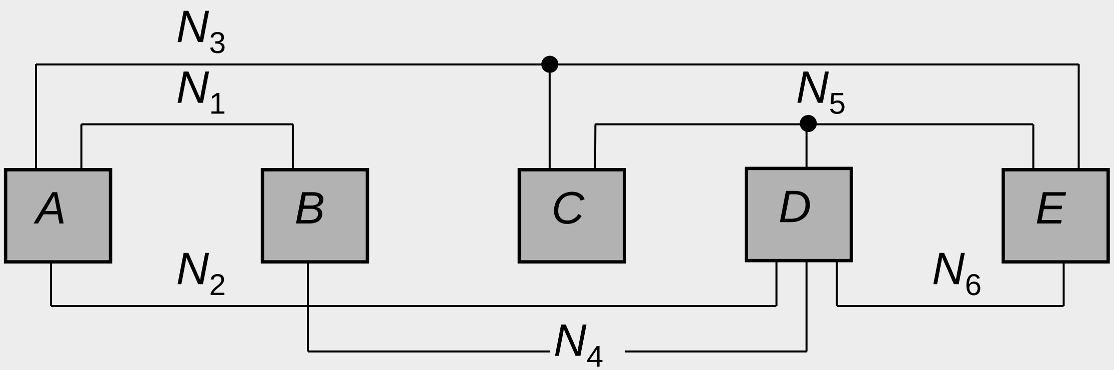


Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	***A***	N_1,N_2,N_3	--	-3	--

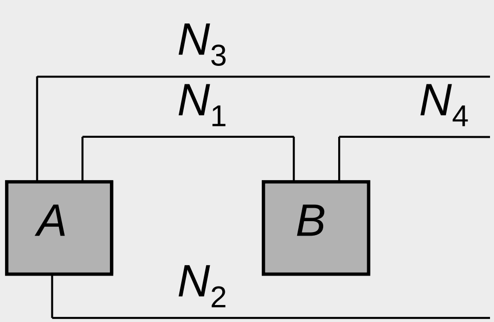
Initial block

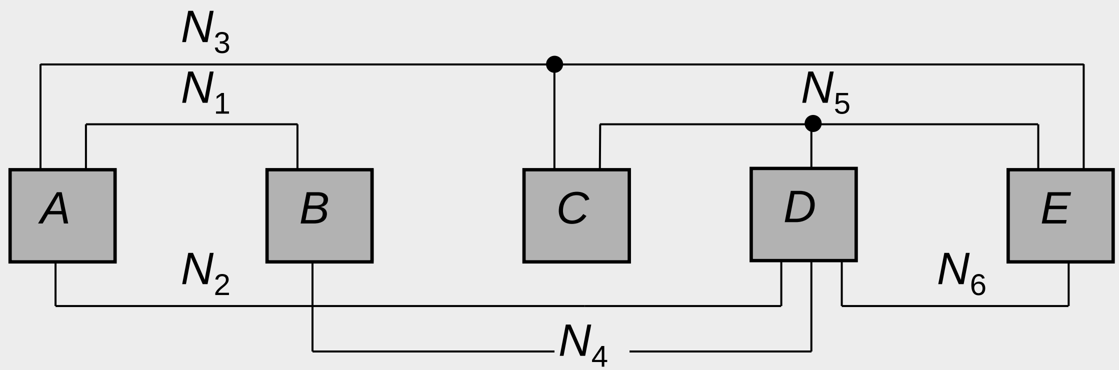
$Gain_A$ = (Number of terminating nets of *A*) – (New nets of *A*)



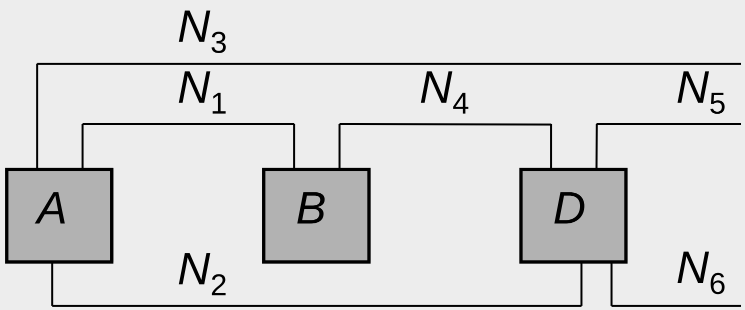


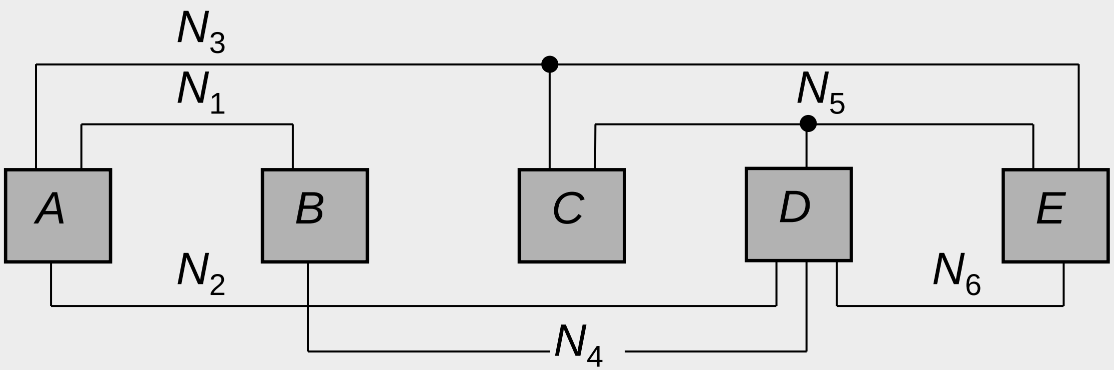
Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	***A***	N_1,N_2,N_3	--	-3	--
1	*B*	N_4	N_1	0	--
	C	N_5	--	-1	N_3
	D	N_4,N_5,N_6	N_2	-2	--
	E	N_5,N_6	--	-2	N_3





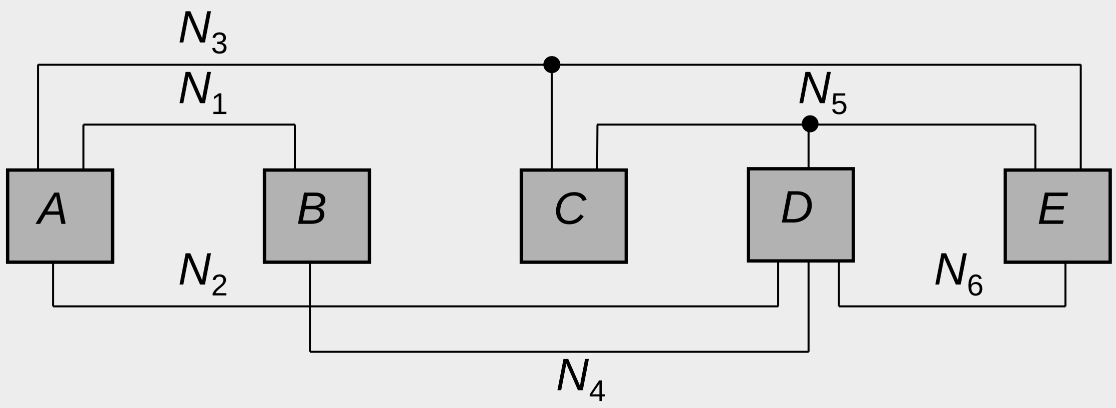
Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	***A***	N_1,N_2,N_3	--	-3	--
1	***B***	N_4	N_1	0	--
	C	N_5	--	-1	N_3
	D	N_4,N_5,N_6	N_2	-2	--
	E	N_5,N_6	--	-2	N_3
2	*C*	N_5	--	-1	N_3
	D	N_5,N_6	N_2,N_4	0	--
	E	N_5,N_6	--	-2	N_3

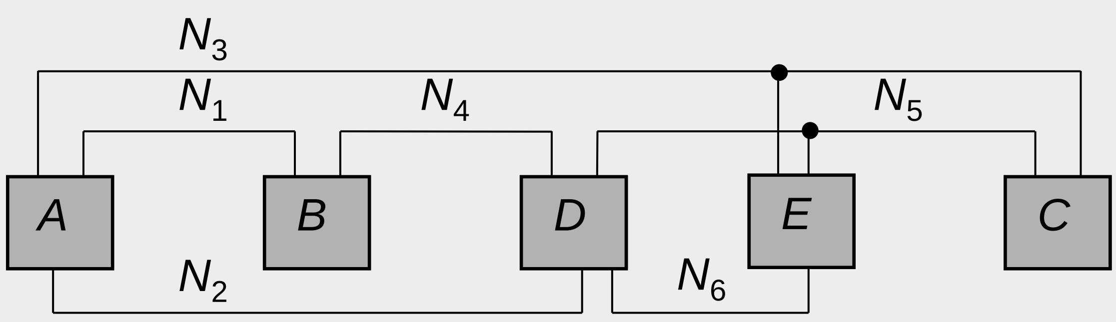




Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	***A***	N_1,N_2,N_3	--	-3	--
1	***B***	N_4	N_1	0	--
	C	N_5	--	-1	N_3
	D	N_4,N_5,N_6	N_2	-2	--
	E	N_5,N_6	--	-2	N_3
2	*C*	N_5	--	-1	N_3
	D	N_5,N_6	N_2,N_4	0	--
	E	N_5,N_6	--	-2	N_3
3	*C*	--	--	0	N_3,N_5
	E	--	N_6	1	N_3,N_5
4	***C***	--	N_3,N_5	2	--

3.5.2 Cluster Growth – Linear Ordering (Example)





Input: set of all blocks *M*, cost function *C*

Output: optimized floorplan *F* based on *C*

```
F = Ø
order = LINEAR_ORDERING(M)                  // generate linear ordering
for (i = 1 to |order|)
    curr_block = order[i]
    ADD_TO_FLOORPLAN(F,curr_block,C)        // find location and orientation
                                            // of curr_block that causes
                                            // smallest increase based on
                                            // C while obeying constraints
```

3.5.2 Cluster Growth

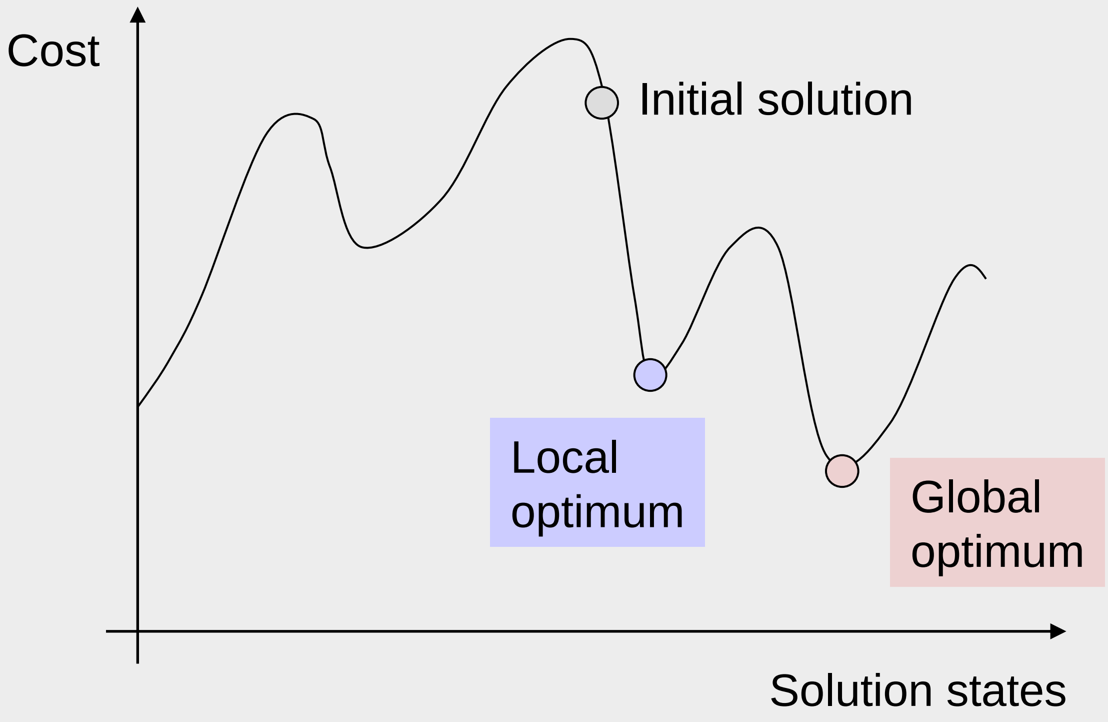
Analysis

- The objective is to minimize the total wirelength of connections blocks
- Though this produces mediocre solutions, the algorithm is easy to implement and fast.
- Can be used to find the initial floorplan solutions for iterative algorithms such as *simulated annealing*.

Introduction

- Simulated Annealing (SA) algorithms are iterative in nature.
- Begins with an initial (arbitrary) solution and seeks to incrementally improve the objective function.
- During each iteration, a local neighborhood of the current solution is considered. A new candidate solution is formed by a small perturbation of the current solution.
- Unlike greedy algorithms, SA algorithms can accept candidate solutions with higher cost.

3.5.3 Simulated Annealing



3.5.3 Simulated Annealing

What is annealing?

- Definition (from material science): controlled cooling process of high-temperature materials to modify their properties.
- Cooling changes material structure from being highly randomized (chaotic) to being structured (stable).
- The way that atoms settle in low-temperature state is probabilistic in nature.

- Slower cooling has a higher probability of achieving a perfect lattice with minimum-energy
  - Cooling process occurs in steps
  - Atoms need enough time to try different structures
  - Sometimes, atoms may move across larger distances and create (intermediate) higher-energy states
  - Probability of the accepting higher-energy states decreases with temperature

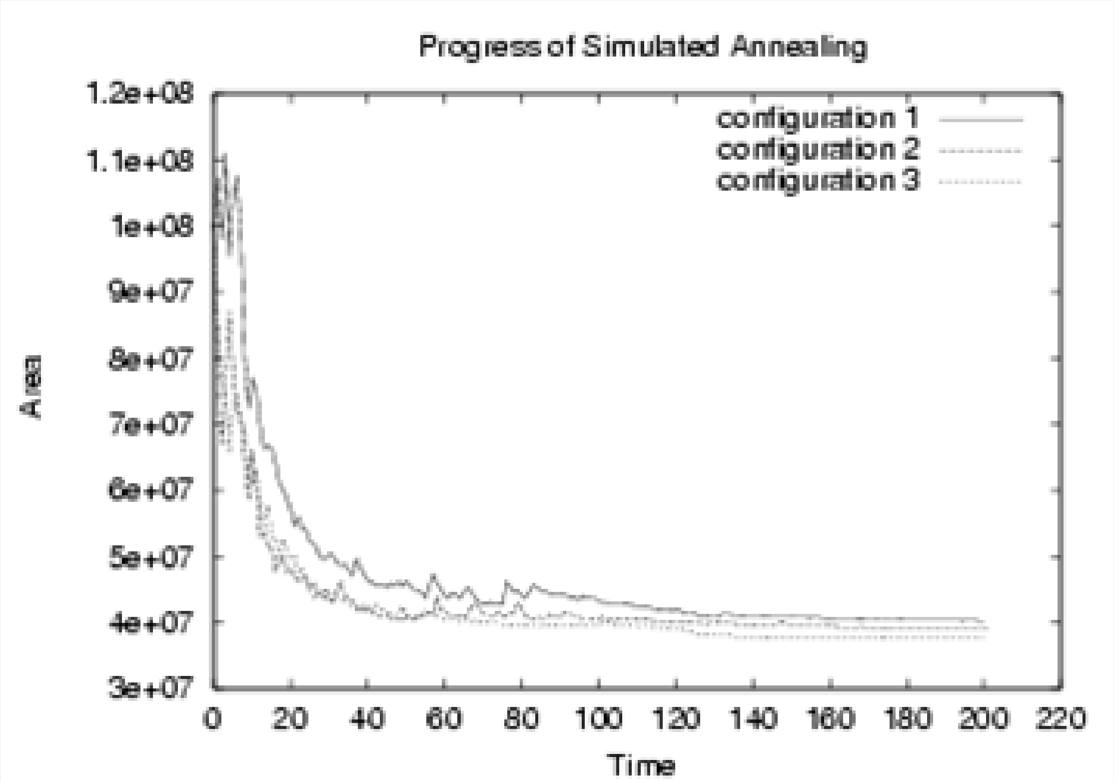
Simulated Annealing

- Generate an initial solution S_{init}, and evaluate its cost.
- Generate a new solution S_{new} by performing a random walk
- S_{new} is accepted or rejected based on the temperature T
  - Higher T means a higher probability to accept S_{new} if $COST(S_{new}) > COST(S_{init})$
  - T slowly decreases to form the final solution

- Boltzmann acceptance criterion, where r is a random number [0,1)

$$e^{\frac{COST(S_{curr})-COST(S_{new})}{T}} > r$$

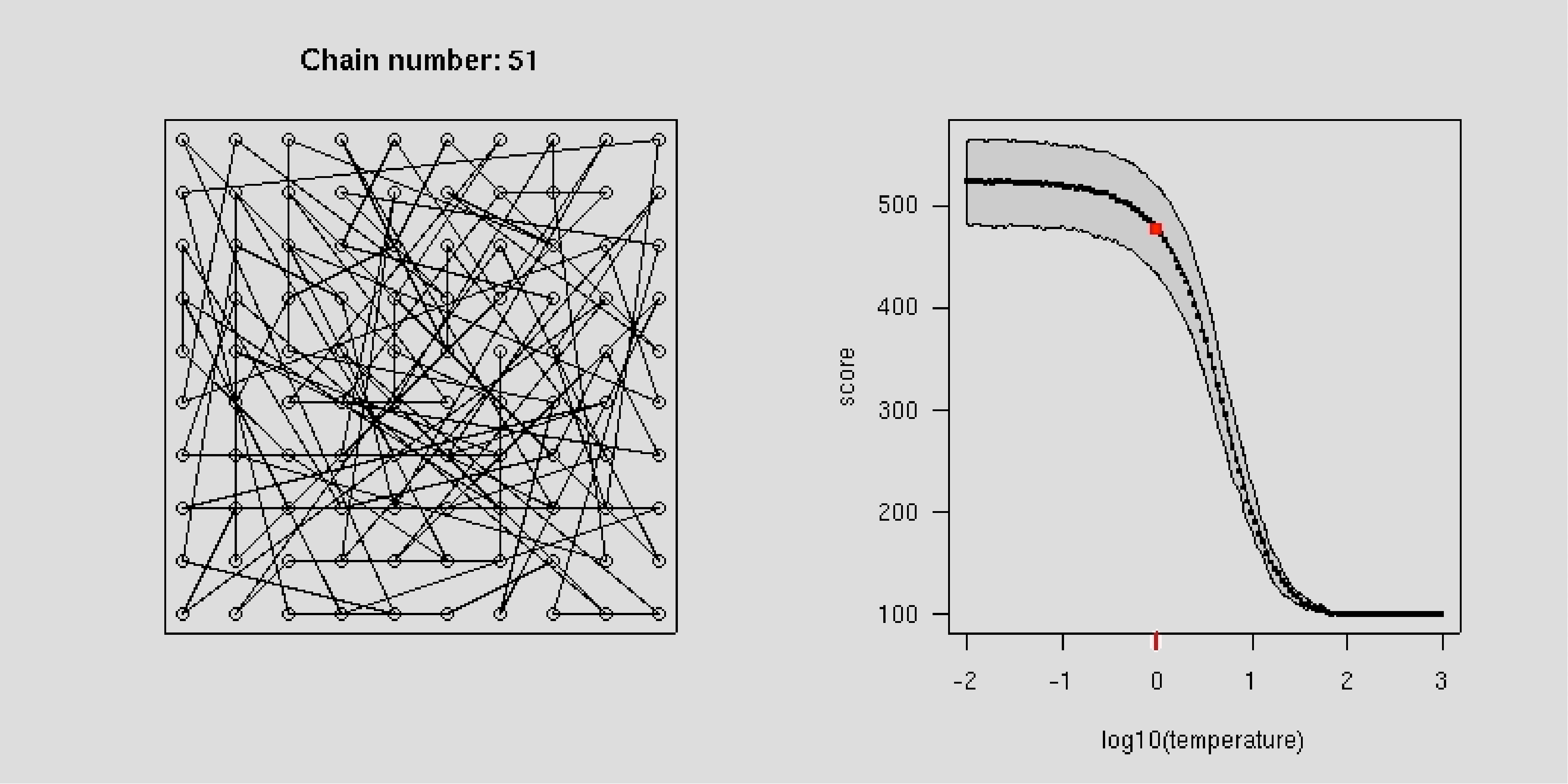
3.5.3 Simulated Annealing – Algorithm



Input: initial solution *init_sol*
Output: optimized new solution *curr_sol*

```
T = T_0                                        // initialization
i = 0
curr_sol = init_sol
curr_cost = COST(curr_sol)
while (T > T_min)
    while (stopping criterion is not met)
     i = i + 1
     (a_i,b_i) = SELECT_PAIR(curr_sol)         // select two objects to perturb
     trial_sol = TRY_MOVE(a_i,b_i)             // try small local change
     trial_cost = COST(trial_sol)
     Δcost = trial_cost – curr_cost
     if (Δcost < 0)                            // if there is improvement,
               curr_cost = trial_cost          //  update the cost and
               curr_sol = MOVE(a_i,b_i)        //  execute the move
     else
               r = RANDOM(0,1)                 // random number [0,1]
               if (r < e^(–Δcost/T))           // if it meets threshold,
                   curr_cost = trial_cost      //   update the cost and
                   curr_sol = MOVE(a_i,b_i)    //   execute the move
     T = α · T                                 // 0 < α < 1, T reduction
```

Simulated Annealing – Animation



Source: http://www.biostat.jhsph.edu/~iruczins/teaching/misc/annealing/animation.html

Simulated Annealing - Notes

- Practical tuning needed for good results:
  - How to choose the T values and how to update it?
  - Should we spend more iterations with high T or low T?
    - High T: More non-greedy moves accepted
    - Low T: Accepts mostly greedy moves, but can get stuck
    - Quality of initial solution should also be considered

Simulated Annealing - Notes

- For floorplanning:
  - Definition of move depends on the representation used
    - e.g. Polish expression, sequence pair, etc.
  - Cost evaluation of a move may involve:
    - packing (e.g. based on horizontal/vertical constraints)
    - block sizing
    - wirelength estimation