《芯片设计自动化与智能优化》
Floorplanning

The slides are based on Prof. David Z. Pan’s lecture notes at UT Austin

Yibo Lin

Peking University

Outline

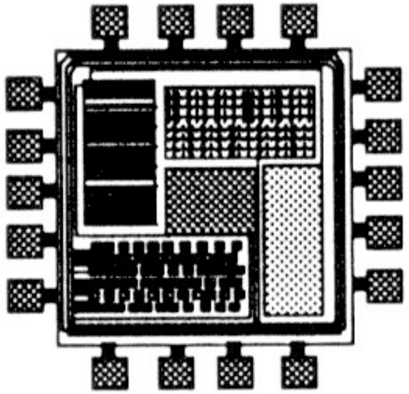
- What is floorplanning
- Searching based algorithms
  - Simulated annealing
  - Evolution algorithm
- Floorplanning representation
  - Polish expression
  - Sequence pair
- Integer linear programming

What is Floorplanning

- Room sketches

Hierarchical Design

- Several blocks after partitioning:

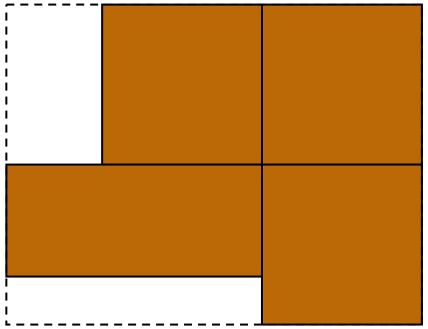


- Need to:
  - Put the blocks together.
  - Design each block.

  Which step to go first?

Hierarchical Design

- 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 the positions and shapes of the modules at the beginning of the design cycle to optimize the circuit performance:
  - 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 $A_1, \ldots, A_n$
  - 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 the circuit performance.

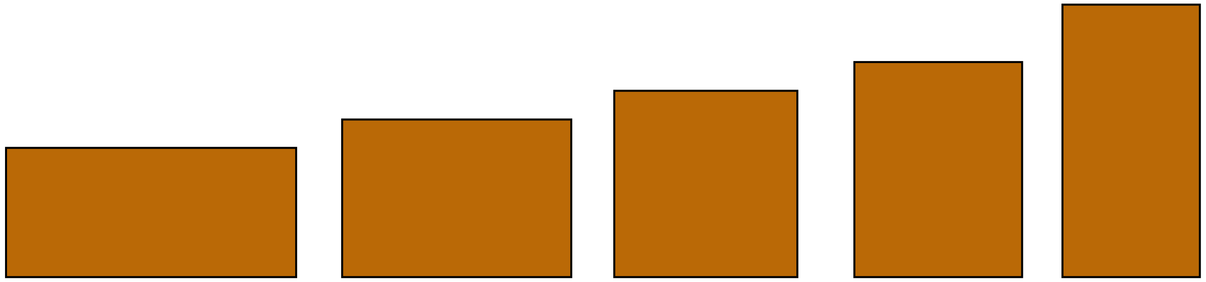
Bounds on Aspect Ratios

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

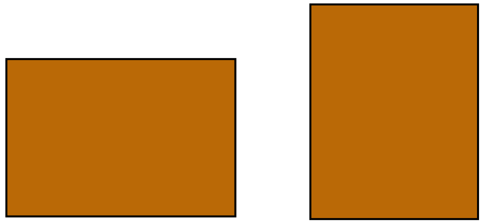
- But we don't want to layout blocks as long strips, so we require $r_i \le \frac{h_i}{w_i} \le s_i$ for each i.

Bounds on Aspect Ratios

- We can also allow several shapes for each block:



- For hard blocks, the orientations can be changed:

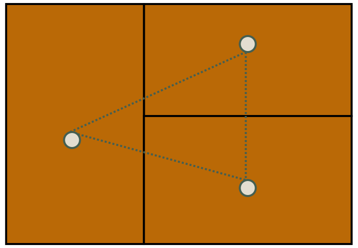


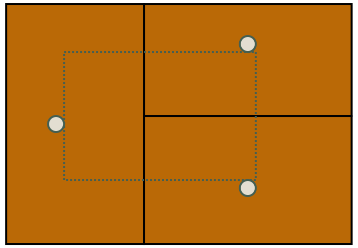
Objective Function

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

Wirelength Estimation

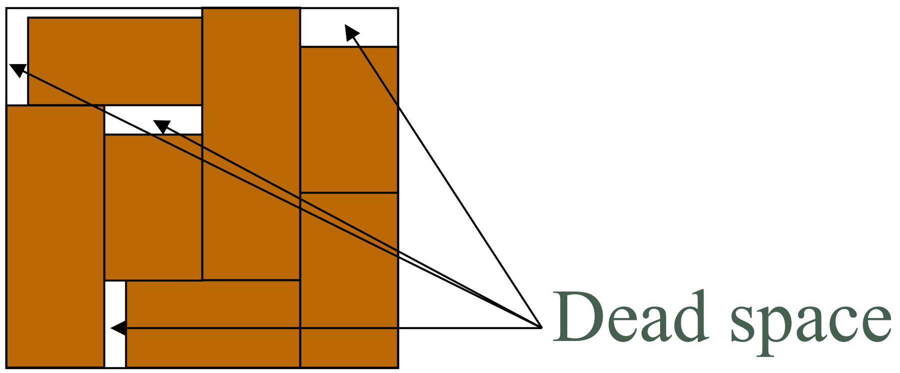
- Exact wirelength of each net is not known until routing is done.
- In floorplanning, even pin positions are not known yet.
- Some possible wirelength estimations:
  - Center-to-center estimation
  - Half-perimeter estimation





Dead space

- Dead space is the space that is wasted:



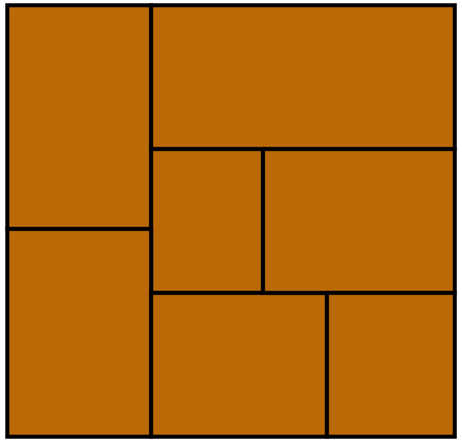
- Minimizing area is the same as minimizing deadspace.
- Dead space percentage is computed as

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

Slicing and Non-Slicing Floorplan

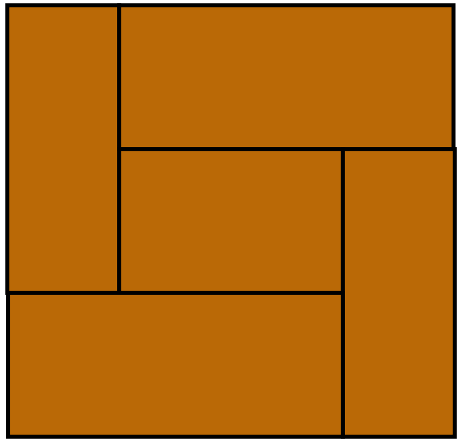
- Slicing Floorplan:

  One that can be obtained by repetitively subdividing (slicing) rectangles horizontally or vertically.



- Non-Slicing Floorplan:

  One that may not be obtained by repetitively subdividing alone.



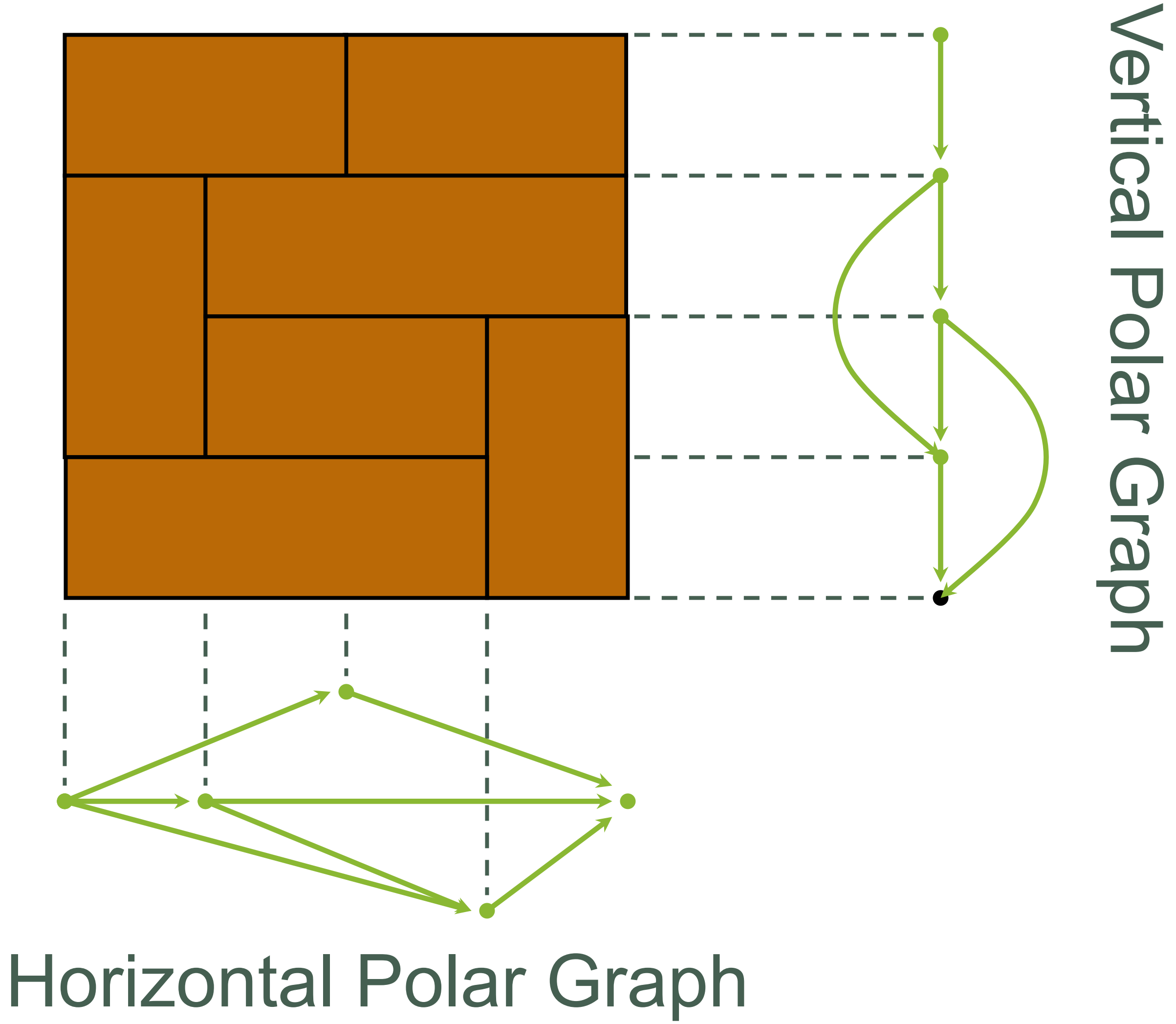
- Otten (LSSS-82) pointed out that slicing floorplans are much easier to handle.

Polar Graph Representation

- A graph representation of floorplan.
- Each floorplan is modeled by a pair of directed acyclic graphs:
  - Horizontal polar graph
  - Vertical polar graph
- For horizontal (vertical) polar graph,
  - Vertex: Vertical (horizontal) channel
  - Edge: 2 channels are on 2 sides of a block
  - Edge weight: Width (height) of the block

Note: There are many other graph representations.

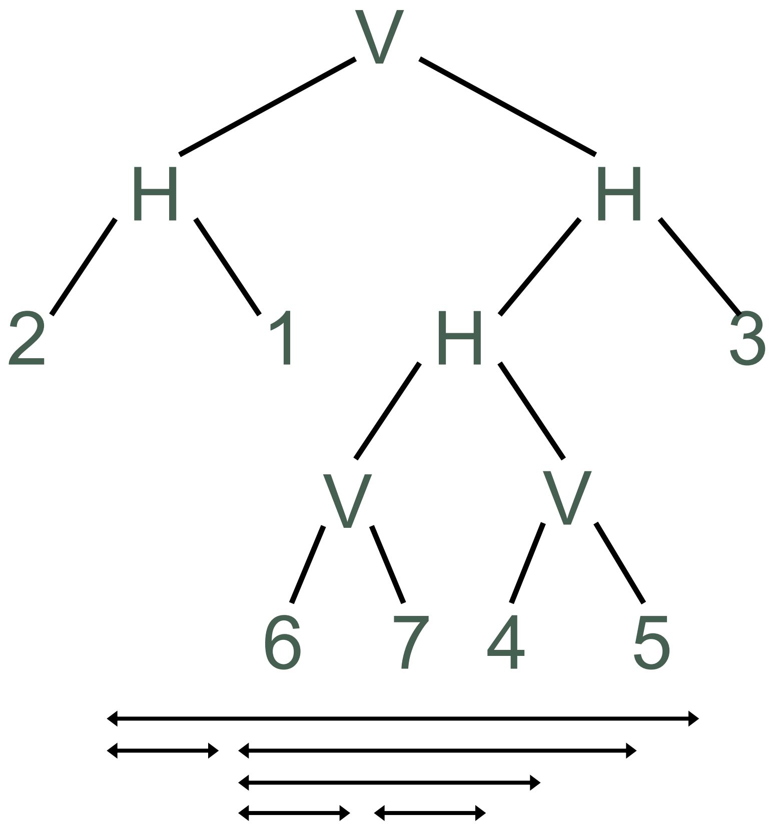
Polar Graph: Example



Simulated Annealing using Polish Expression Representation

- Representation of slicing floorplan

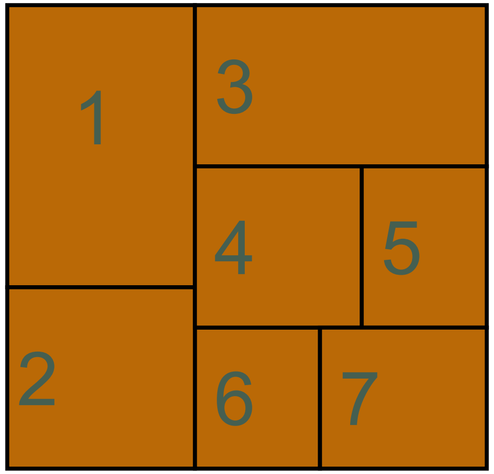
Slicing Tree



Polish Expression
(postorder traversal of slicing tree)

21H67V45VH3HV

Slicing Floorplan



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

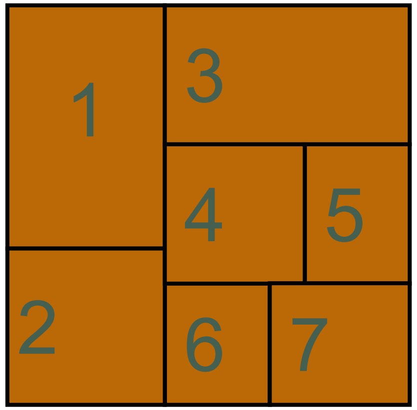
Polish Expression

- 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.

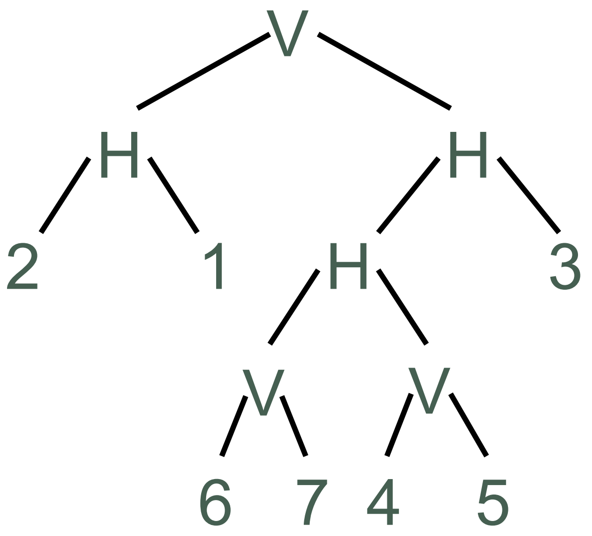
Skewed ST and Normalized PE

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

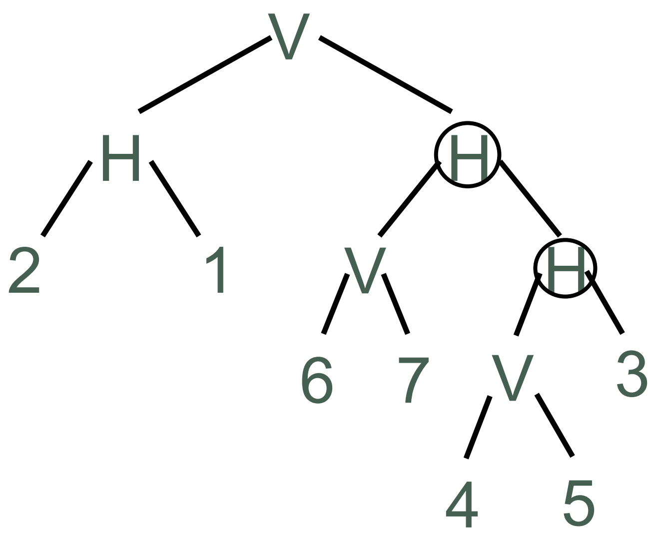
Slicing Floorplan



Slicing Tree (Skewed)



Slicing Tree



Polish Expression

21H67V45VH3HV	21H67V45V3HHV

Normalized Polish Expression

- There is a 1-1 correspondence between Slicing Floorplan, Skewed Slicing Tree, and Normalized Polish Expression.
- Will use Normalized Polish Expression to represent slicing floorplans.
  - What is a valid NPE?
- Can be formulated as a state space search problem.

Neighborhood Structure

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

$\boxed{\text{16H35V2HV74HV}}$

Chains

- The moves:

  M1: Swap adjacent operands (ignoring chains)

  M2: Complement some chain

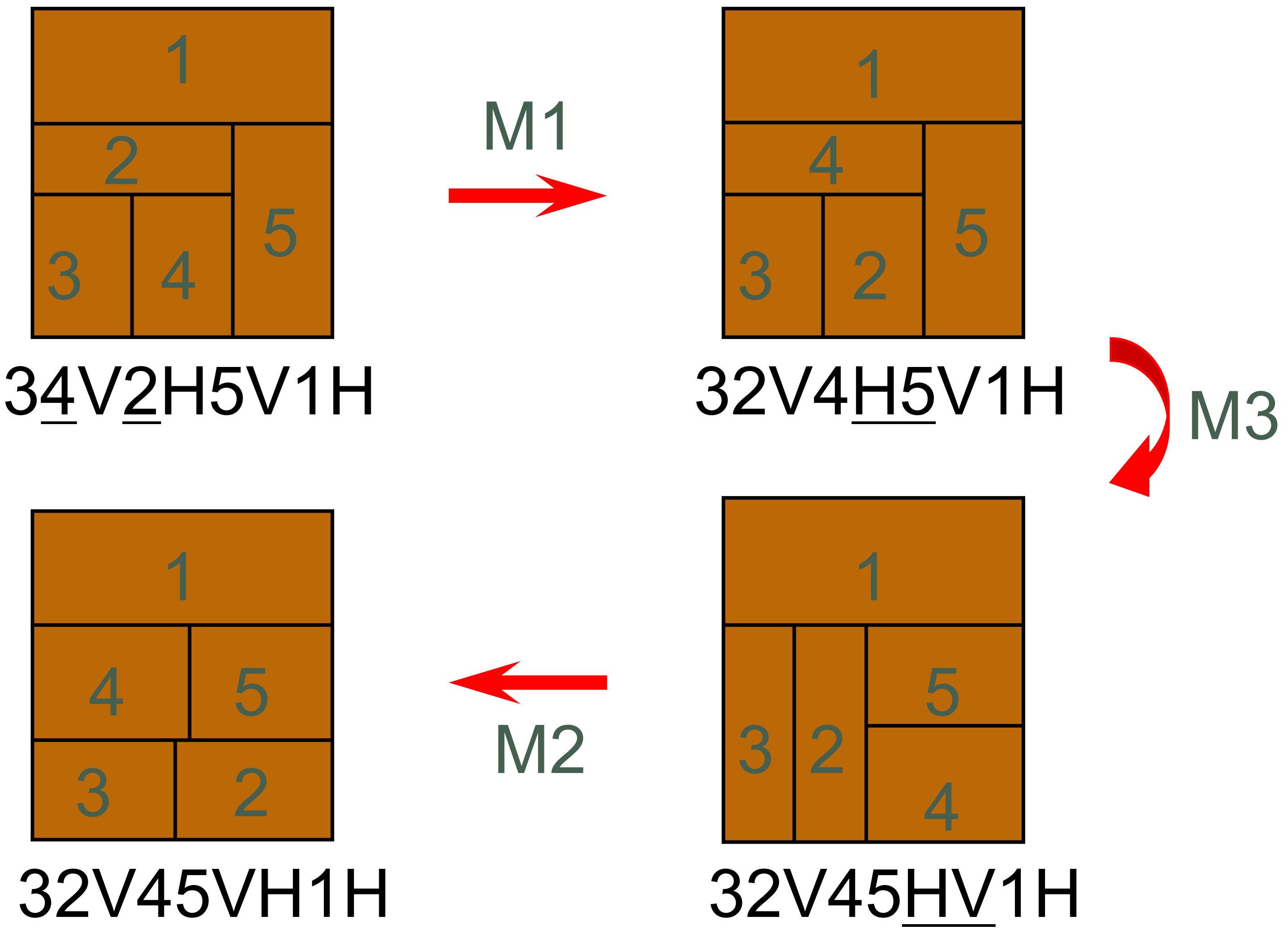
  M3: Swap 2 adjacent operand and operator

  (Note that M3 can give you some invalid NPE.

  So checking for validity after M3 is needed.)

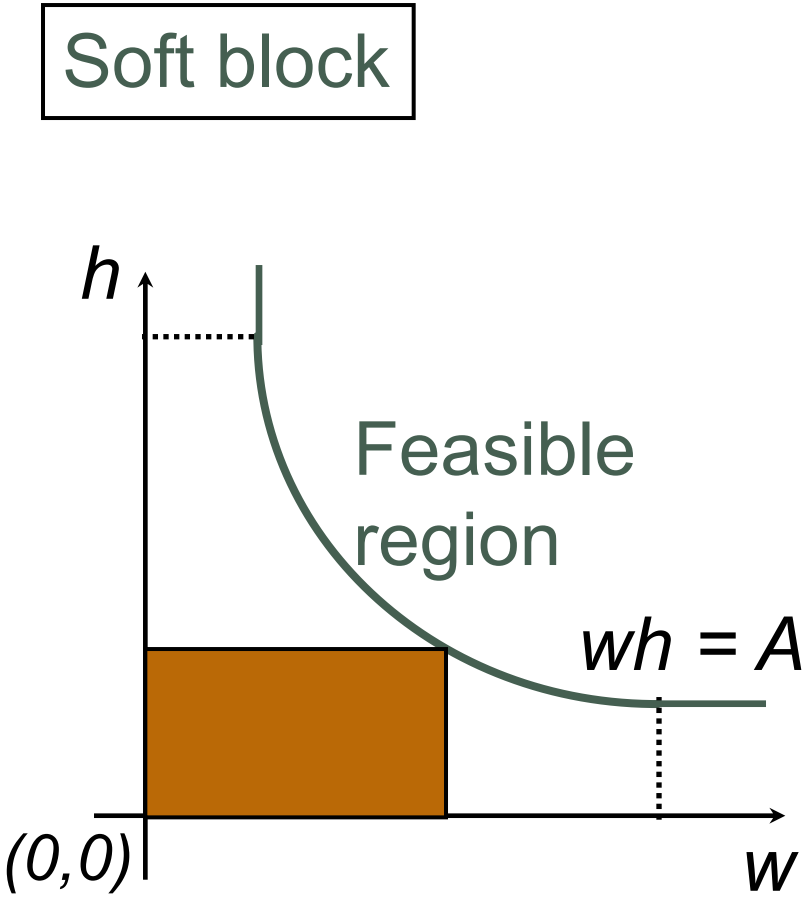
- It can be proved that every pair of valid NPE are connected.

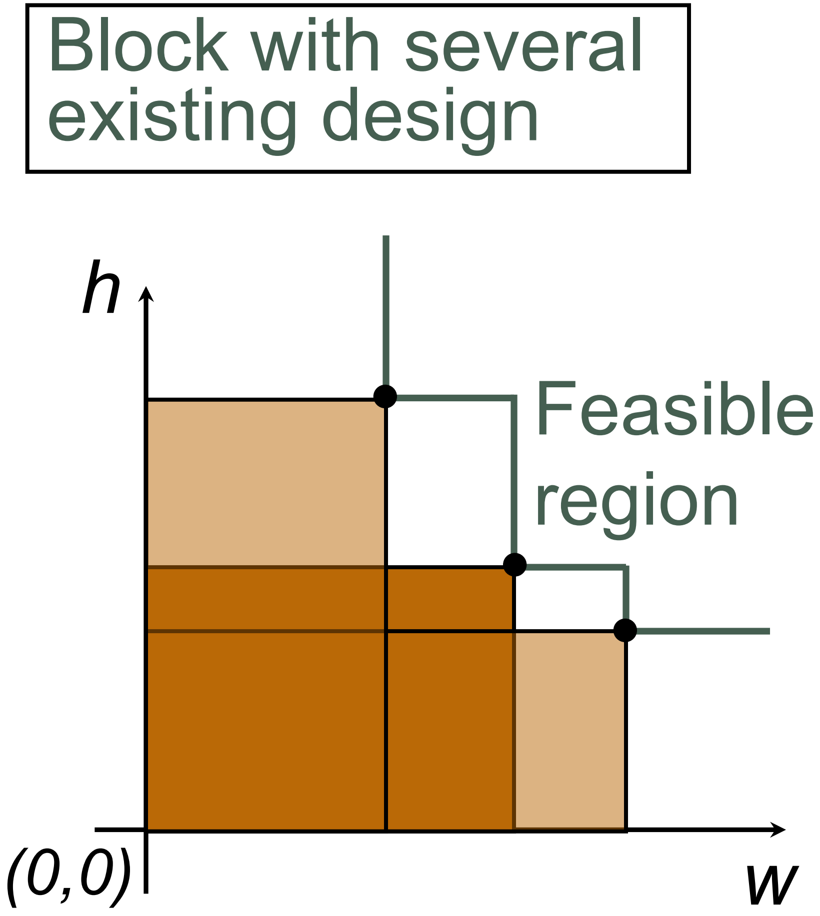
Example of Moves



Shape Curve

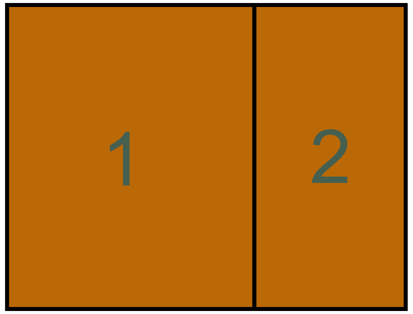
- To represent the possible shapes of a block.

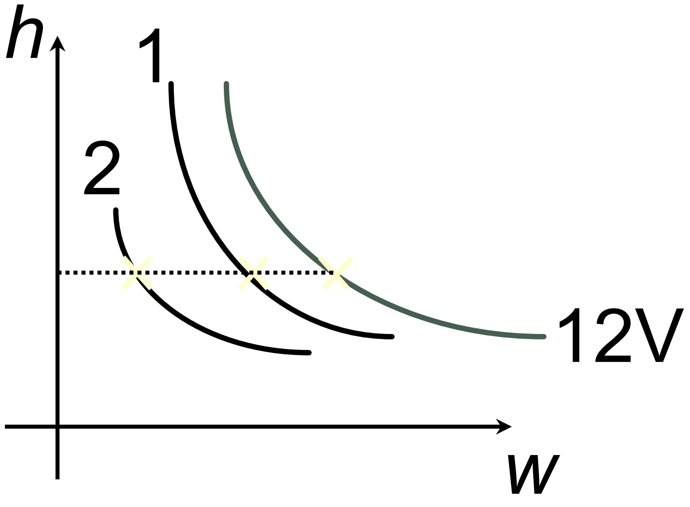




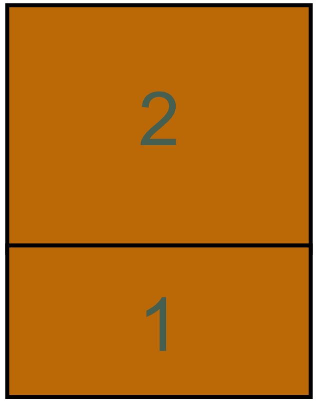
Combining Shape Curves

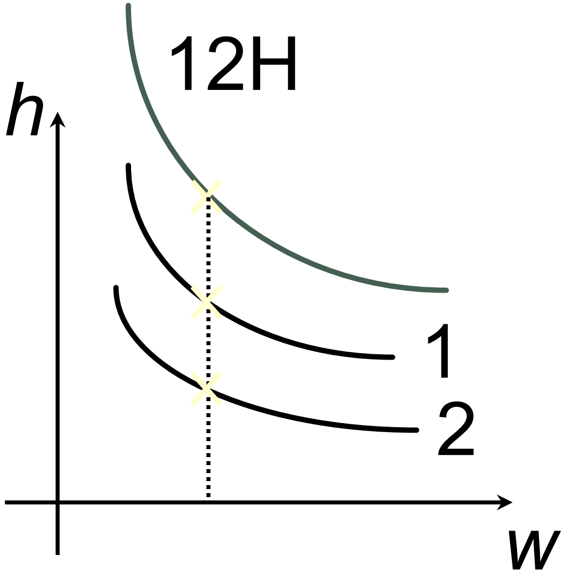
12V





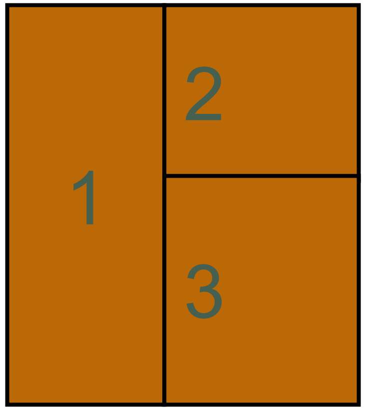
12H

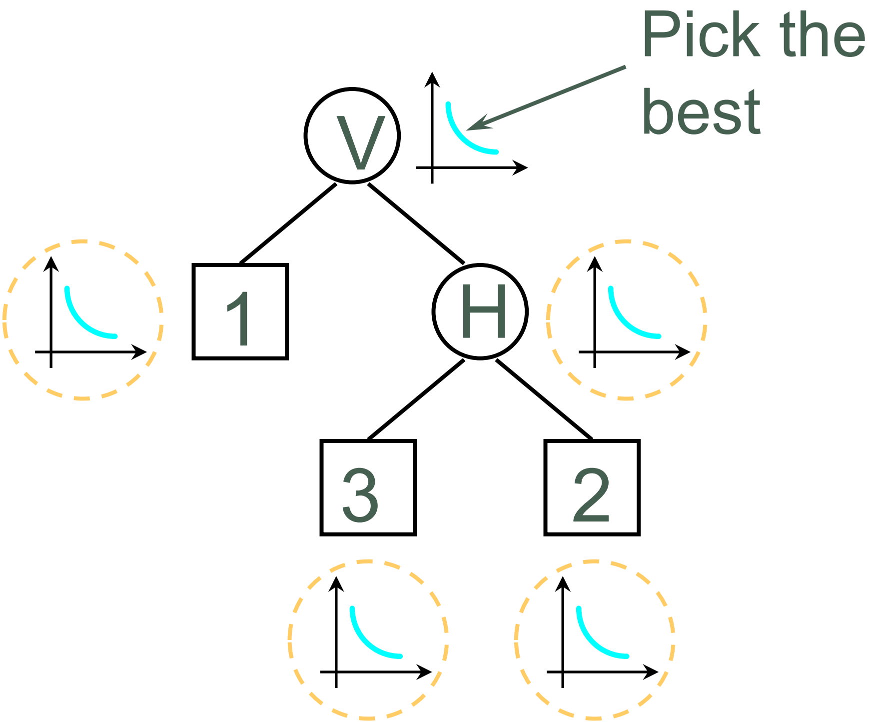




Find the Best Area for a NPE

- Recursively combining shape curves.



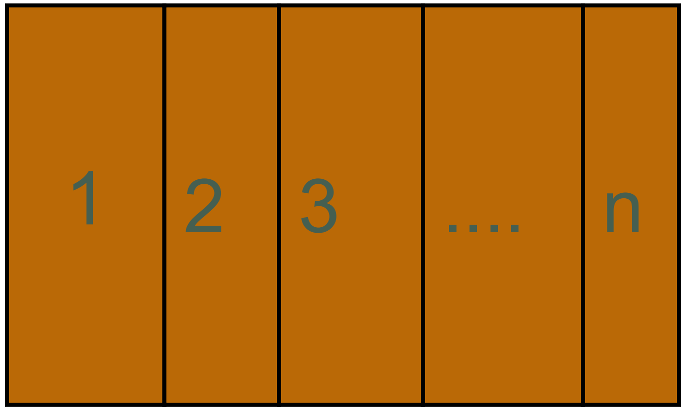


Updating Shape Curves after Moves

- If keeping k points for each shape curve, time for shape curve computation for each NPE is $O(kn)$.
- After each move, there is only small change in the floorplan. So there is no need to start shape curve computation from scratch.
- We can update shape curves incrementally after each move.
- Run time is about $O(k \log n)$.

Initial Solution

- 12V3V4V...nV



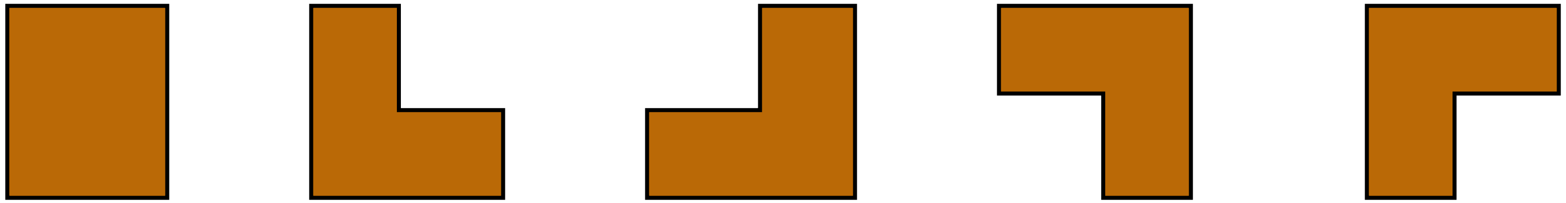
Annealing Schedule

- $T_i = \alpha T_{i-1}$ where $\alpha = 0.85$
- At each temperature, try $k \times n$ moves

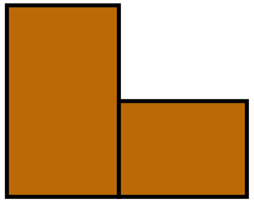
  (k is around 5 to 10)
- Terminate the annealing process if
  - either # of accepted moves < 5%
  - or the temperate is low enough

Handling both Rectangular and L-Shaped Blocks

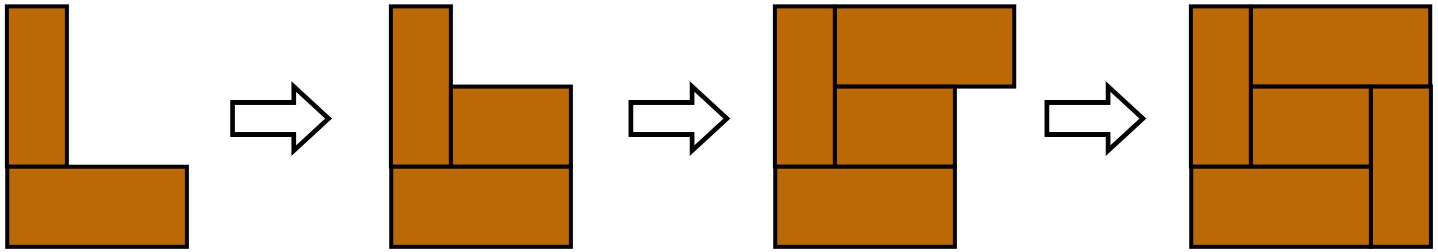
- Rectangular and L-Shaped Blocks
- Possible shapes:



- Note that L-shaped blocks can be produced even if we start with rectangular blocks only.



- Can even generate non-slicing floorplans.



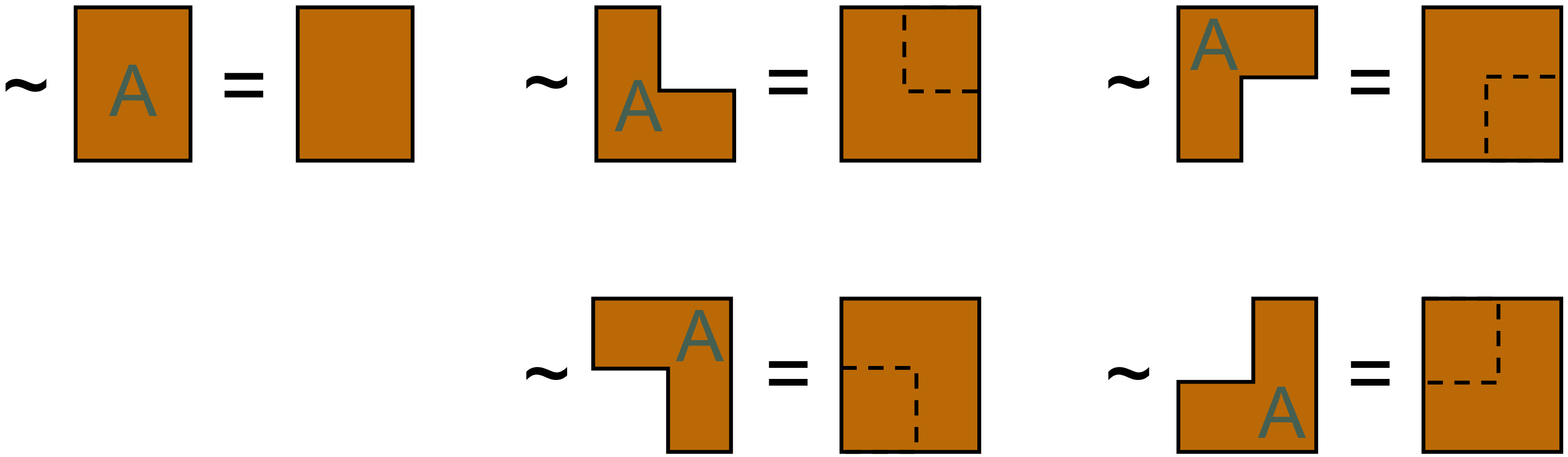
D.F. Wong and C.L. Liu, "Floorplan Design for Rectangular and L-Shaped Modules" ICCAD, 1987, pages 520-523.

Basic Idea

- Similar to the DAC-86 paper by Wong & Liu:
  - Polish Expression representation.
  - Simple moves to locally modify floorplan.
  - Simulated Annealing.
- Differences from the DAC-86 paper:
  - 5 operators and 4 moves defined to handle the more complex shapes.
  - Idea of shape curves no longer applicable.
  - Depend on Simulated Annealing to pick different shapes for blocks probabilistically.

Operators

- 5 operators: ~, V_1, V_2, H_1, H_2
- Completion of A (~A):



Binary Operators V_1, V_2, H_1 and H_2

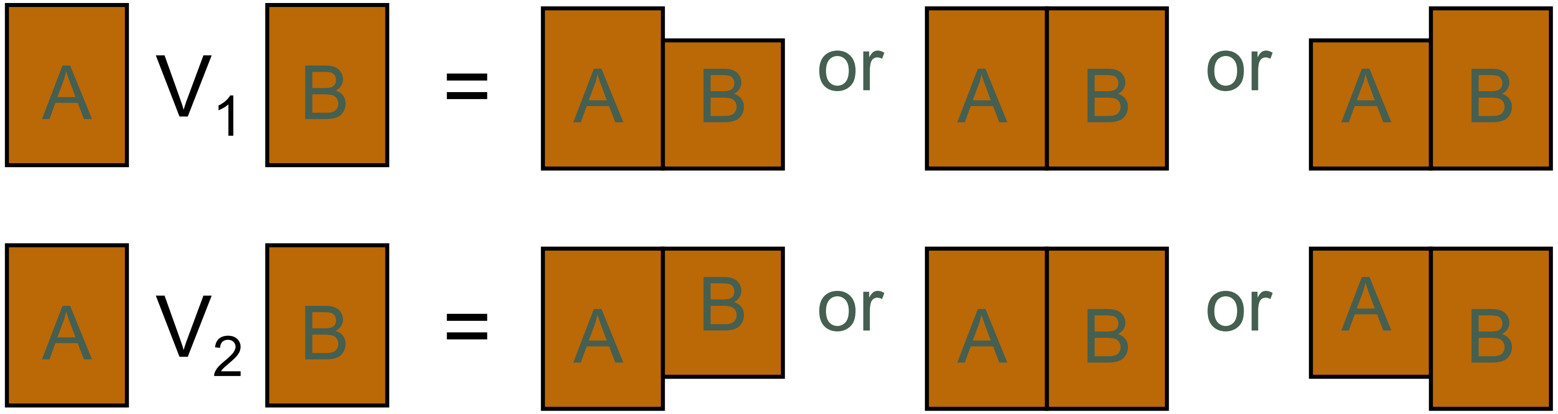
- Need to define what "A op B" means,

  where A and B are rectangular or L-shaped blocks, op is V_1, V_2, H_1 or H_2.

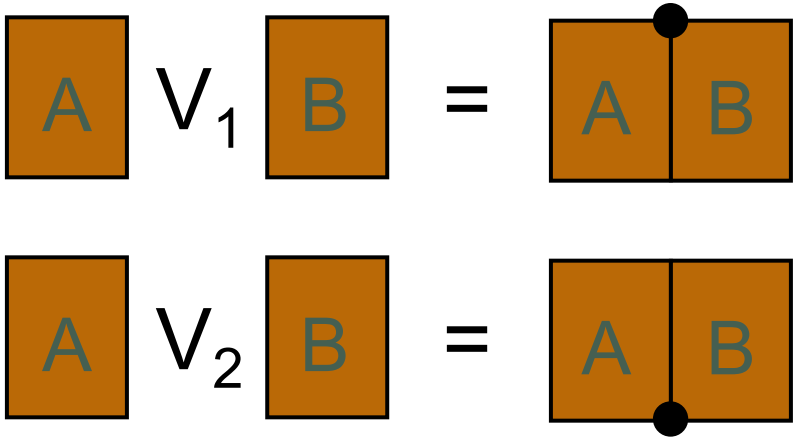
- Total # of ways to combine 2 blocks

  = 5 x 4 x 5 = 100

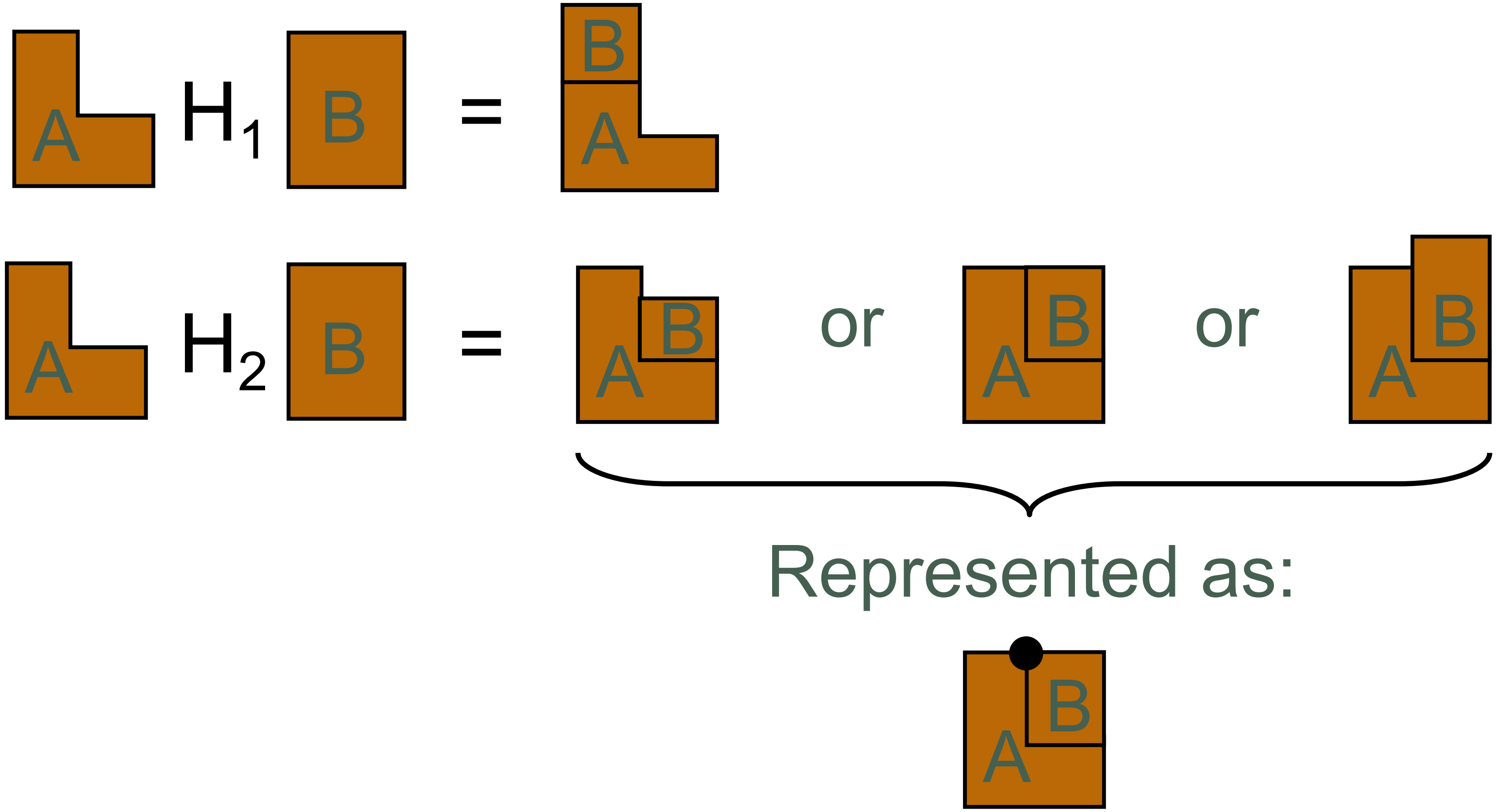
Example of Combining 2 Blocks



- Several possible outcomes. Represented as:



Another Example of Combining



Moves

- Write the Polish Expression in the form:

  $b_1u_1b_2u_2 \ldots b_{2n-1}u_{2n-1}$

  where b_i's are the n blocks, or the n-1 binary operators, and each u_i is either ~ or the empty string ε.

- The moves:

  M1: Modify for some i (change to a different shape or a different binary operator).

  M2: Change u_i to ~ or empty for some i.

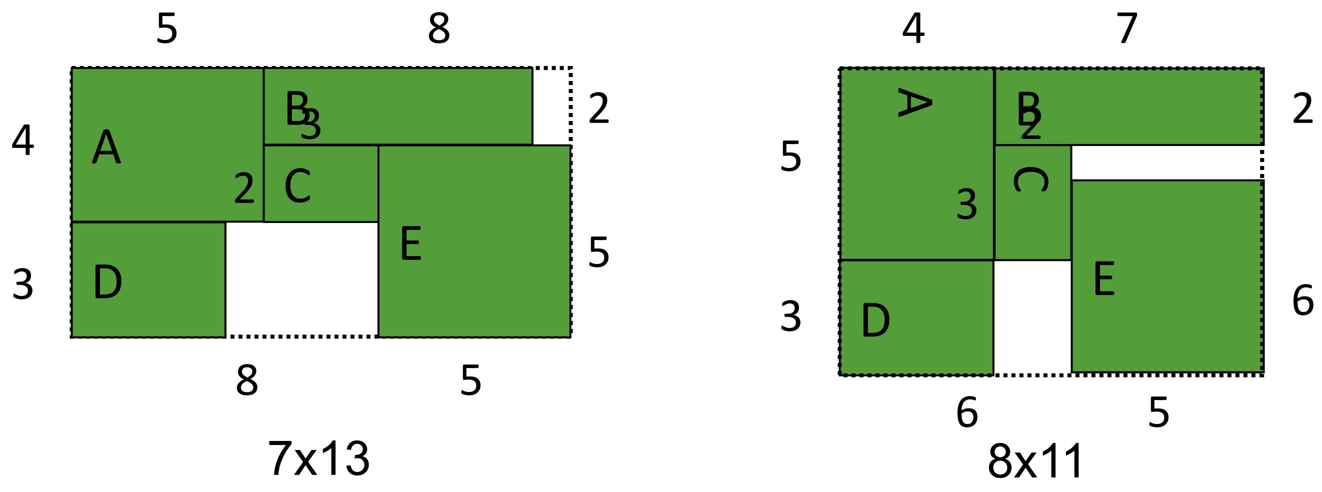
  M3: Swap 2 blocks b_i and b_j.

  M4: Swap b_i and b_{i+1} for some i.

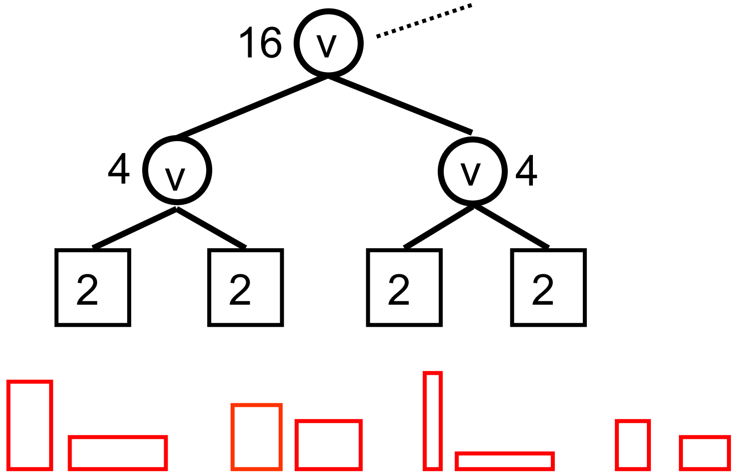
  (M4 can obtain invalid PE. Checking needed.)

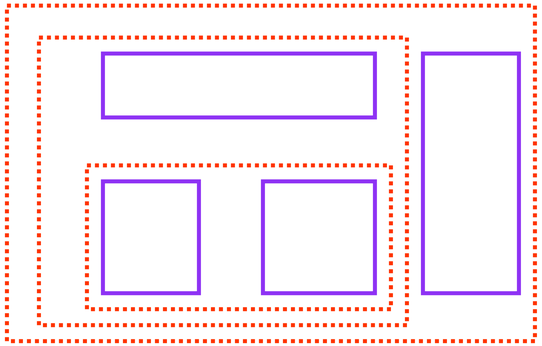
From Slicing Tree to a Floorplan

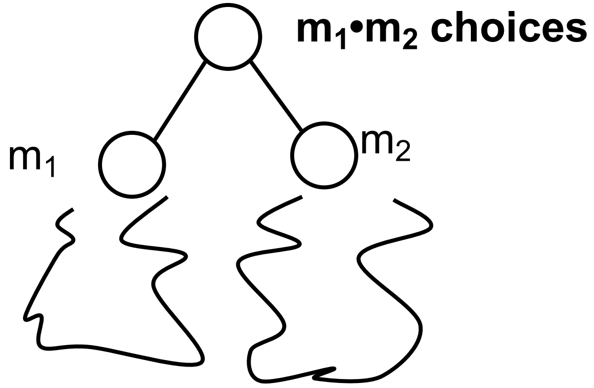
- Given slicing structure and a set of module shapes (or shape list)
  - How to orient these modules such that the total area is smallest?
- "Optimal Orientation of Cells in Slicing floorplan Designs"
  - L. Stockmeyer, Information and Control 57(1983), 91-101
  - This is an earlier paper than [Wong-Liu'86], dealing with simpler problem



Difficulty



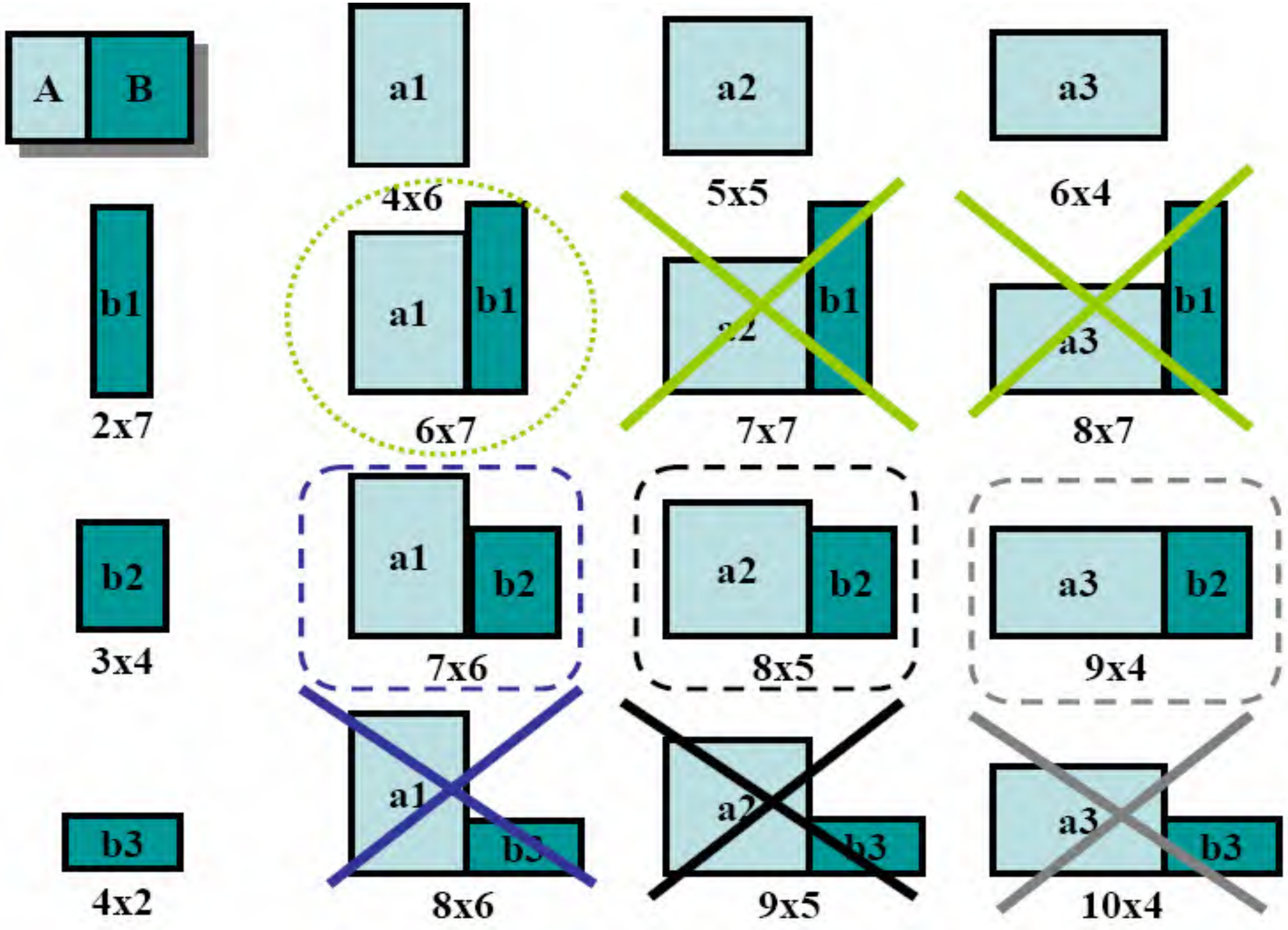




Key Idea

- 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

Example of Merging: only keep irredundant solutions



Stockmeyer Algorithm

- **Phase 1: bottom-up**
  - Input: floorplan tree, modules shapes
  - Start with sorted shapes lists of modules
  - Perform Vertical_Node_Sizing & Horizontal_Node_Sizing
  - When get to the root node, we have a list of shapes. Select the one that is best in terms of area
- **Phase 2: top-down**
  - Traverse the floorplan tree and set module locations

Stockmeyer Algorithm (Cont’d)

Procedure Vertical_Node_Sizing

Input: Sorted lists $L = \{(a_1, b_1), \ldots, (a_s, b_s)\}$, $R = \{(x_1, y_1), \ldots, (x_t, y_t)\}$,

  where $a_i < a_j$, $b_i > b_j$, $x_i < x_j$, $y_i > y_j$ (for all $i < j$)

Output: A sorted list $H = \{(c_1, d_1), \ldots, (c_u, d_u)\}$,

  where $u \leq s + t - 1$, $c_i < c_j$, $d_i > d_j$ (for all $i < j$)

Begin

  $H := \emptyset$

  $i := 1, j := 1, k = 1$

  while $(i \leq s)$ and $(j \leq t)$ do

    $(c_k, d_k) := (a_i + x_j, \max(b_i, y_j))$

    $H := H \cup \{(c_k, d_k)\}$

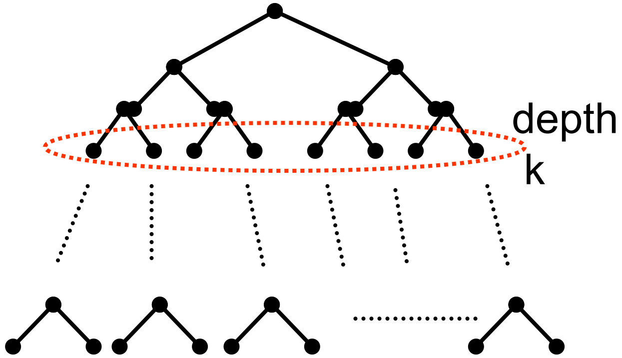
    $k := k + 1$

    if $\max(b_i, y_j) = b_i$ then $i := i + 1$

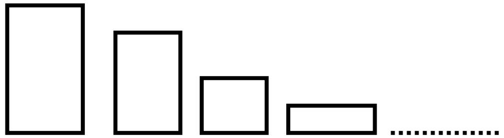
    if $\max(b_i, y_j) = y_j$ then $j := j + 1$

Complexity of the Algorithm

- n= # of leaves = 2 * # of modules
- d=depth of the tree
- Running time= O(nd)
- Storage = O(n)
- because, at depth k,
  - sum of the lengths of the lists =O(n)
  - time to construct these lists =O(n)
  - configurations stored at this node can be release as soon as the node is processed



- Extension
- Each module has k possible shapes
- Running time and storage O(nkd)



Summary: What's BIG idea?

- Floorplan problem is definitely NP hard
- How to represent it compactly is a big deal.
- Slicing is easier to deal with, so let's start with it
- Polish expression is very elegant and easy to make new moves
  - Need to be unique (NPE)
  - Bounding curve for area computation when merging two blocks, which can be computed incrementally
- For a given set of modules and the slicing tree, Stockmeyer's algorithm can give the optimal solution
  - But it's a very ideal situation that doesn't happen often ☹
  - Nice algorithm using dynamic programming

Floorplan Representations

- Slicing
  - Normalized Polish Expression: Wong & Liu [DAC-86]
- Mosaic (and General)
  - Corner Block List (CBL): Hong et al. [ICCAD-00]
  - Q-Sequence: Sakanushi & Kajitani [APCCAS-00]
  - Twin Binary Sequence (TBS): Young, Chu, Shen [ISPD-02]
- General
  - Polar graphs: Ohtsuki et al. [ICCST-70]
  - Sequence pair: Murata et al. [ICCAD-95]
  - Bounded Slicing Grid (BSG): Nakatake [ICCAD-96]
  - Transitive Closure Graph (TCG): Lin & Chang [DAC-01]
- Compacted
  - O-tree: Guo et al. [DAC-99]
  - B*-tree: Chang et al. [DAC-00]

General Floorplanning by Simulated Annealing: Sequence-Pair Representation

- Sequence-Pair vs. Polish Expression
- Sequence-Pair is a succinct representation of non-slicing floorplans of rectangles
  - Just like Polish Expression for slicing floorplans
- Represent a non-slicing floorplan by a pair of sequences of blocks.
- Using Simulated Annealing to find a good sequence-pair
- Can only handle hard blocks
  - i.e., cannot do things like shape-curve computation
- Essentially macro placement
- Techniques for soft block shaping exist (e.g., Lagrangian Relaxation) but are very slow

Murata, Fujiyoshi, Nakatake, Kajitani, "VLSI Module Placement based on Rectangle-Packing by the Sequence-Pair", TCAD, 15:1518-1524, December 1996.

Sequence Pair

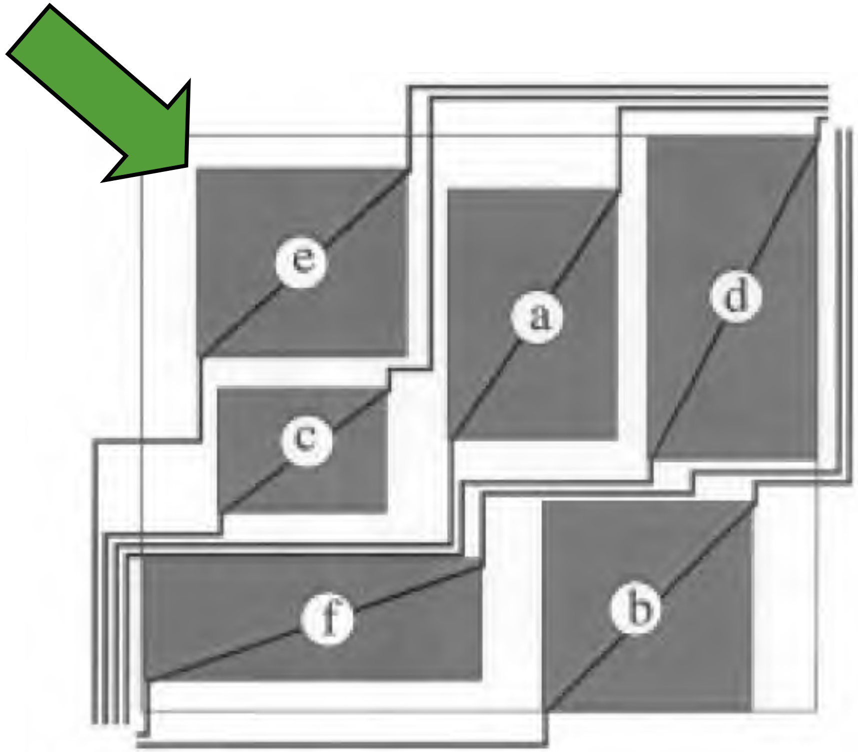
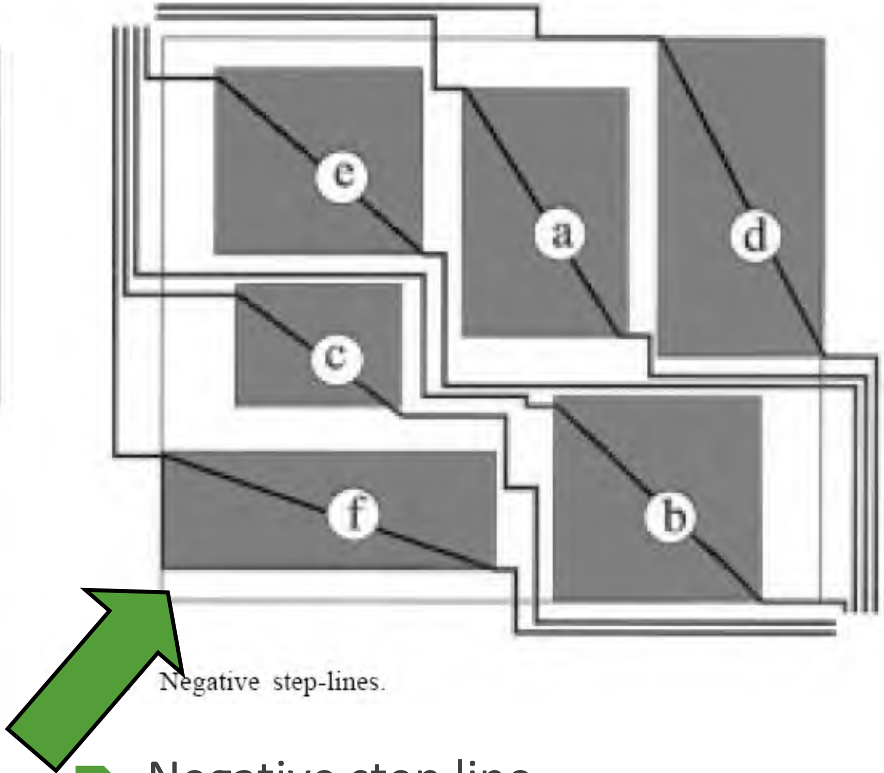


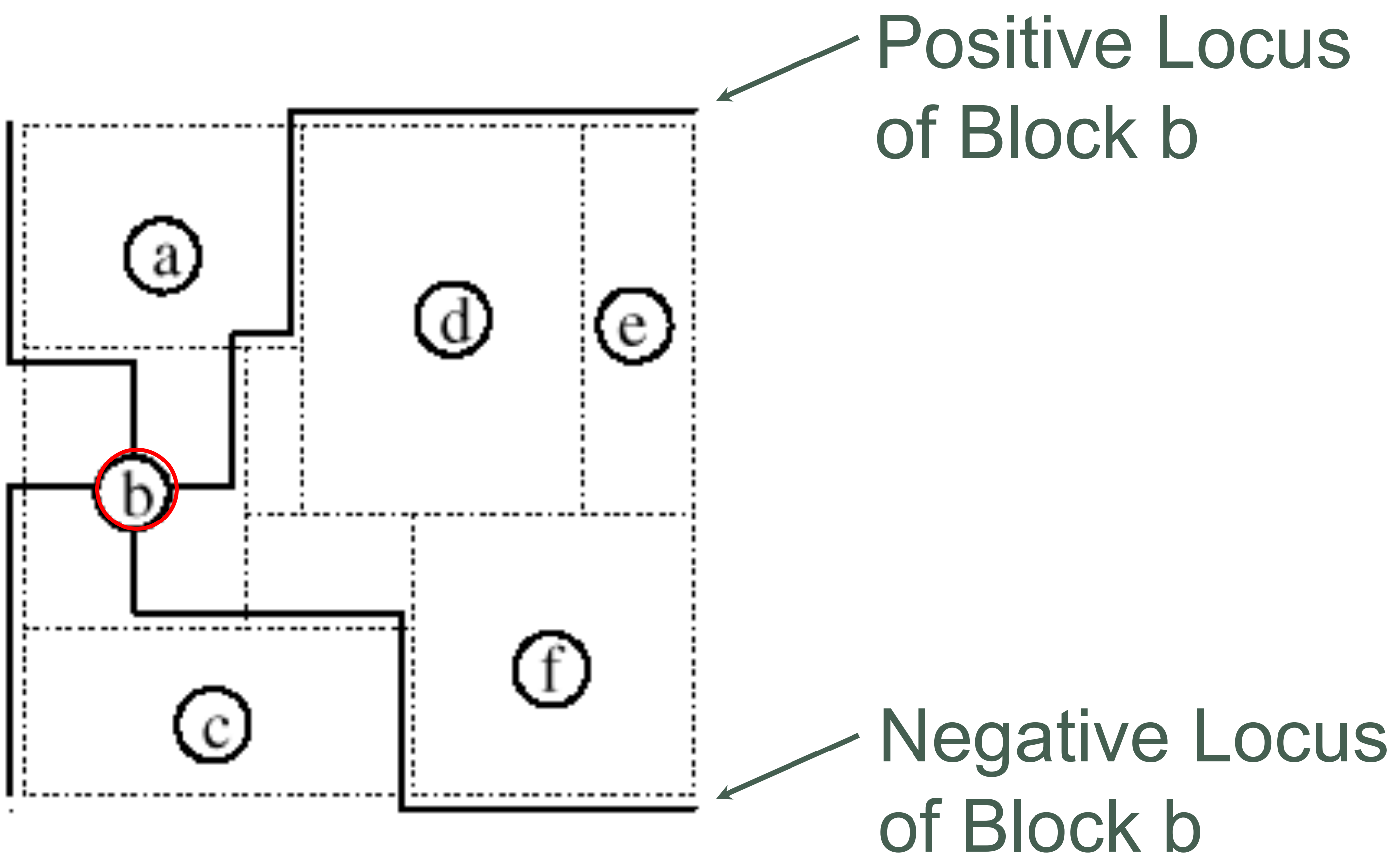
Fig. 2. Positive step-lines.



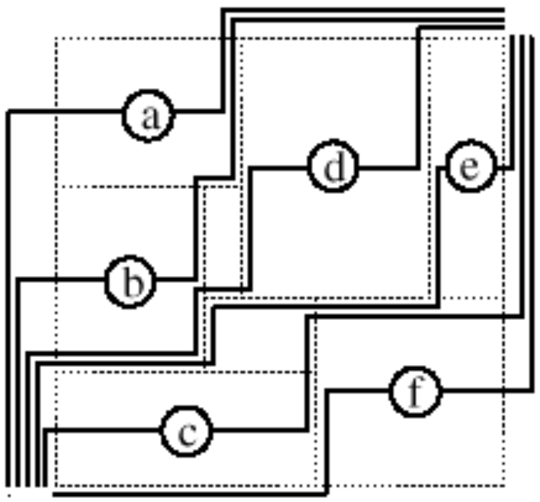
Negative step-lines.

- Positive step line sequence: ecadfb
- Negative step line sequence: fcbead

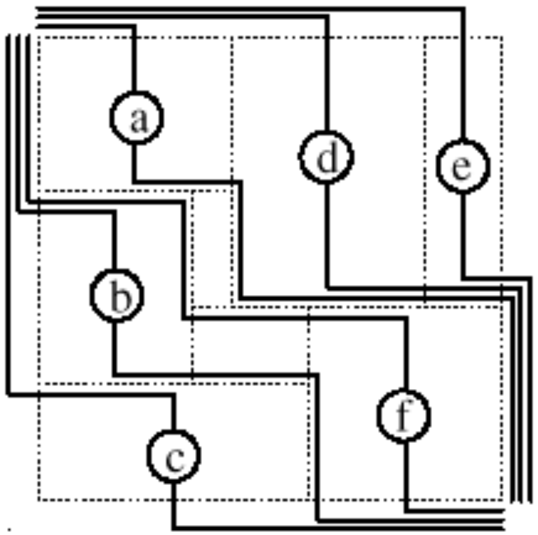
Positive Locus and Negative Locus



Sequence-Pair



Positive Loci



Negative Loci

Sequence-Pair = (abdecf, cbfade)

Geometric Info of Sequence-Pair

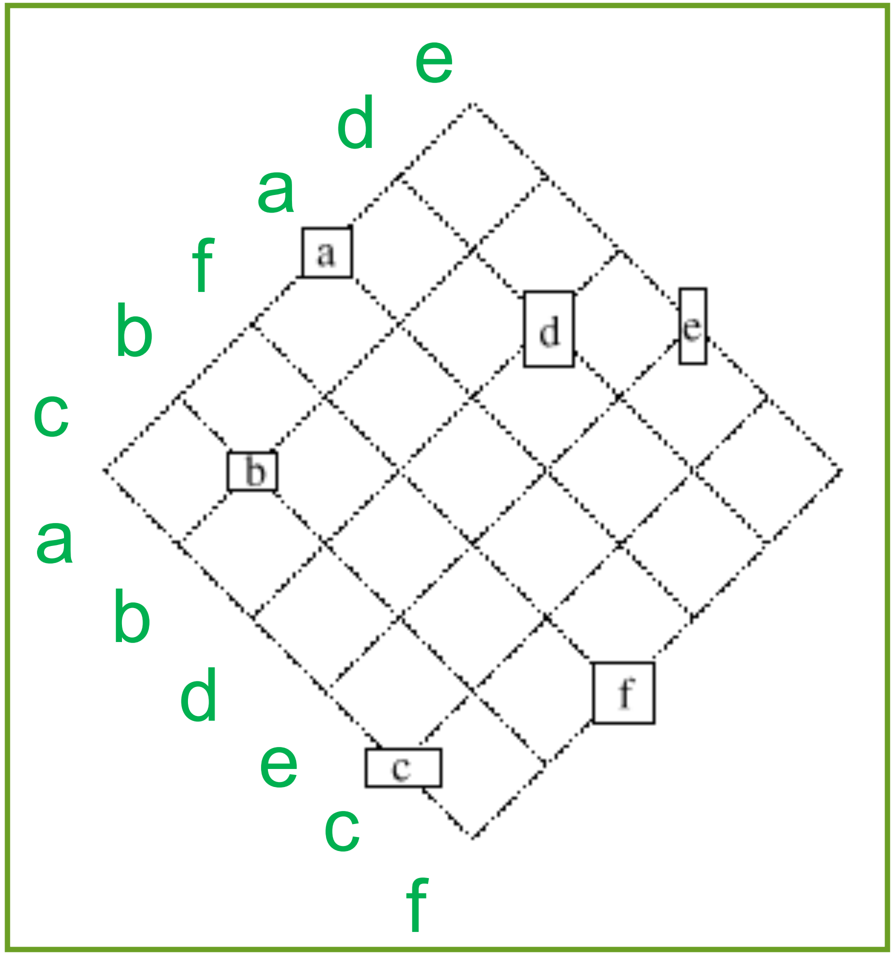
Given a placement and the corresponding sequence-pair (P, N):

- **a** is right to **b** iff **a** is after **b** in both P and N.
- **a** is left to **b** iff **a** is before **b** in both P and N.
- **a** is above **b** iff **a** is before **b** in P and after b in N.
- **a** is below **b** iff **a** is after **b** in P and before b in N.

From Sequence-Pair to a Floorplan

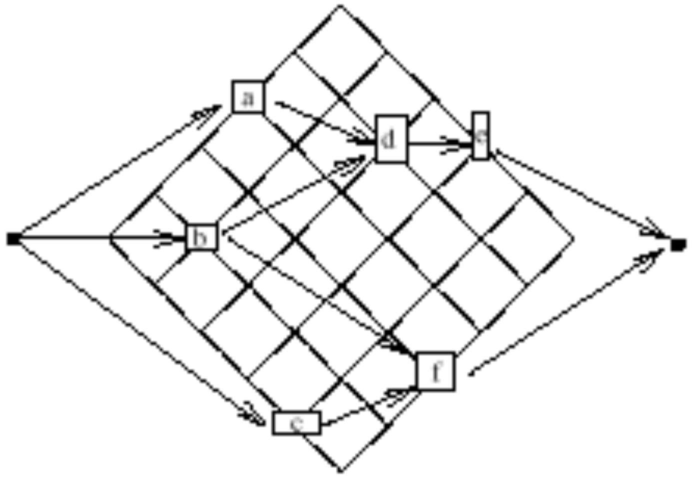
- Given a sequence-pair, the placement with smallest area can be found in $O(n^2)$ time.
- Algorithms of time $O(n \log\log n)$ or $O(n \log n)$ exist. But faster than $O(n^2)$ algorithm only when n is quite large.

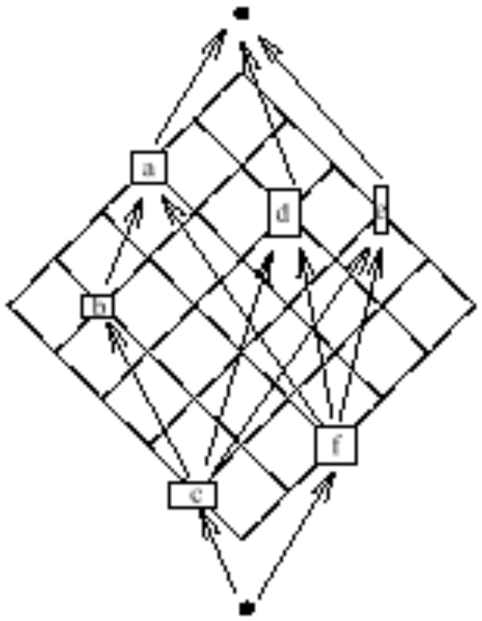
Labeled grid for (abdecf, cbfade)



From Sequence-Pair to a Floorplan

- Distance from left (bottom) edge can be found using the longest path algorithm on the horizontal (vertical) constraint graph.





Horizontal Constraint Graph

Vertical Constraint Graph

Sequence Pair (SP)

A floorplan is represented by a pair of permutations of the module names:

e.g.   1 3 2 4 5

        3 5 4 1 2

A sequence pair (s_1, s_2) of n modules can represent all possible floorplans formed by the n modules by specifying the pair-wise relationship between the modules.

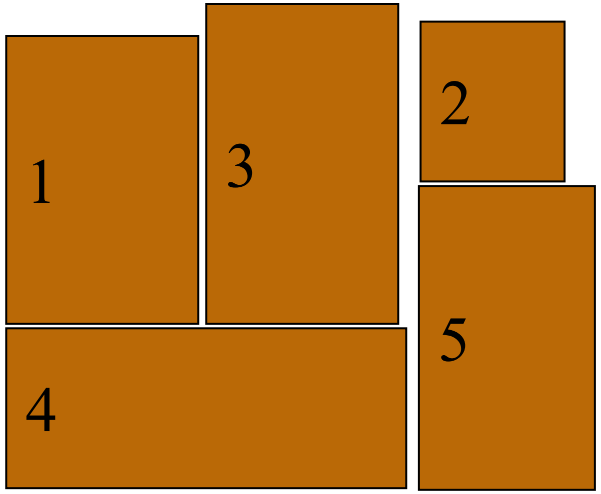
Sequence Pair

Consider a pair of modules A and B. If the arrangement of A and B in s_1 and s_2 are:

- (…A…B…, …A…B…), then the right boundary of A is on the left hand side of the left boundary of B.
- (…A…B…, …B…A…), then the upper boundary of B is below the lower boundary of A.

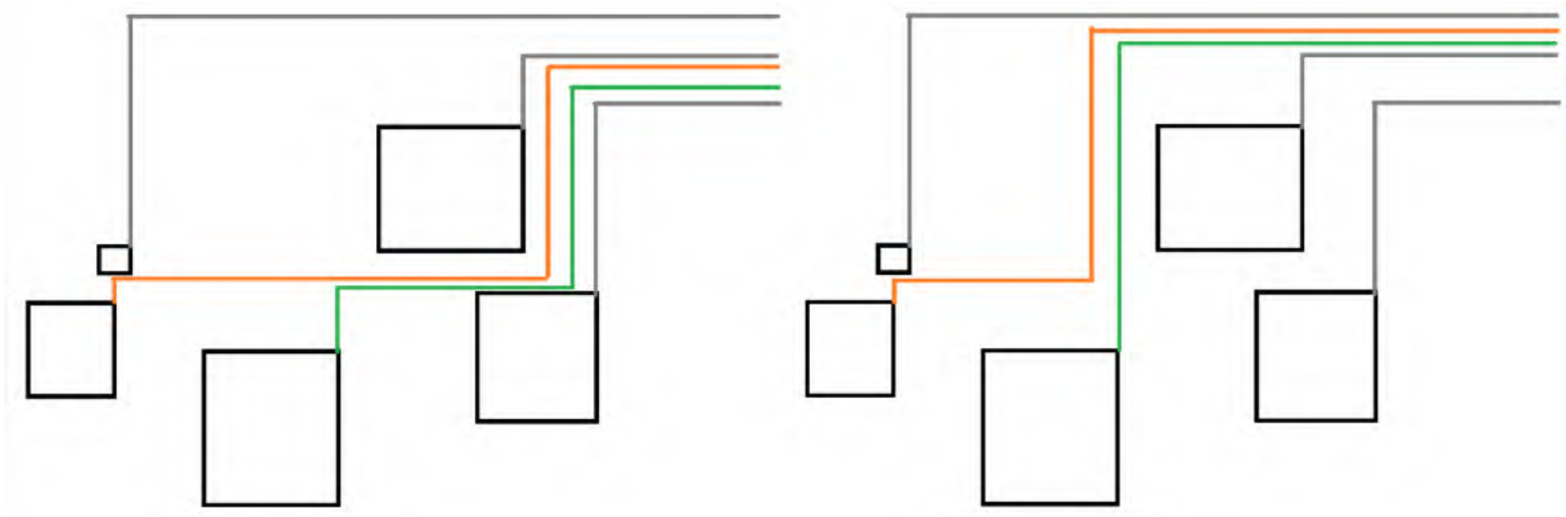
Example

- Consider the sequence pair:
  - (13245, 41352)



- Any other SP that is also valid for this packing?

Non-Unique Sequence Pair



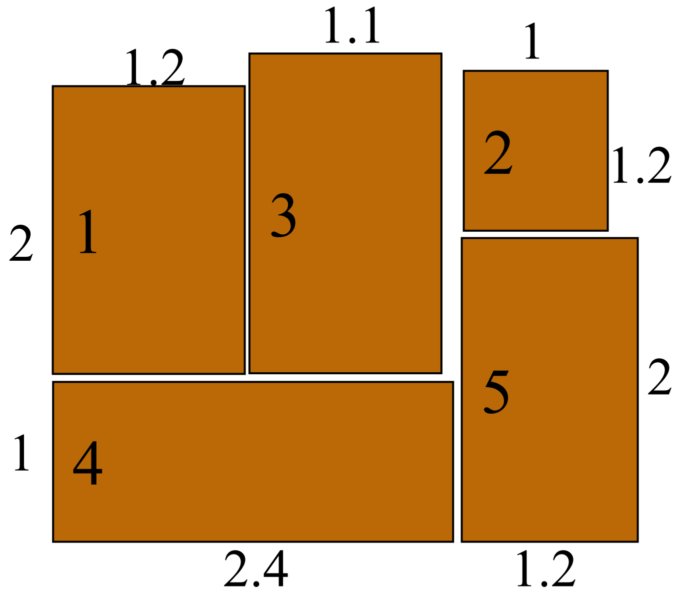
Floorplan Realization

- Floorplan realization is the step to construct a floorplan from its representation.
- How to construct a floorplan from a sequence pair?
- We can make use of the horizontal and vertical constraint graphs (G_h and G_v).

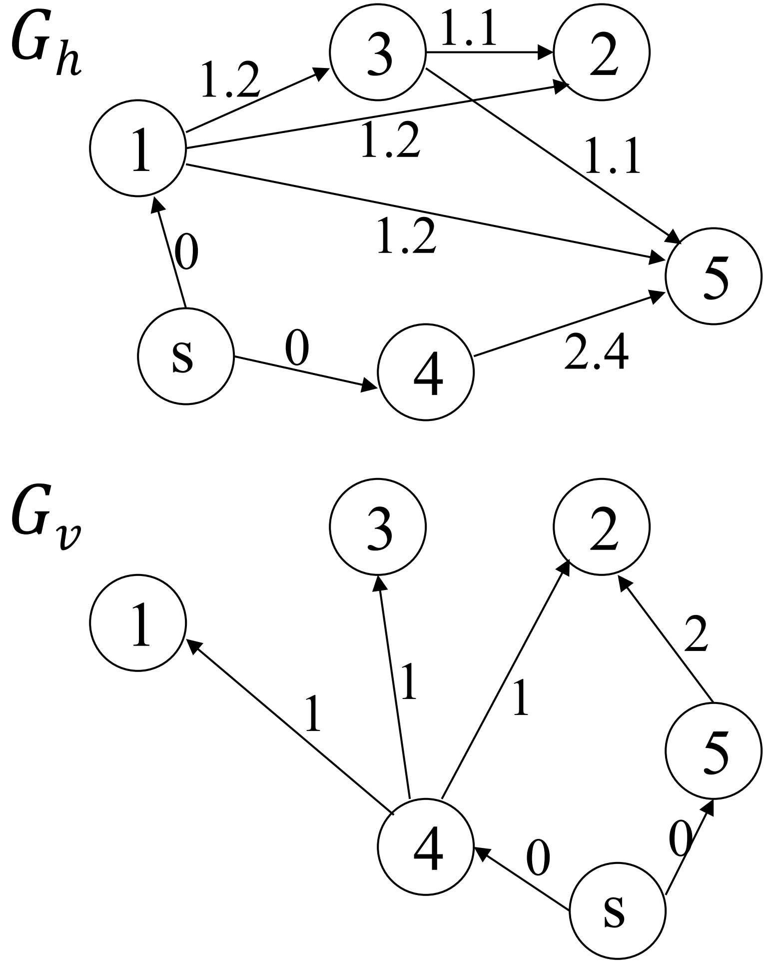
Floorplan Realization

- Whenever we see (…A…B…, …A…B…), add an edge from A to B in G_h with weight w_A.
- Whenever we see (…A…B…, …B…A…), add an edge from B to A in G_v with weight h_A.
- Add a source vertex s to G_h and G_v pointing, with weight 0, to all vertices without incoming edges.
- Finally, find the longest paths from s to every vertex in G_h and G_v (how?), which are the coordinates of the lower left corner of the module in the packing.

Example



(13245, 41352)



Constraint Graphs

- How many edges are there in G_h and G_v in total?
- Is there any transitive edges in G_h and G_v?
- How to remove the transitive edges?
- Can we reduce the size of G_h and G_v to linear, i.e., no. of edges is of order $O(n)$, by removing all the transitive edges?

Moves

- Three kinds of moves in the annealing process:

  M1: Rotate a module, or change the shape of a module

  M2: Interchange 2 modules in both sequences

  M3: Interchange 2 modules in the first sequence

- Does this set of move operations ensure reachability? Why?

Pros and Cons of SP

- Advantages:
  - Simple representation
  - All floorplans can be represented.
  - The solution space is finite. (How big?)
- Disadvantages:
  - Redundant representation. The representation is not 1-to-1.
  - The size of the constraint graphs, and thus the runtime to construct the floorplan is quadratic

Questions

- Can we improve the runtime to realize a floorplan from its SP representation? ("FAST-SP: A Fast Algorithm for Block Placement on Sequence Pair", X. Tang and D.F. Wong, ASP-DAC 2001, pp. 521-526.)

Linear Programming Approach

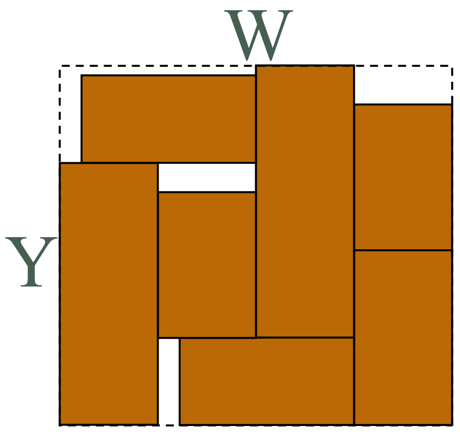
- Mixed Integer Linear Program
- A mathematical program such that:
  - The objective is a linear function.
  - All constraints are linear functions.
  - Some variables are real numbers and some are integers, i.e., “mixed integer”.
- It is almost like a linear program, except that some variables are integers.
- Can you think of which variables may be integer?

“An Analytical Approach to Floorplan Design and Optimization”, Sutanthavibul, Shragowitz and Rosen, IEEE Transaction on CAD, 10:761-769, June 1991.

Problem Formulation

- Minimize the packing area:
  - Assume that one dimension W is fixed.
  - Minimize the other dimension Y.
- Need to have constraints

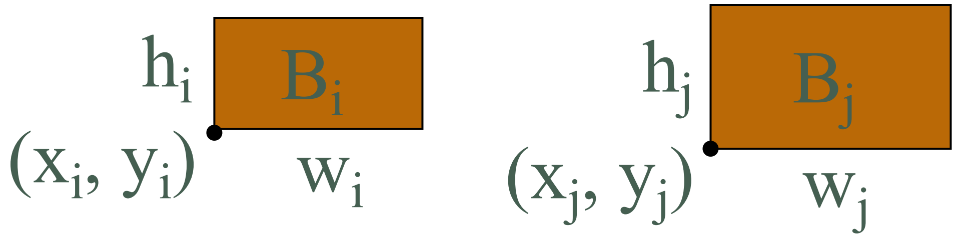
  so that blocks do not overlap.
- Associate each block B_i with 4 variables:
  - x_i and y_i: coordinates of its lower left corner.
  - w_i and h_i: width and height.



Non-overlapping Constraints

- For two non-overlapping blocks B_i and B_j, at least one of the following four linear constraints must be satisfied:

	(1)	$x_i + w_i \le x_j$	if B_i is to the left of B_j
or	(2)	$x_i - w_j \ge x_j$	if B_i is to the right of B_j
or	(3)	$y_i + h_i \le y_j$	if B_i is below B_j
or	(4)	$y_i - h_j \ge y_j$	if B_i is above B_j



Integer Variables

- Use integer (0 or 1) variables x_{ij} and y_{ij}:

  $x_{ij}=0$ and $y_{ij}=0$ if (1) is true.

  $x_{ij}=0$ and $y_{ij}=1$ if (2) is true.

  $x_{ij}=1$ and $y_{ij}=0$ if (3) is true.

  $x_{ij}=1$ and $y_{ij}=1$ if (4) is true.

- Let W and H be upper bounds on the total width and height. Non-overlapping constraints:

$$
\boxed{
\begin{array}{ll}
(1') & x_i + w_i \le x_j + W(x_{ij} + y_{ij}) \\
(2') & x_i - w_j \ge x_j - W(1 + x_{ij} - y_{ij}) \\
(3') & y_i + h_i \le y_j + H(1 - x_{ij} + y_{ij}) \\
(4') & y_i - h_j \ge y_j - H(2 - x_{ij} - y_{ij})
\end{array}
}
$$

Formulation

$$
\begin{array}{lll}
\text{Min.} & Y & \\
\text{s.t.} & 0 \le x_i,\ x_i + w_i \le W & 1 \le i \le n \\
& 0 \le y_i,\ y_i + h_i \le Y & 1 \le i \le n \\
& x_i + w_i \le x_j + W(x_{ij} + y_{ij}) & 1 \le i < j \le n \\
& x_i - w_j \ge x_j - W(1 + x_{ij} - y_{ij}) & 1 \le i < j \le n \\
& y_i + h_i \le y_j + H(1 - x_{ij} + y_{ij}) & 1 \le i < j \le n \\
& y_i - h_j \ge y_j - H(2 - x_{ij} - y_{ij}) & 1 \le i < j \le n \\
& x_{ij} = 0 \text{ or } 1 & 1 \le i < j \le n \\
& y_{ij} = 0 \text{ or } 1 & 1 \le i < j \le n
\end{array}
$$

Formulation with Hard Blocks

- If the blocks can be rotated, use a 0-1 integer variable z_i for each block B_i s.t. $z_i = 0$ if B_i is in the original orientation and $z_i = 1$ if B_i is rotated 90^o .

$$
\begin{array}{lll}
\text{Min.} & Y & \\
\text{s.t.} & 0 \le x_i,\ x_i + z_i h_i + (1 - z_i) w_i \le W & 1 \le i \le n \\
& 0 \le y_i,\ y_i + z_i w_i + (1 - z_i) h_i \le Y & 1 \le i \le n \\
& x_i + z_i h_i + (1 - z_i) w_i \le x_j + W(x_{ij} + y_{ij}) & 1 \le i < j \le n \\
& x_i - z_j h_j - (1 - z_j) w_j \ge x_j - W(1 + x_{ij} - y_{ij}) & 1 \le i < j \le n \\
& y_i + z_i w_i + (1 - z_i) h_i \le y_j + H(1 - x_{ij} + y_{ij}) & 1 \le i < j \le n \\
& y_i - z_j w_j - (1 - z_j) h_j \ge y_j - H(2 - x_{ij} - y_{ij}) & 1 \le i < j \le n \\
& x_{ij} = 0 \text{ or } 1 & 1 \le i < j \le n \\
& y_{ij} = 0 \text{ or } 1 & 1 \le i < j \le n
\end{array}
$$

Formulation with Soft Blocks

- If B_i is a soft block, $w_i h_i = Ai$. But this constraint is quadratic!
- Linearized by taking the first two terms of the Taylor expression of $h_i = Ai/w_i$ at w_{imax} (max. width of block B_i).

$$h_i = h_{imin} + li(w_{imax} - wi)$$

where $h_{imin} = Ai/w_{imax}$ and $l_i = Ai/wimax^2$

Formulation with Soft Blocks

- If B_i is soft and B_j is hard:

$$
\begin{array}{ll}
(1) & x_i + w_i \le x_j + W(x_{ij} + y_{ij}) \\
(2) & x_i - w_j \ge x_j - W(1 + x_{ij} - y_{ij}) \\
(3) & y_i + h_{imin} + \lambda_i (w_{imax} - w_i) \le y_j + H(1 - x_{ij} + y_{ij}) \\
(4) & y_i - h_j \ge y_j - H(2 - x_{ij} - y_{ij})
\end{array}
$$

- If both B_i and B_j are soft:

$$
\begin{array}{ll}
(1) & x_i + w_i \le x_j + W(x_{ij} + y_{ij}) \\
(2) & x_i - w_j \ge x_j - W(1 + x_{ij} - y_{ij}) \\
(3) & y_i + h_{imin} + \lambda_i (w_{imax} - w_i) \le y_j + H(1 - x_{ij} + y_{ij}) \\
(4) & y_i - h_{jmin} - \lambda_j (w_{jmax} - w_j) \ge y_j - H(2 - x_{ij} - y_{ij})
\end{array}
$$

Another way to linearize

- Assumptions: w_i, h_i are unknown; area lower bound: A_i.
- Module size constraints: $w_i h_i \geq A_i$; $a_i \leq \frac{w_i}{h_i} \leq b_i$.
- Hence, $w_{min} = \sqrt{A_i a_i}$, $w_{max} = \sqrt{A_i b_i}$, $h_{min} = \sqrt{\frac{A_i}{b_i}}$, $h_{max} = \sqrt{\frac{A_i}{a_i}}$.
- $w_i h_i \geq A_i$ nonlinear! How to fix?
  - Can apply a first-order approximation of the equation: a line passing through (w_{min}, h_{max}) and (w_{max}, h_{min}).

$$
\begin{aligned}
h_i &= \Delta_i w_i + c_i && /*\ y = mx + c\ */ \\
\Delta_i &= \frac{h_{max} - h_{min}}{w_{min} - w_{max}} && /*\ slope\ */ \\
c_i &= h_{max} - \Delta_i w_{min} && /*\ c = y_0 - m x_0\ */
\end{aligned}
$$

  - Substitute $\Delta_i w_i + c_i$ for h_i to form linear constraints (x_i, y_i, w_i are unknown; Δ_i, c_i, can be computed as above).

Solving Linear Program

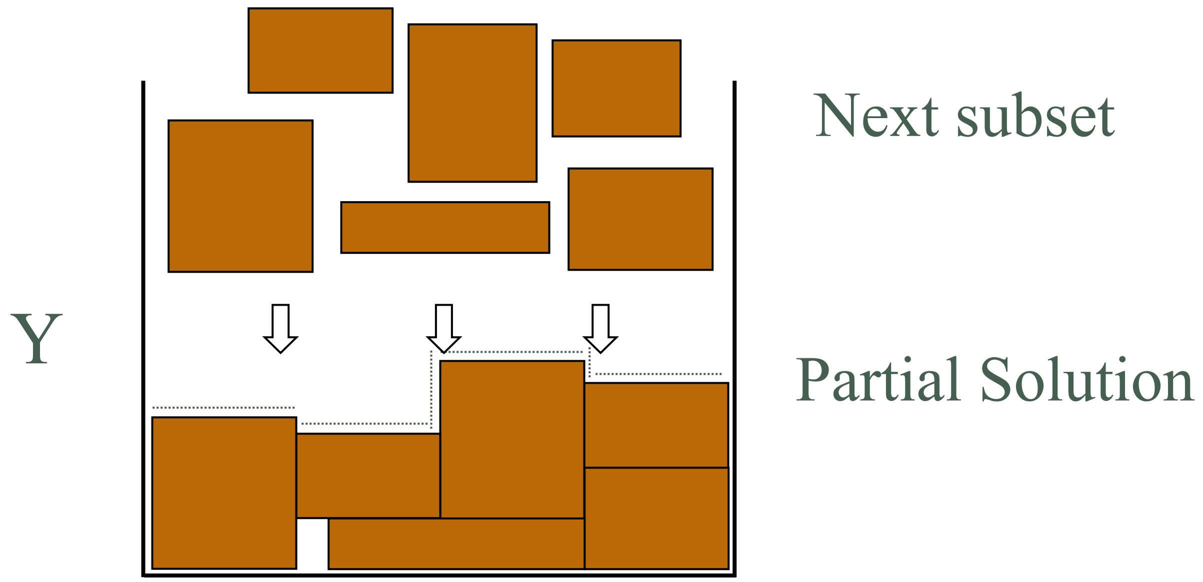
- Linear Programming (LP) can be solved by classical optimization techniques in polynomial time.
- Mixed Integer LP (MILP) is NP-Complete.
  - The run time of the best known algorithm is exponential to the number of variables and equations

Complexity

- For a problem with n blocks, and for the simplest case, i.e., all blocks are hard:
  - $2n$ continuous variables (xi, y_i)
  - $2n(n-1)\ + n$ integer variables (x_{ij}, y_{ij}, z_i)
  - $4n^2 - 2n$ linear constraints
- Practically, this method can only solve small size problems.

Successive Augmentation

- A classical greedy approach to keep the problem size small: repeatedly pick a small subset of blocks to formulate a MILP, solve it together with the previously picked blocks with fixed locations and shapes:



Summary of Floorplanning

- Stockmeyer
  - Slicing with given set of modules
  - Dynamic programming (only keep irredundant solutions)
- Wong-Liu
  - Slicing floorplan
  - Nice polish expression
- Sequence pair
  - Nonslicing
  - Nice compact representation
- Mixed Integer Linear Programming
  - Nonslicing
  - Not scalable