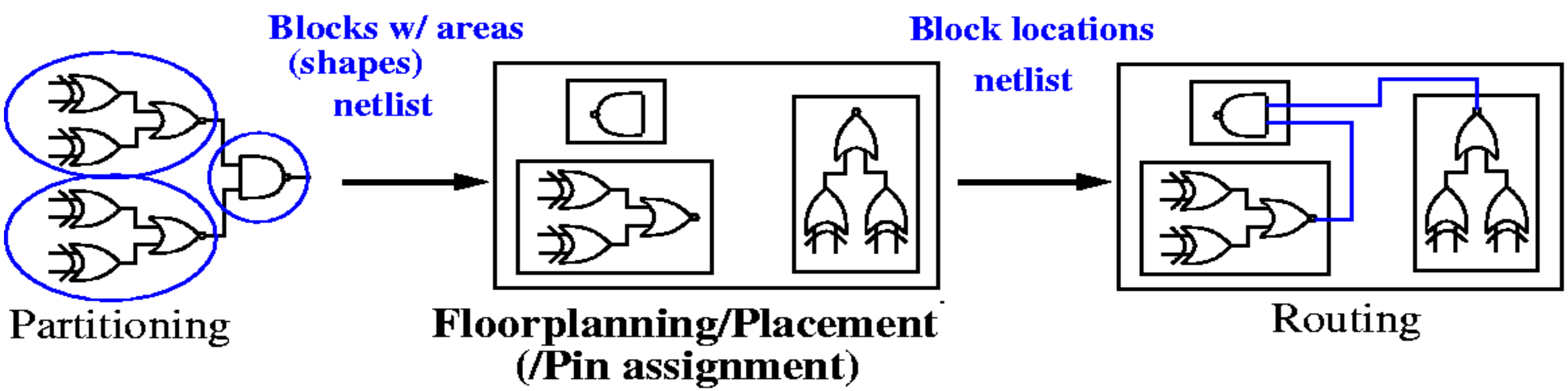
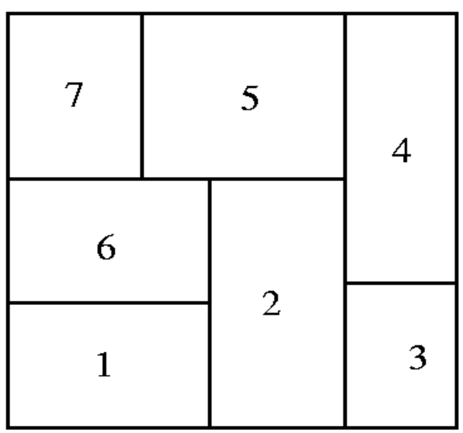
Floorplanning

- Partitioning leads to
  - Modules (or called blocks) with well-defined areas and shapes (*hard modules*).
  - Modules with approximated areas and no particular shapes (*soft modules*).
  - A netlist specifying connections between the modules.
- Objectives
  - Find **locations** for all modules, as well as **orientations (if allowable)** for hard modules.
  - Find shapes (and pin locations) of the soft modules.

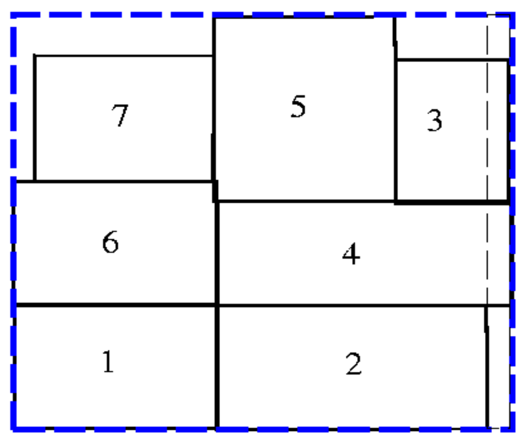


Floorplanning Problem

- Inputs:
  - A set of modules, hard or soft.
  - Pin locations of hard modules.
  - A netlist.
- Objectives: Minimize area, reduce wirelength for (critical) nets, maximize routability (minimize congestion), determine shapes of soft modules.



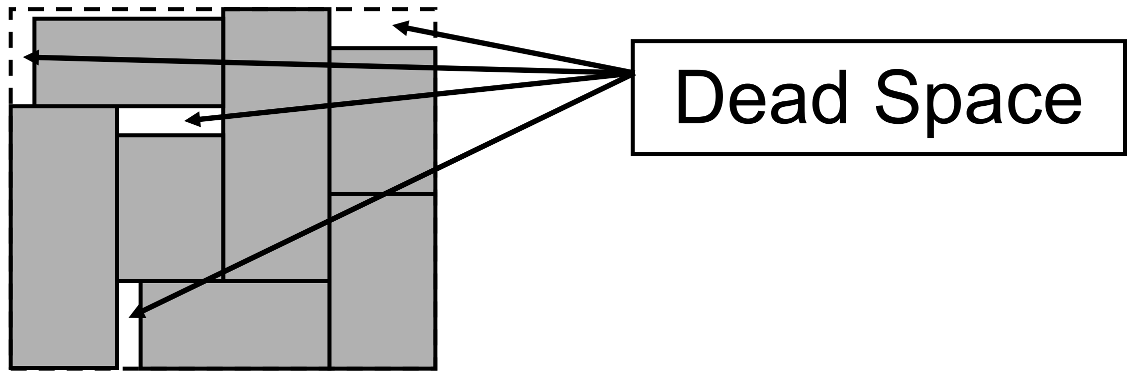
An optimal floorplan, in terms of area



A non–optimal floorplan

Dead Space

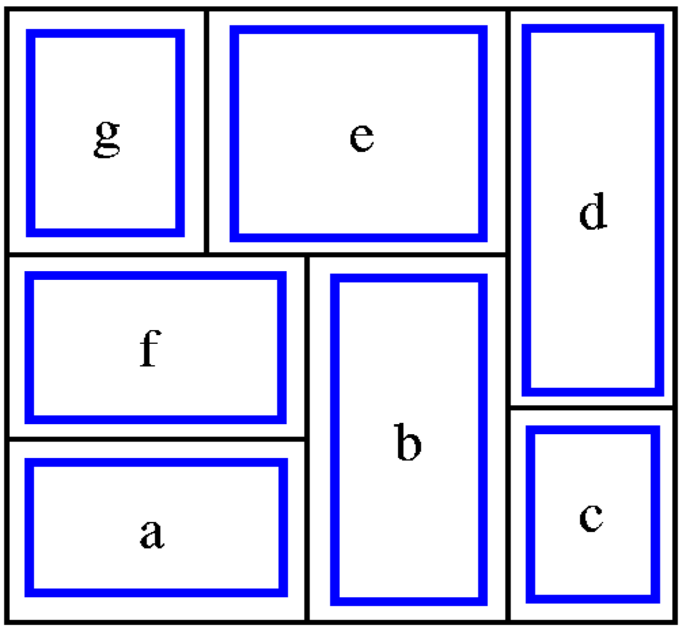
- The space that is wasted.

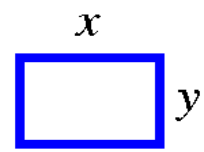


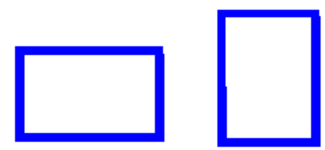
- Minimizing area is the same as minimizing dead space.
- Percentage of dead space

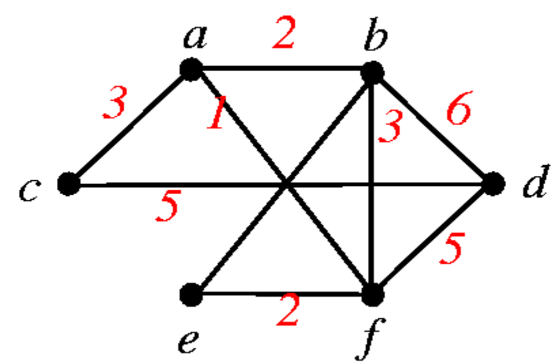
  = ((Area of resulting rectangle / Total area of all modules) – 1) x 100%.

Floorplan Design



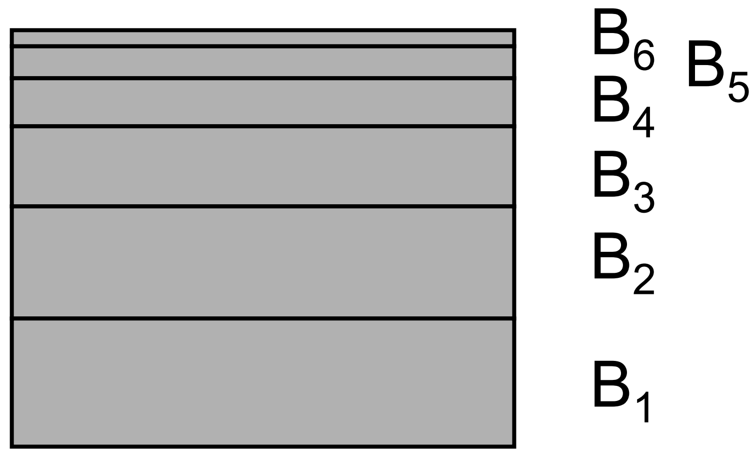
- *Modules:* 

- *Area: A=xy*
- *Aspect ratio: r <= y/x <= s*
- *Rotation:* 
- *Module connectivity*



Bounds on Aspect Ratios

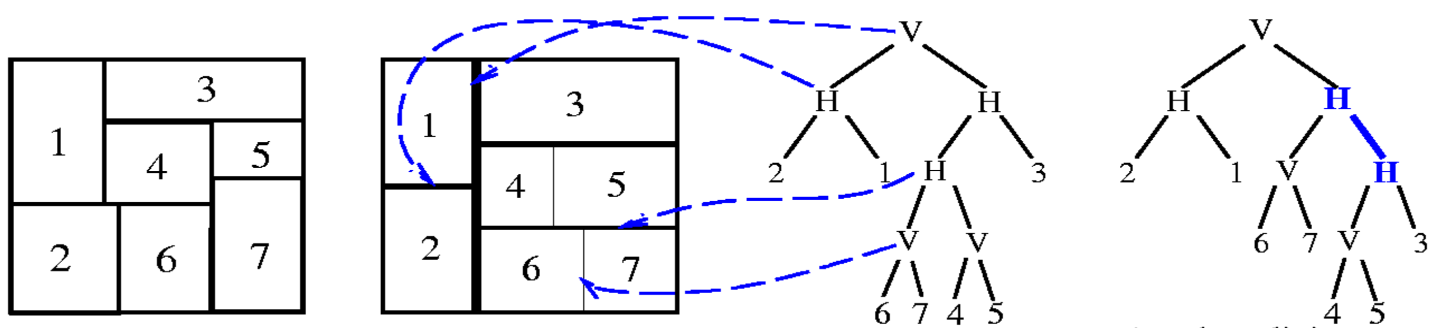
- If there is no bounds on aspect ratios, we can always pack modules completely tight (i.e., no dead space).



- We do not want to layout a module as a long strip.

Terminology

- **Rectangular dissection**: Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- **Slicing structure**: a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.
- **Slicing tree**: A binary tree, where each internal node represents a vertical cut line or horizontal cut line, and each leaf a basic rectangle.
- **Skewed slicing tree**: A slicing tree in which no node and its right child are the same type of cut line.



Non–slicing floorplan   Slicing floorplan   A slicing tree (skewed)   Another slicing tree (non–skewed)

Slicing Floorplan Design by Simulated Annealing

- Related works
  - Wong & Liu, "A new algorithm for floorplan design," DAC'86.
  - Wong, Leong & Liu, *Simulated Annealing of VLSI Design*, pp. 31-51, Kluwer Academic Publishers, 1988.
- Ingredients: solution space, neighborhood structure, cost function, annealing schedule.

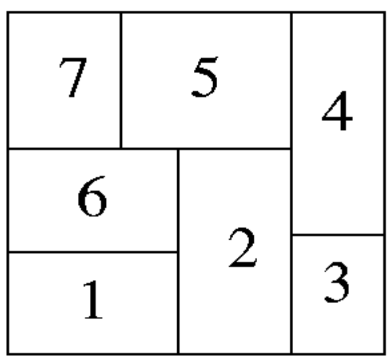
Solution Representation

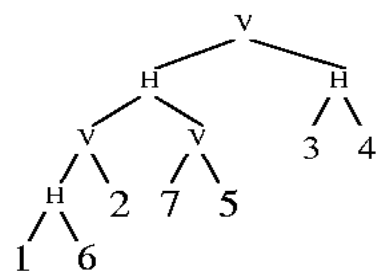
- An expression $E = e_1e_2\ldots e_{2n\text{-}1}$, where $e_i \in \{1, 2, \ldots, n, H, V\}$, $1 \leq i \leq 2n\text{-}1$, is a **Polish expression** of length $2n$-1 iff
  1. every operand j, $1 \leq j \leq n$, appears exactly once in E;
  2. (**balloting property**) for every sub-expression $E_i = e_1\ldots e_i$, $1 \leq i \leq 2n\text{-}1$, #operands > #operators.

1 6 H 3 5 V 2 H V 7 4 H V

\# of operands = 4 .... = 7
\# of operators = 2 .... = 5

- Polish expression ↔ Postorder traversal.
- *ijH*: *i* below *j*; *ijV*: *i* on the left of *j*.



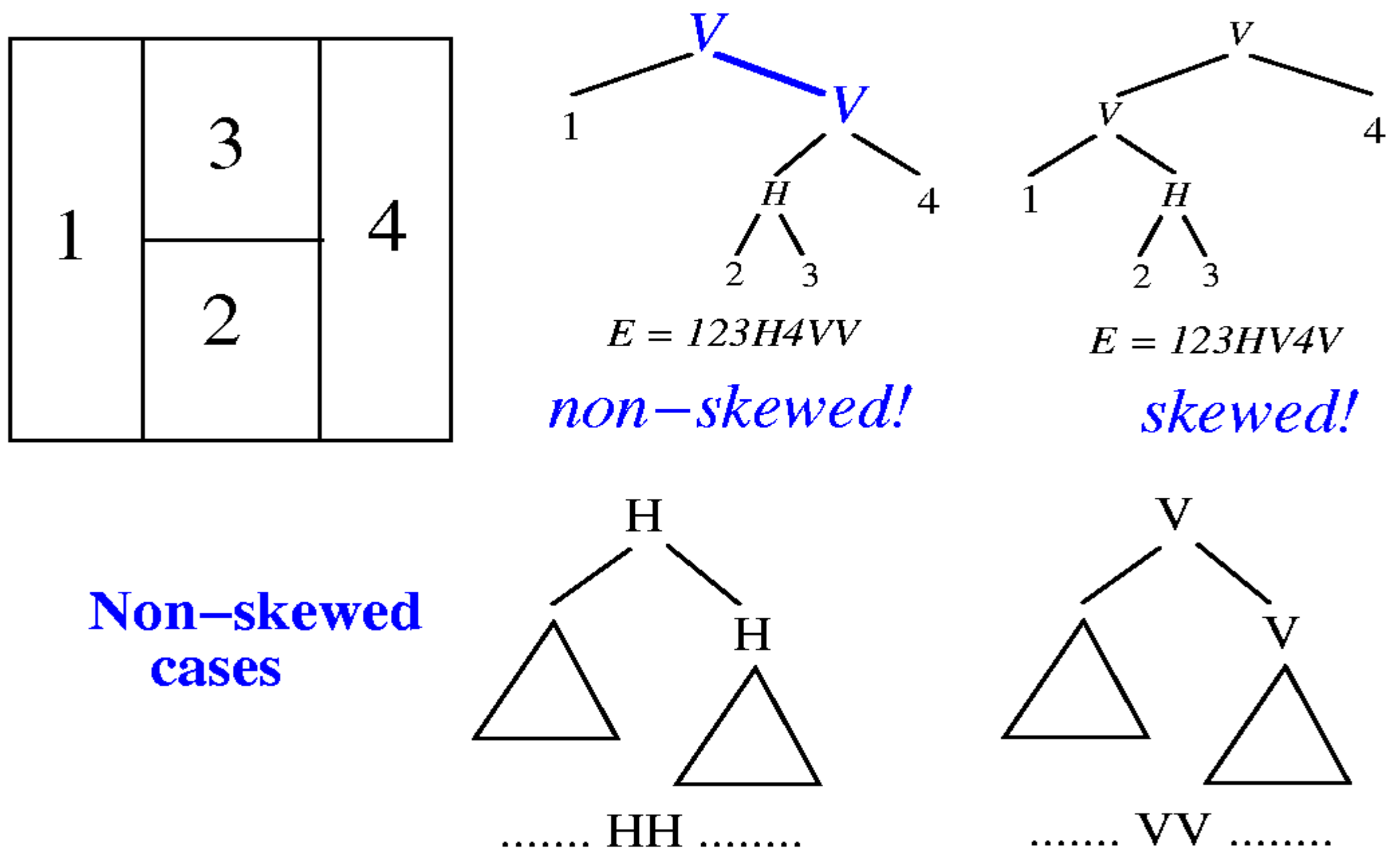


E = 16H2V75VH34HV

*E = 16+2*75*+34+**

Postorder traversal of a tree!

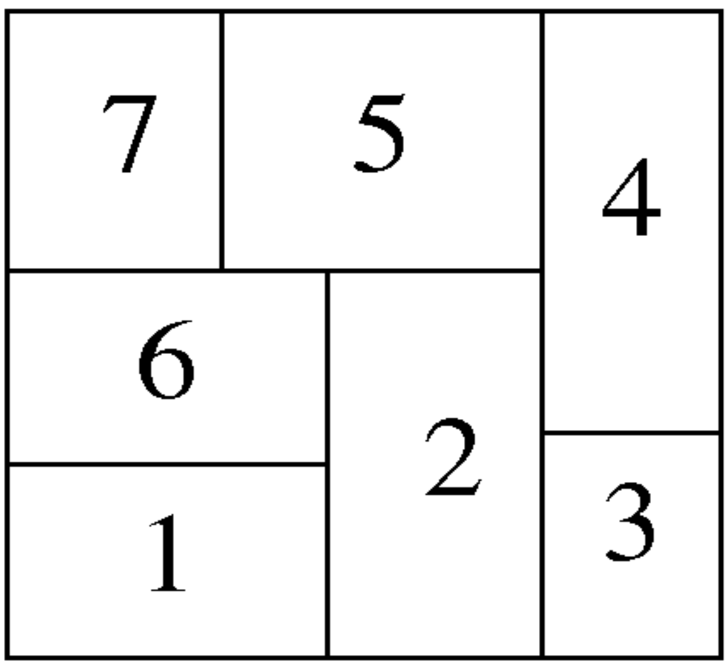
Solution Representation (cont'd)

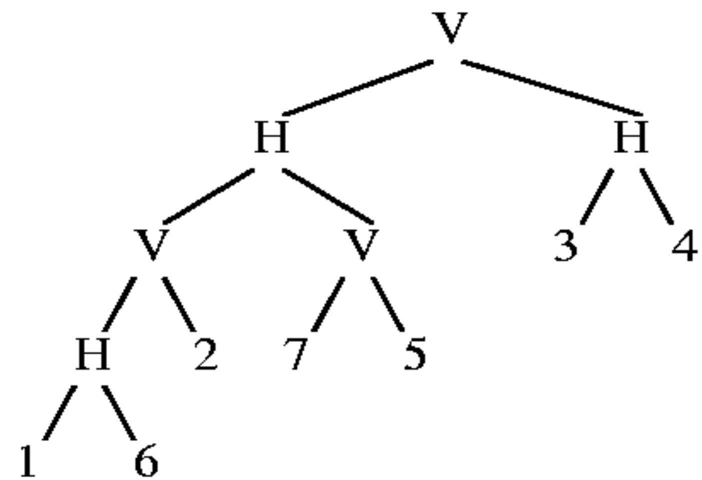


- **Question**: how to eliminate redundant representations?

Normalized Polish Expression

- A Polish expression $E = e_1 e_2 \ldots e_{2n\text{-}1}$ is called **normalized** iff E has no consecutive operators of the same type (H or V).
- Given a **normalized** Polish expression, we can construct a **unique** rectangular slicing structure.





$E = 16H2V75VH34HV$

A normalized Polish expression

Neighborhood Structure

- **Chain**: *HVHVH*…or *VHVHV*…

$$1\ \ 6\ \ \boxed{H}\ \ 3\ \ 5\ \ \boxed{V}\ \ 2\ \ \boxed{H\ \ V}\ \ 7\ \ 4\ \ \boxed{H\ \ V}$$

chain

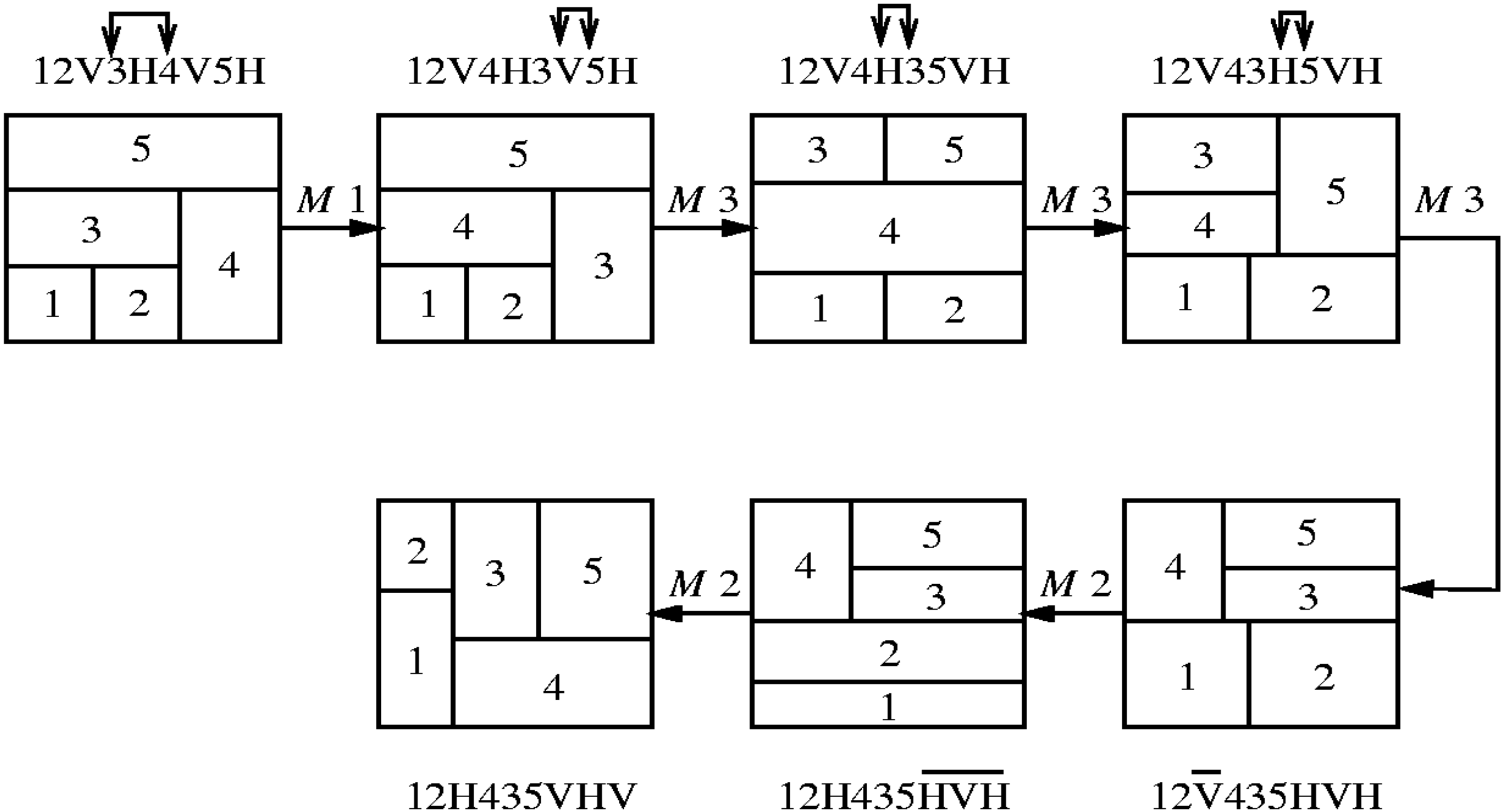
- **Adjacent**: 1 and 6 are adjacent operands; 2 and 7 are adjacent operands; 5 and *V* are adjacent operand and operator.
- 3 types of moves:

  - M_1 (**Operand Swap**): swap two adjacent operands.

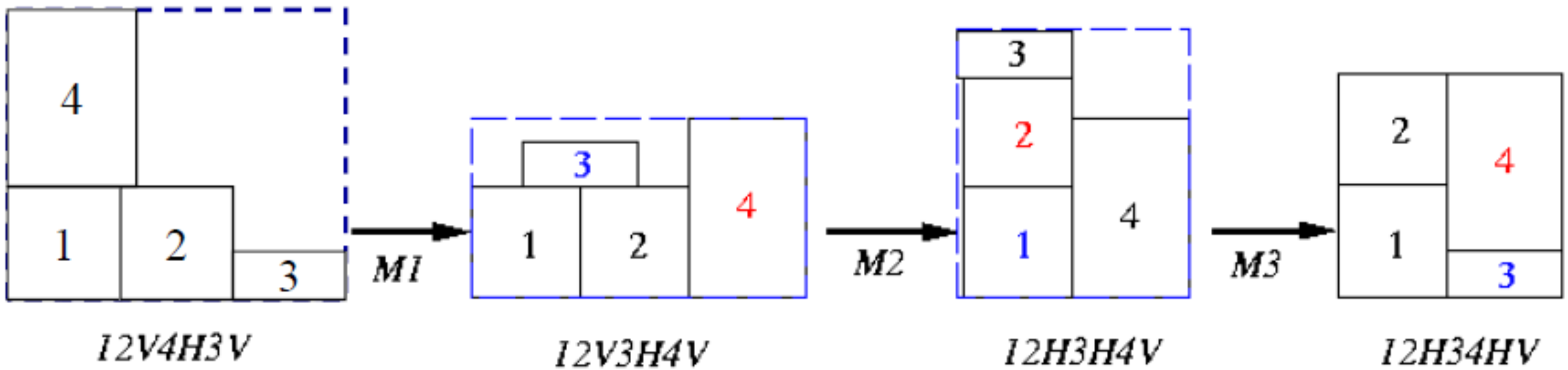
  - M_2 (**Chain Invert**): Complement a chain. $(\overline{V} = H, \overline{H} = V)$

  - M_3 (**Operator/Operand Swap**): Swap two adjacent operand and operator.
- It can be proved that each normalized Polish expression can be obtained from any other one through a finite set of moves of the above three types.

Solution Perturbation



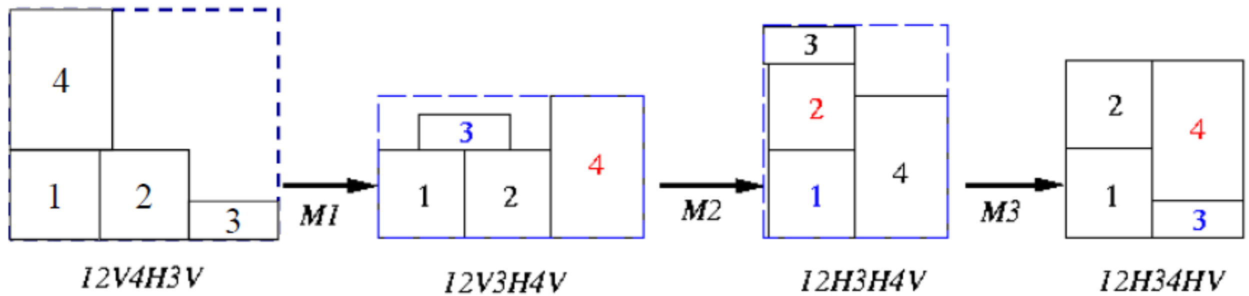
Effects of Perturbation



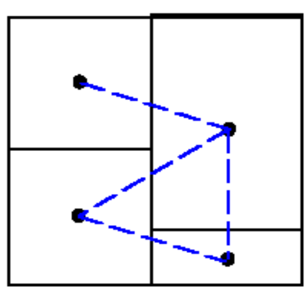
- **Question**: Does the balloting property hold during the moves?
  - M_1 and M_2 moves are OK.
  - Check the M_3 moves! Reject "illegal" M_3 moves.
- **Check M_3 moves**: Assume that the M_3 move swaps the operand e_i with the operator e_{i+1}, $1 \leq i \leq 2n\text{-}2$. Then, the swap will not violate the balloting property iff $2N_{i+1} < i$.
  - N_k: # of operators in the Polish expression $E = e_1 e_2 \ldots e_k$, $1 \leq k \leq 2n\text{-}1$.

Cost Function

- $\Phi = A + \lambda W$
  - A: area of the smallest rectangle
  - W: overall wiring length
  - λ: user-specified parameter

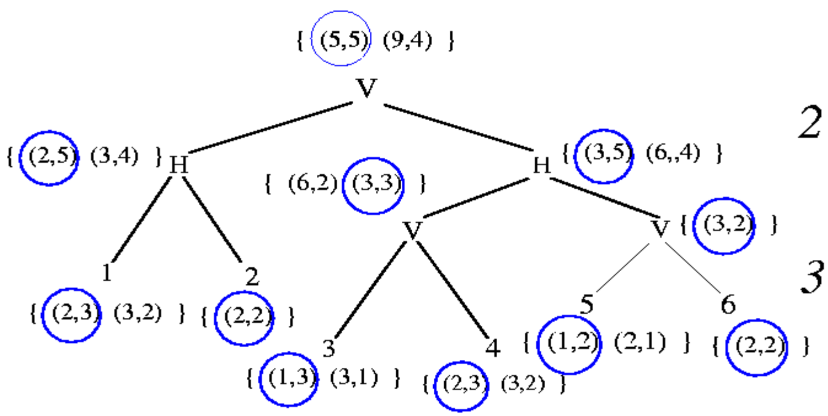


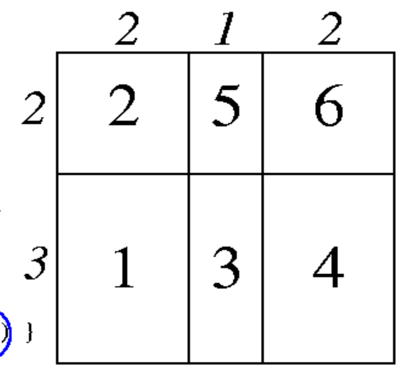
- $W = \sum_{ij} c_{ij} d_{ij}$
  - c_{ij}: # of connections between blocks i and j.
  - d_{ij}: center-to-center distance between basic rectangles i and j.

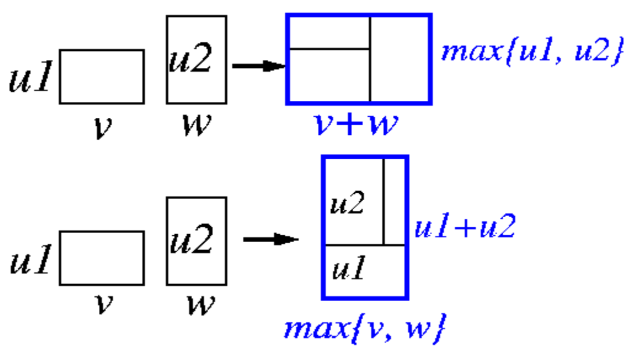


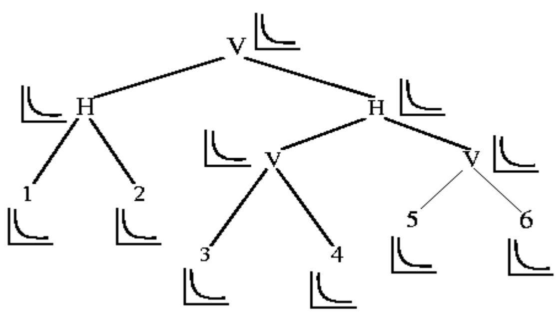
Area Computation for Hard Blocks

- Stockmeyer, "Optimal orientations of cells in slicing floorplan designs," *Information and Control*, 1983.
- Time complexity: $O(knd)$, where n is # modules, d is the depth of tree, and each module has $O(k)$ possible shapes.







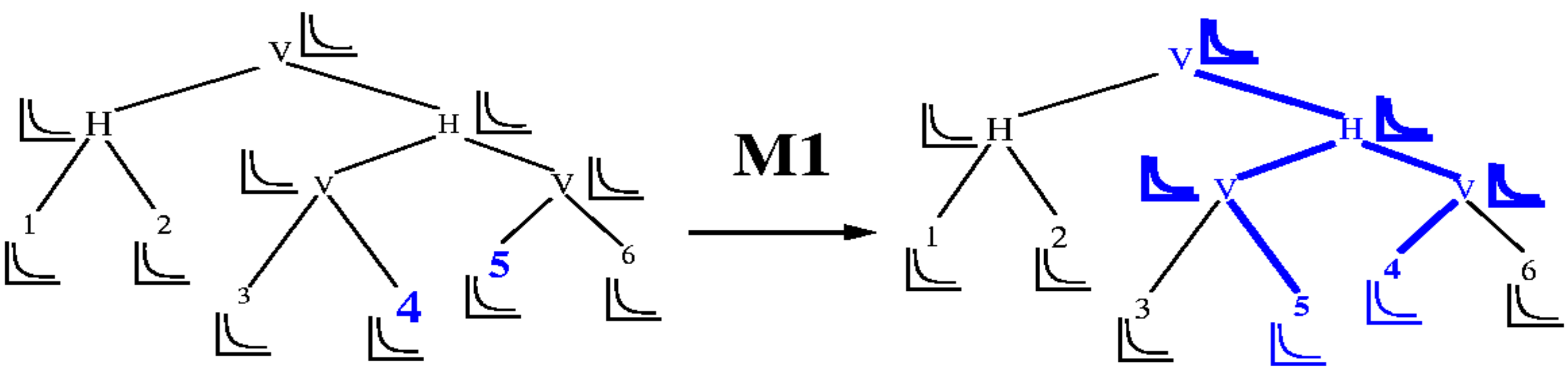


Slicing Floorplan Sizing

- The shape function of each leaf block is given as a staircase (or piecewise linear) function.
- Traverse the slicing tree to compute the shape functions of all composite blocks (bottom-up composition).
- Choose the desired shape of the top-level block
  - Only the corner points of the function need to be evaluated for area minimization.
- Propagate the consequences of the choice down to the leaf blocks (top-down propagation).

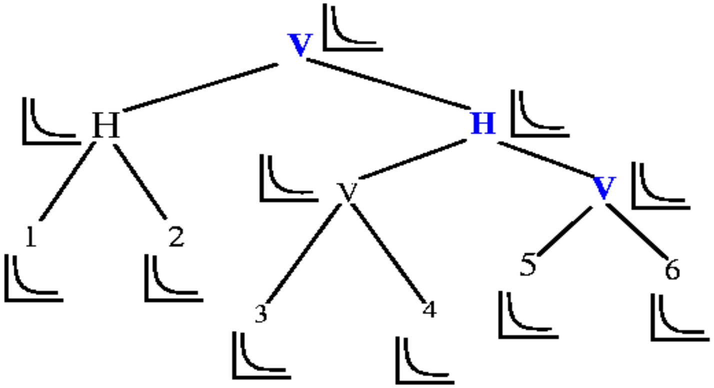
Incremental Area Computation

- Each move leads to only a minor modification of the Polish expression.
- At most **two paths** of the slicing tree need to be updated for each move.

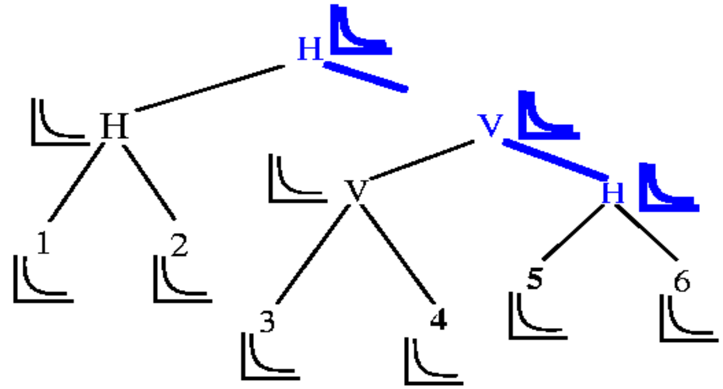


E = 12H34V56VHV **E = 12H35V46VHV**

Incremental Area Computation (cont'd)

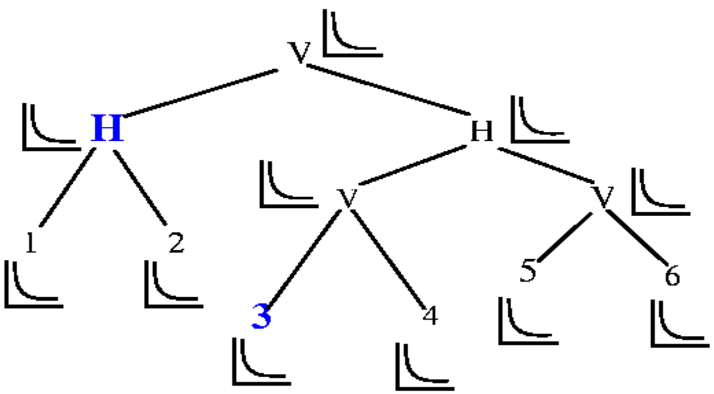


M2

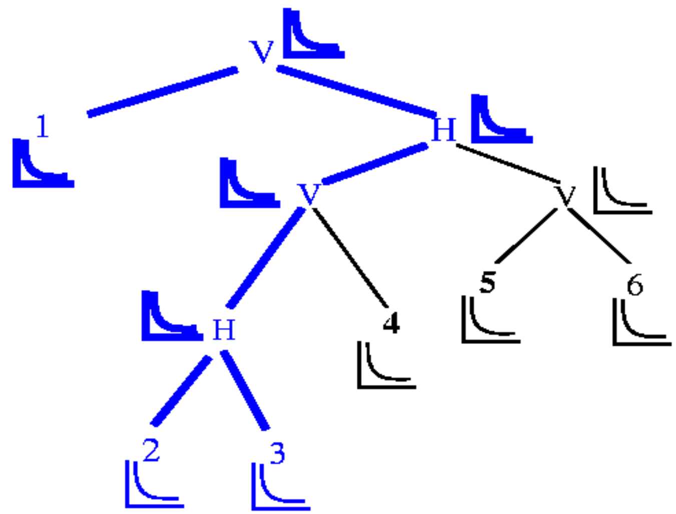


E = 12H34V56VHV

E = 12H34V56HVH



M3



E = 12H34V56VHV

E = 123H4V56VHV

Annealing Schedule

- Initial solution: $12V3V\ldots nV$

- $T_i = r^i T_0$, $i = 1, 2, 3, \ldots$; $r = 0.85$
- At each temperature, try kn moves (k = 5-10)
- Terminate the annealing process if
  - \# of accepted moves < 5%
  - Temperature is low enough, or
  - Run out of time.

Algorithm: Simulated_Annealing_Floorplanning(P, ϵ, r, k)

begin
$E \leftarrow 12V3V4V \ldots nV$; /* initial solution */
$Best \leftarrow E$; $T_0 \leftarrow -\frac{\Delta_{avg}}{ln(P)}$; $M \leftarrow MT \leftarrow uphill \leftarrow 0$; $N = kn$;
repeat
    $MT \leftarrow uphill \leftarrow reject \leftarrow 0$;
    repeat
        SelectMove(M);
        Case M of
        M_1: Select two adjacent operands e_i and e_j; $NE \leftarrow Swap(E, e_i, e_j)$;
        M_2: Select a nonzero length chain C; $NE \leftarrow Complement(E, C)$;
        M_3: $done \leftarrow FALSE$;
            while not $(done)$ **do**
                Select two adjacent operand e_i and operator e_{i+1};
                if $(e_{i-1} \neq e_{i+1})$ **and** $(2N_{i+1} < i)$ **then** $done \leftarrow TRUE$;
            $NE \leftarrow Swap(E, e_i, e_{i+1})$;
        $MT \leftarrow MT + 1$; $\Delta cost \leftarrow cost(NE) - cost(E)$;
        if $(\Delta cost \leq 0)$ **or** $(Random < e^{\frac{-\Delta cost}{T}})$
        then
            if $(\Delta cost > 0)$ **then** $uphill \leftarrow uphill + 1$;
            $E \leftarrow NE$;
            if $cost(E) < cost(best)$ **then** $best \leftarrow E$;
        else $reject \leftarrow reject + 1$;
    until $(uphill > N)$ **or** $(MT > 2N)$;
    $T = rT$; /* reduce temperature */
until $(\frac{reject}{MT} > 0.95)$ **or** $(T < \epsilon)$ **or** $OutOfTime$;
end