Floorplanning and Placement
(4541.554 Introduction to Computer-Aided Design)

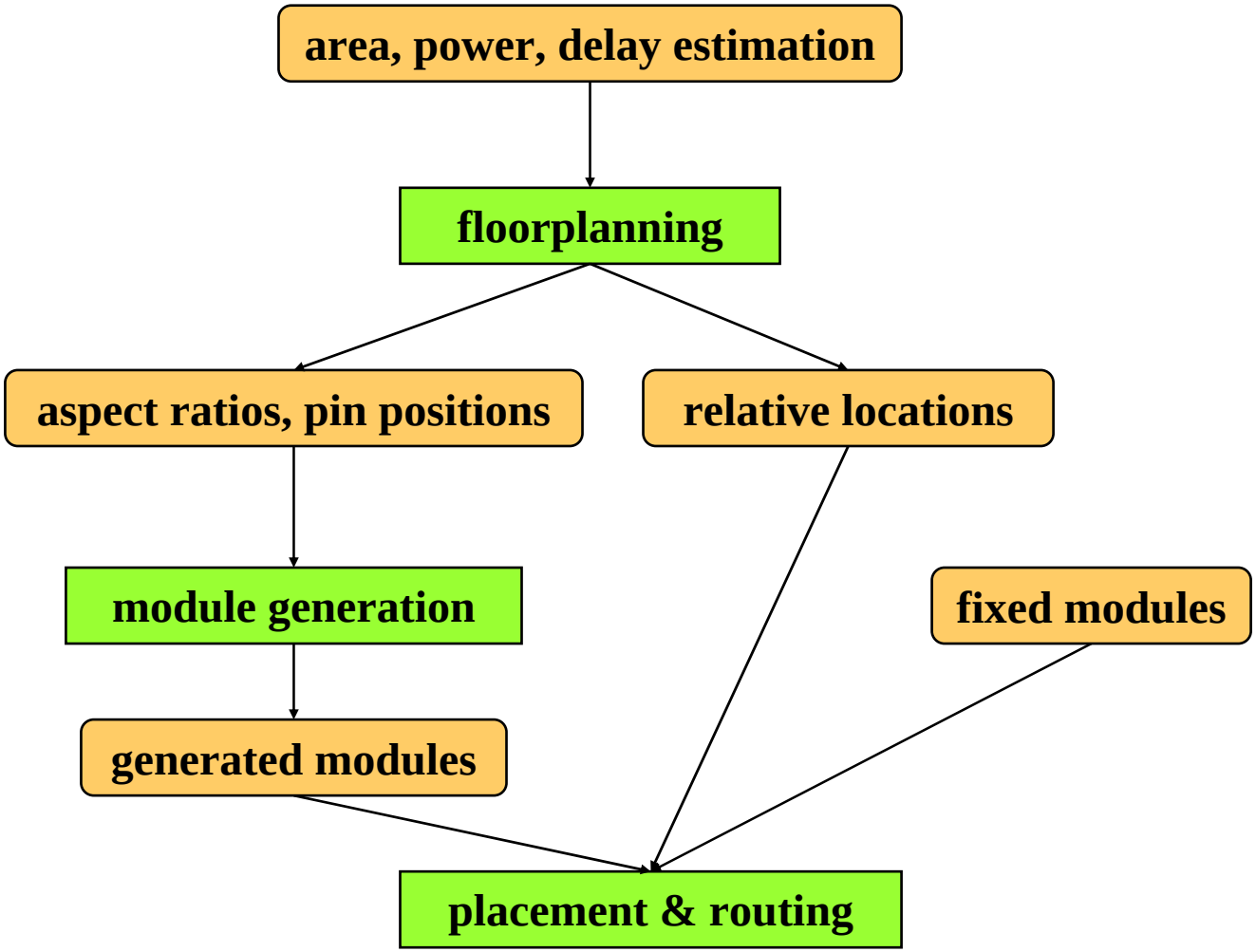
School of EECS

Seoul National University

Floorplanning and Placement

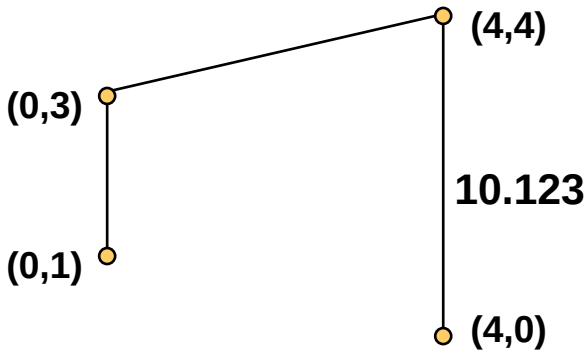
- **Floorplanning**
  - **Determines relative module position minimizing cost (e.g. total wiring length) and satisfying constraints (e.g. total area, layout aspect ratio, timing)**
  - **Also determines geometry and pin positions for all modules having flexibility in aspect ratios and pin positions (e.g. PLAs or modules built of standard cells)**
  - **Needs estimation of area, power, and speed**
  - **Guideline for module generation, placement, and routing**
- **Placement**
  - **Determines absolute positions of modules with fixed geometry minimizing cost (e.g. total wiring length) and satisfying constraints (e.g. area, aspect ratio, timing)**

- **Example of layout synthesis flow**

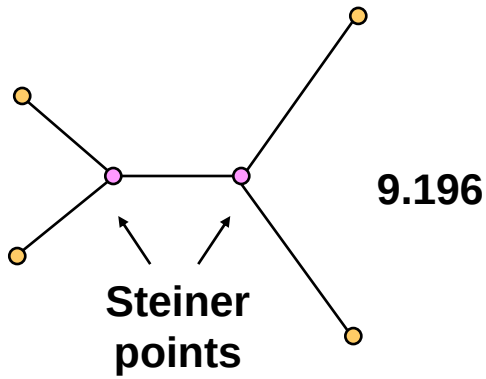


Cost Function

- **We may use**
  - **Total net length**
  - **Critical net length**
  - **Congestion**
  - **Area (estimated routing area)**
- **Net length calculation**
  - **Euclidean Steiner tree problem**



Shortest spanning tree

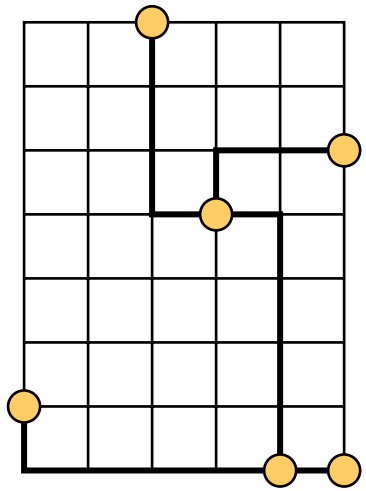


Shortest Steiner tree

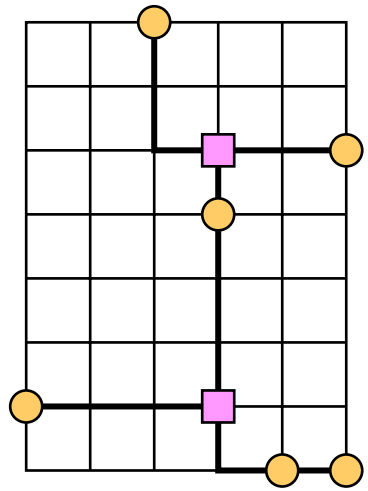
- **Assuming Manhattan geometry**
  - **d(a,b)=|x(a)-x(b)|+|y(a)-y(b)|**
- **Problem**
  - **Find a rectilinear interconnection of minimum length for a single n-pin net (Rectilinear Steiner tree problem)**

  --> NP-hard
- **When there is grid**
  - **Is there a finite set V' of (grid) points such that the tree spanning (terminals+V') has total length less than K?**

  --> NP-complete

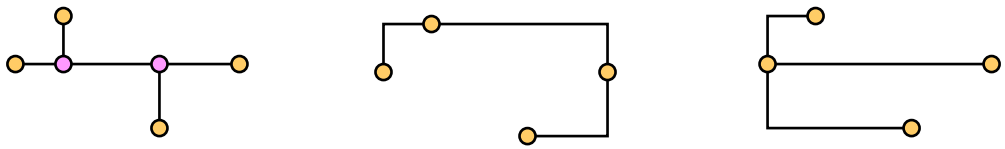


18

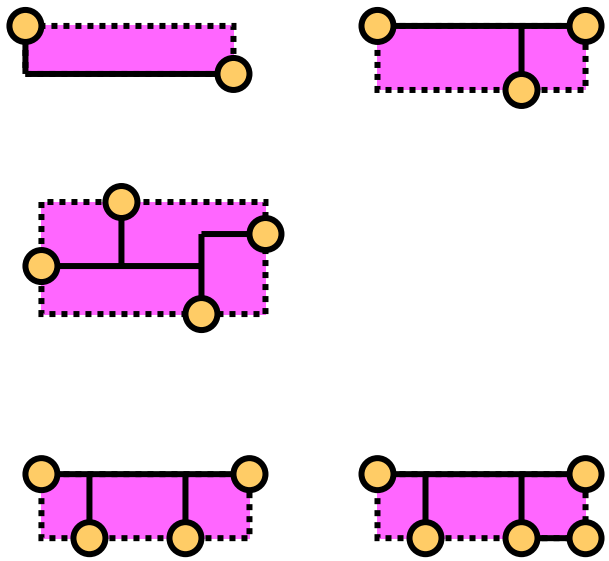


15

- **Approximation**
  - **Spanning tree**
  - **Half-perimeter of bounding box**
  - **Sum of distances**
  - **Sum of distance squares**
- **Shortest spanning tree**
  - **No vertices other than the pins**
  - **Prim's algorithm: $O(|V|^2)$, $O(|E|\log|V|)$**
  - **Kruskal's algorithm: $O(|E|\log|E|)$**



- **Half-perimeter of the smallest bounding box**
  - **Exact for 2-3 pin nets**
  - **Lower bound on the length of the shortest rectilinear Steiner tree**
  - **1/2 upper bound for 4-5 pin nets (bounding box is an upper bound)**



- **Sum of distances**

$$\sum_{p_i, p_j \in P_t} d_{ij}$$

where P_t: set of pins of net n_t

d_{ij}: Manhattan distance or Euclidean distance of p_i and p_j

--> assume net interconnection forms a complete graph

- **Sum of distance squares**

$$\sum_{p_i, p_j \in P_t} d_{ij}$$

where P_t: set of pins of net n_t

d_{ij}: square of Euclidean distance

$d_{ij}=(x(i)-x(j))^2 +(y(i)-y(j))^2$

- **Modules are represented by points to reduce computation**

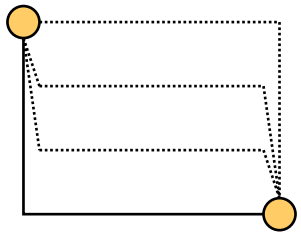
$$\sum_{m_i, m_j, i \neq j} c_{ij} d_{ij}$$

where c_{ij}: number of nets which connects modules m_i and m_j

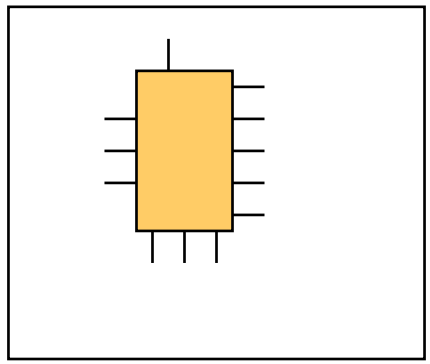
- **Area utilization**
  - **Module area + routing area**
  - **Gate arrays**
    - **Penalize placements that yield routing densities larger than channel capacities**
    - **Routing congestion is used in the cost function**
    - **Assumptions for simplifying routing congestion estimation:**
      - **A net is routed entirely in the smallest bounding box**
      - **Routing alternatives**

        --> probability

        --> summation of the probabilities for each routing region gives an estimation of the congestion

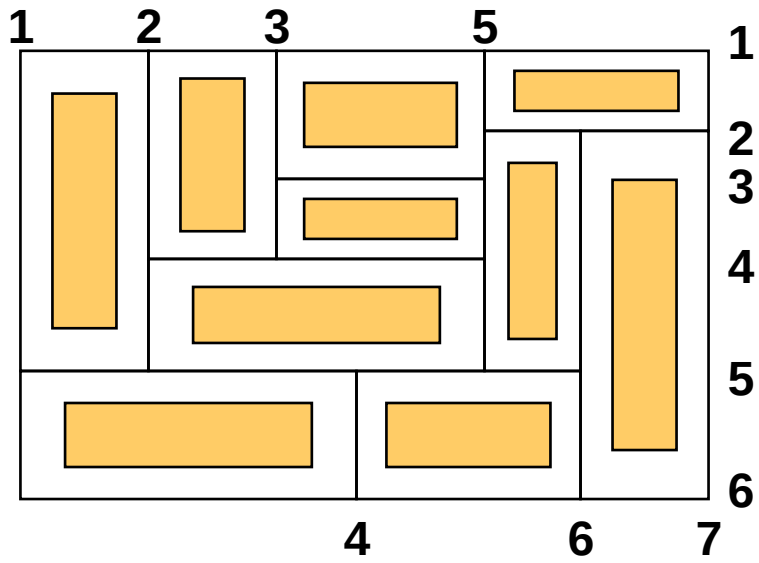


- **Standard cells**
  - **Minimize sum of routing densities**
- **Macro cells**
  - **Estimate routing area by inflating cells**

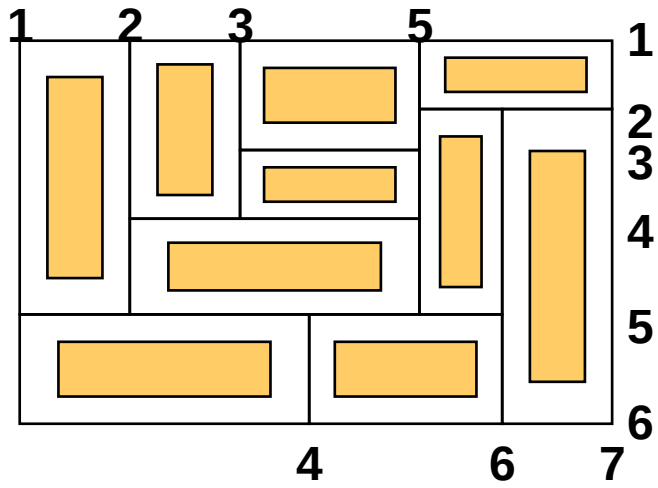


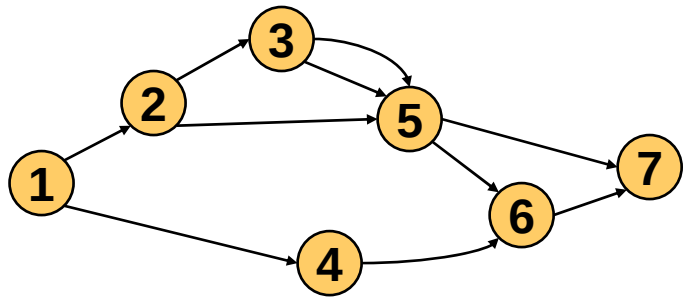
Data Structure to Represent Layout

- **Rectangle dissection**

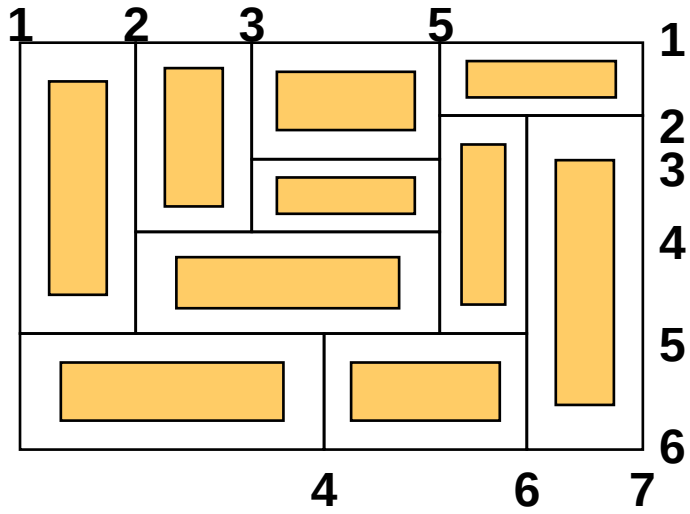


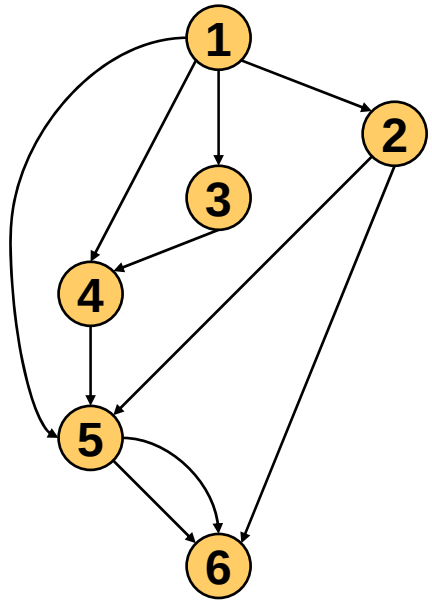
- **Horizontal polar graph $G_H(V,E)$**
  - **V: Vertical boundaries**
  - **E: Set of directed edges**
  - **Edge (i,j): There is a module which has left boundary at i and right boundary at j**
  - **module <--> edge**
  - **Source: Left-most vertical boundary**
  - **Sink: Right-most vertical boundary**





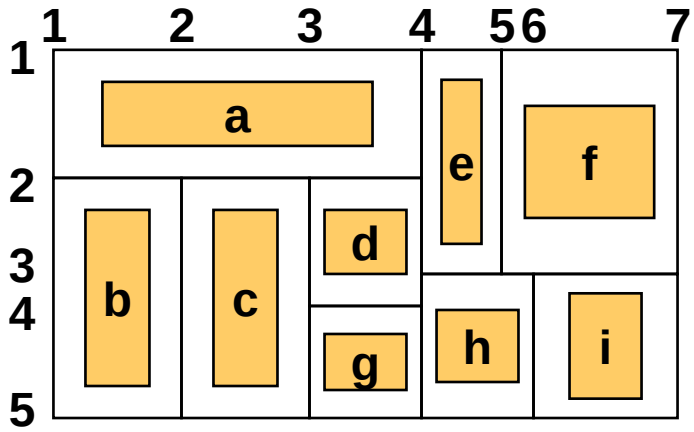
- **Vertical polar graph $G_V(V,E)$**

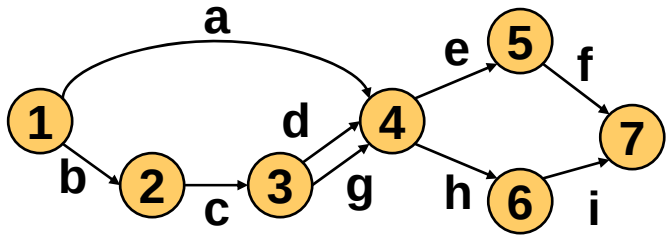


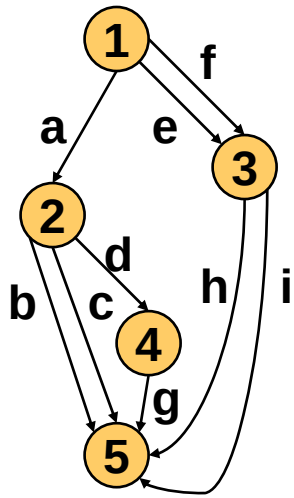


  - **Two polar graphs uniquely define a rectangle dissection.**
  - **Label the edges with minimum distances**
    - **Critical path between source and sink gives minimum x and y dimensions**
    - **Similar to compaction problem**

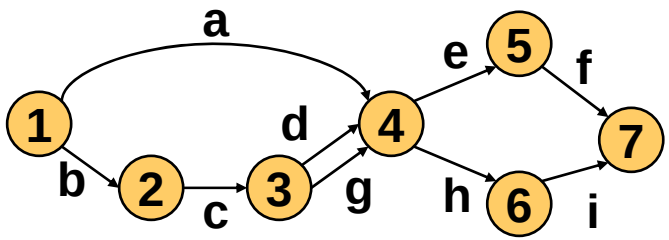
- **Slicing structures**
  - **Special case of rectangle dissection**
  - **Obtained by bisecting recursively**
  - **Series-parallel polar graphs**

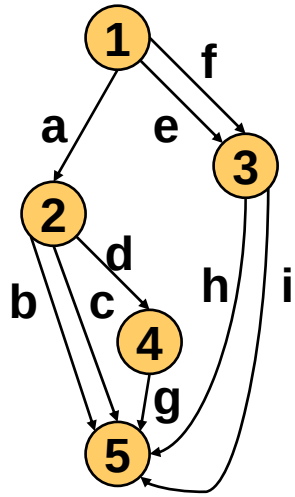


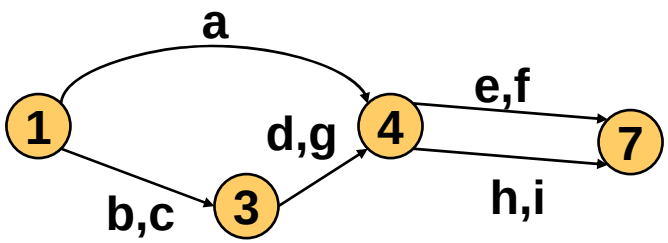


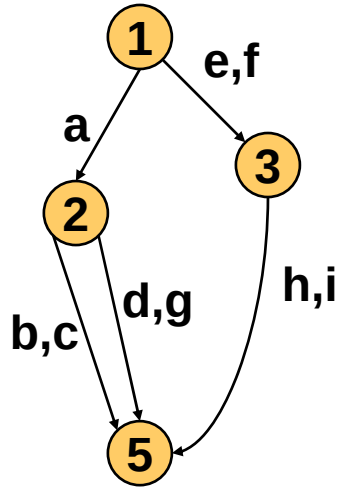


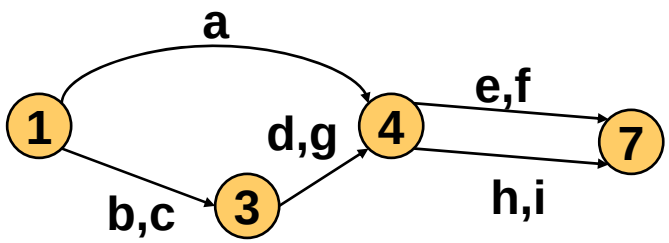
– **Reduction (dual reduction)**

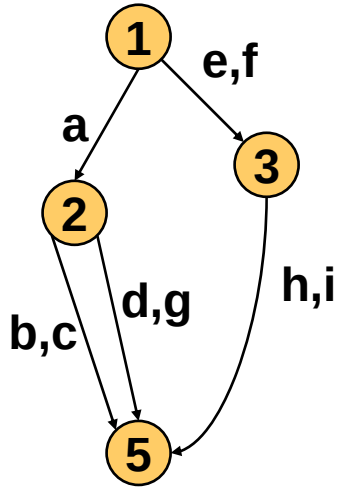


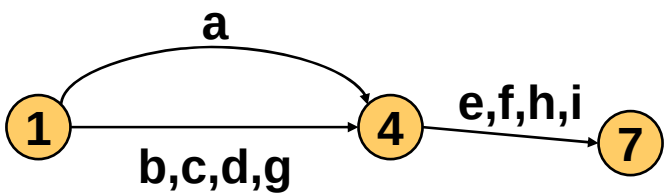


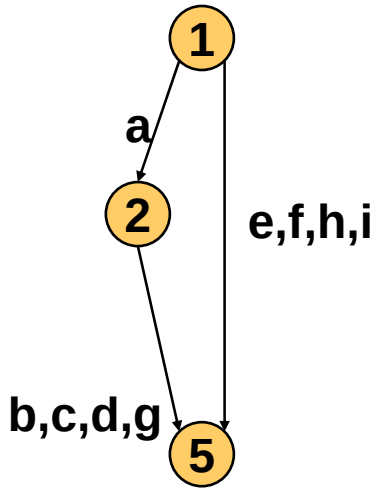


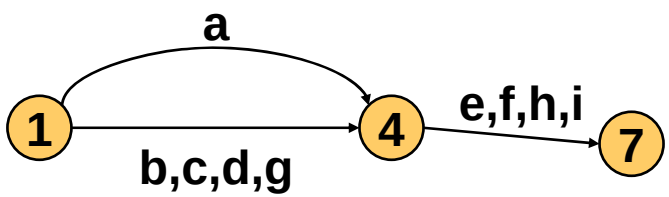


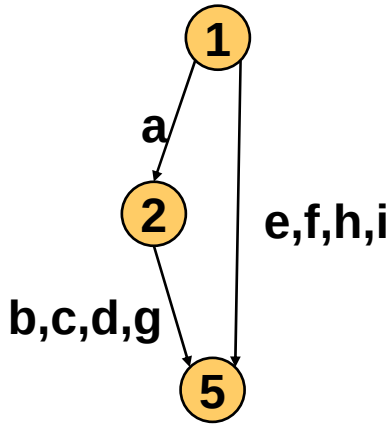

a
e,f
1
d,g
4
7
b,c
3
h,i

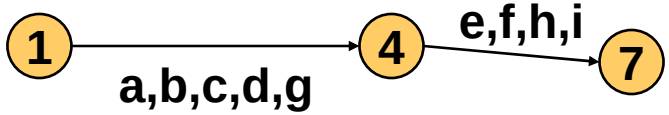

1
e,f
a
3
2
h,i
d,g
b,c
5

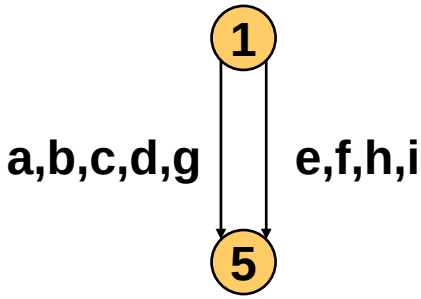

a
1
e,f,h,i
4
7
b,c,d,g


1
a
2
e,f,h,i
b,c,d,g
5

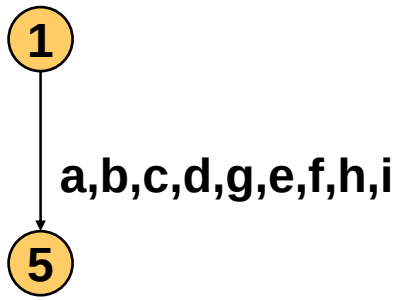

a
1
4
7
e,f,h,i
b,c,d,g


1
a
2
e,f,h,i
b,c,d,g
5

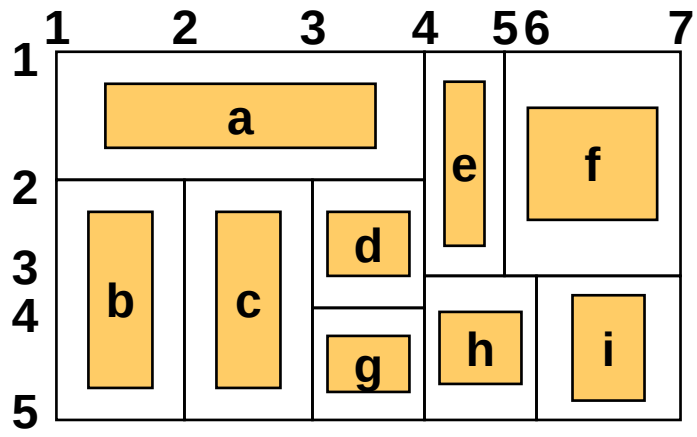

1
4
7
e,f,h,i
a,b,c,d,g

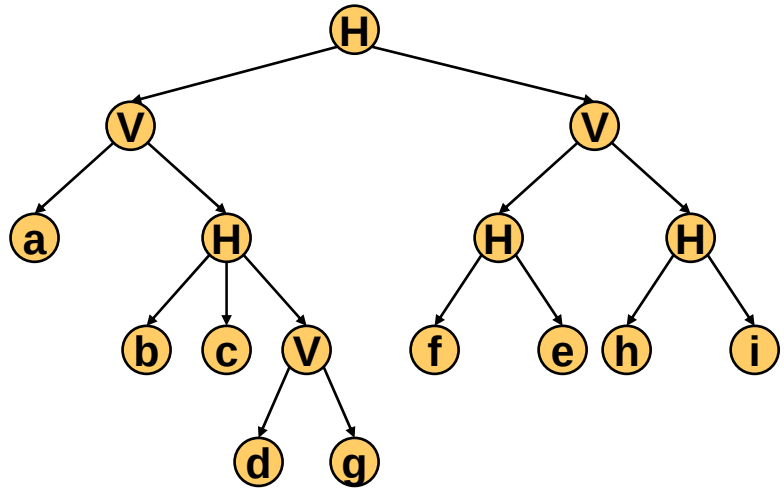

1
a,b,c,d,g
e,f,h,i
5


1
7
a,b,c,d,g,e,f,h,i

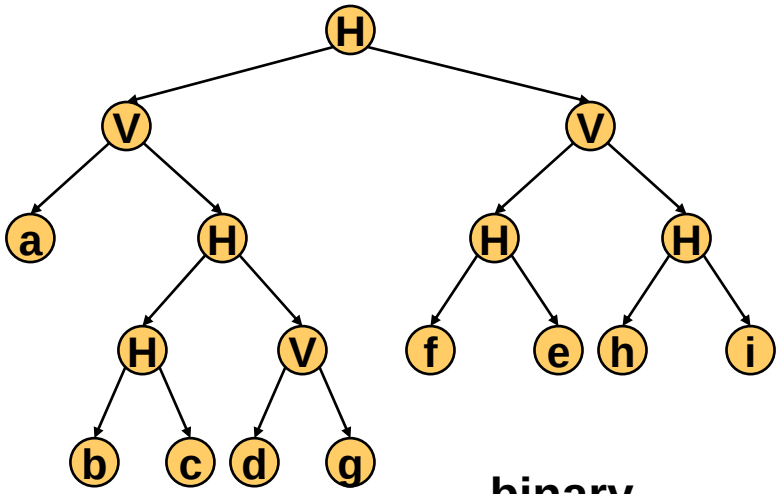

1
a,b,c,d,g,e,f,h,i
5

– **Decomposition tree**

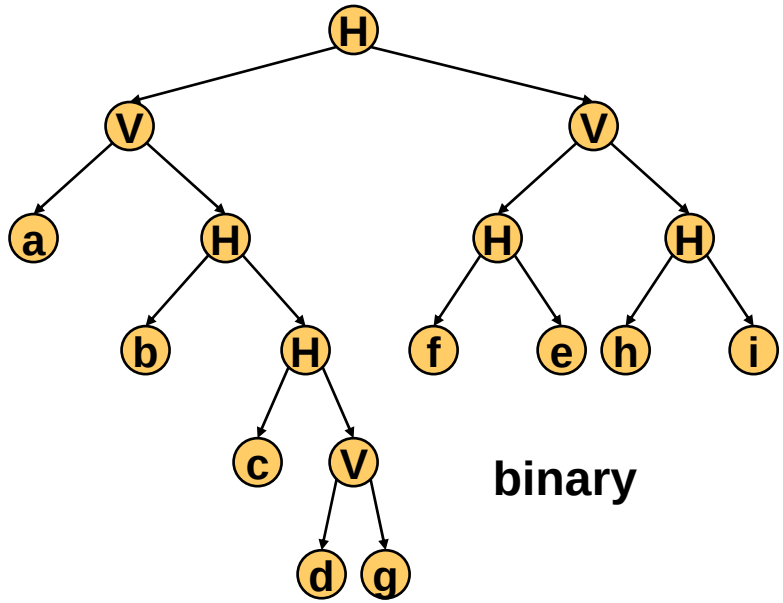




non-binary, unique



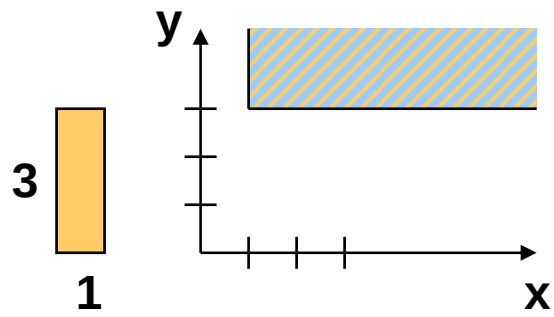
binary



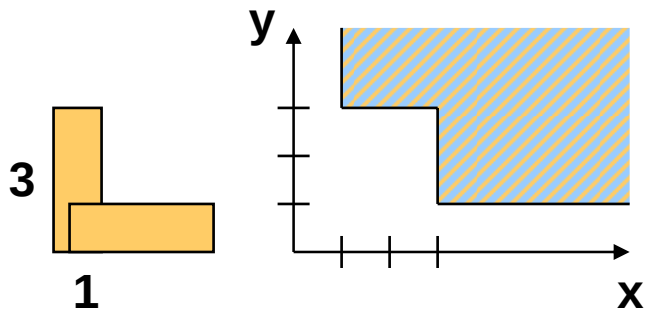
binary

Shape Constraint

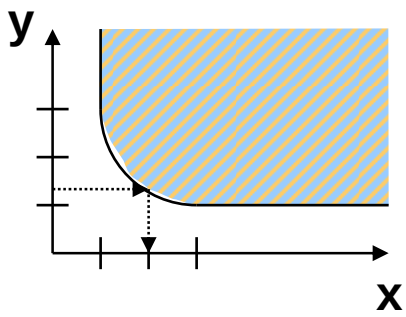
- {(x,y)|y>=h,x>=w}



- Module can be rotated by 90 degrees.
  - {(x,y)|y>=h,x>=w or y>=w,x>=h}



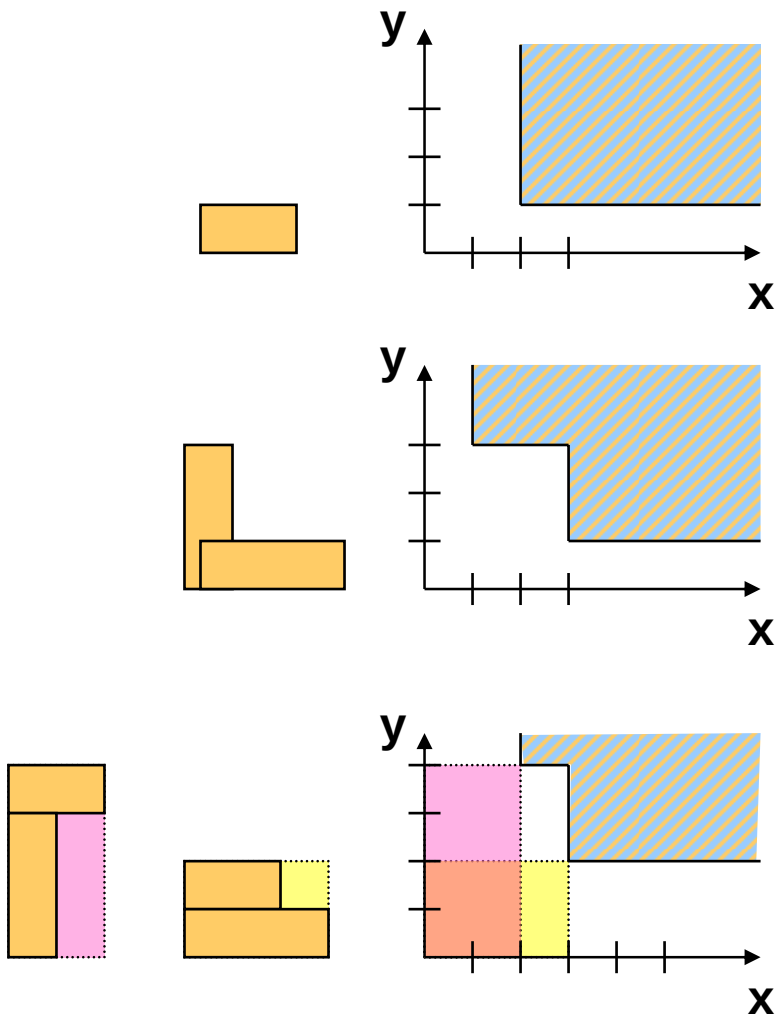
- Continuously varying aspect ratio
  - {(x,y)|xy>=A,y>=h,x>=w}



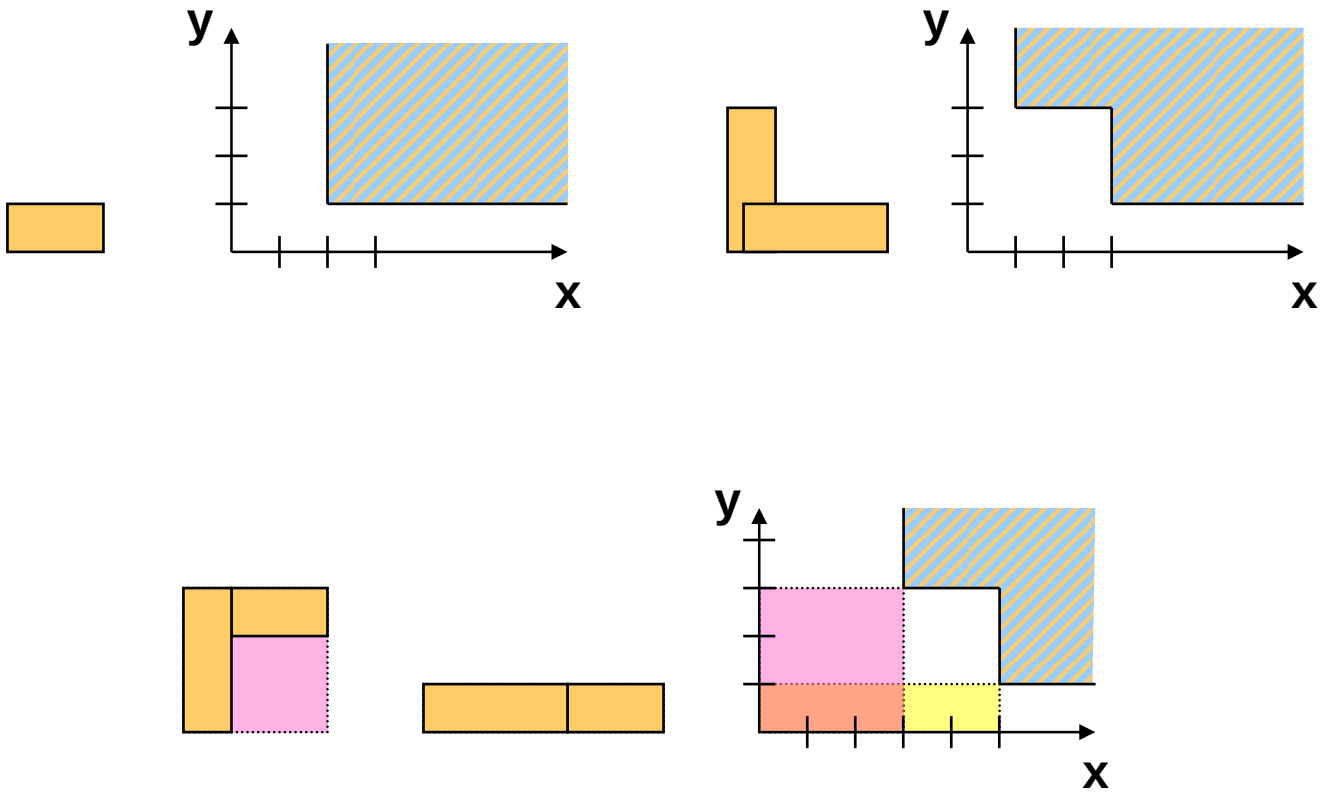
- One dimension is fixed

--> Use shape constraint to optimize the other dimension

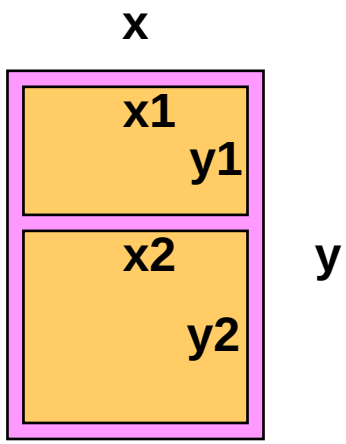
- **Composition of shape constraints**



- **Composition of shape constraints**



- **Problem: given dimension of enclosing rectangle, find proper size of two rectangles enclosed**
- **Solution example:**

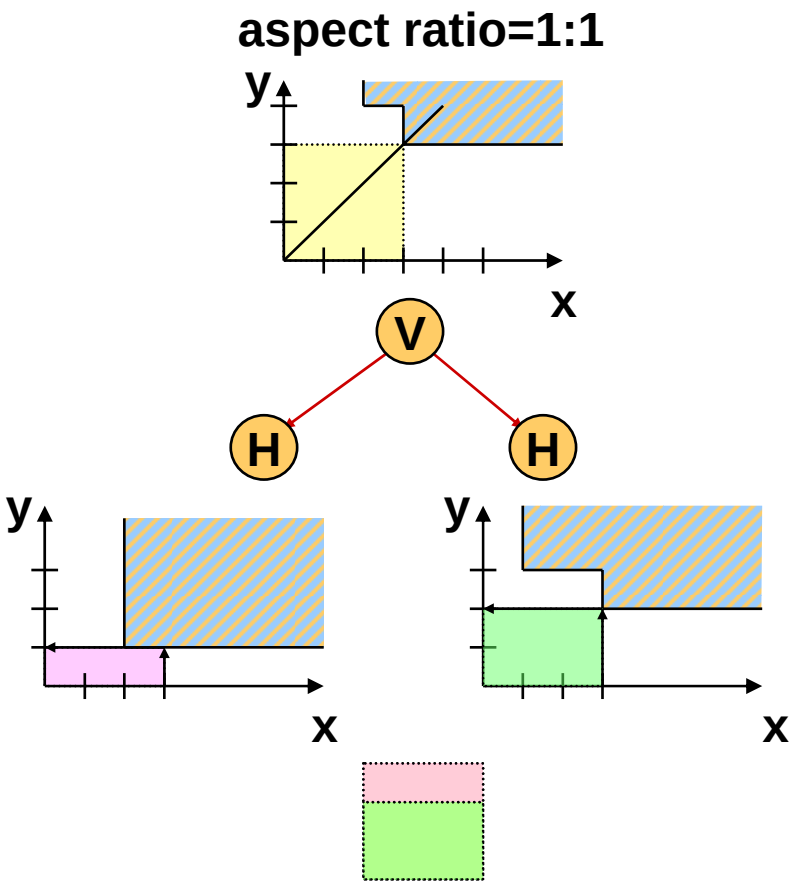


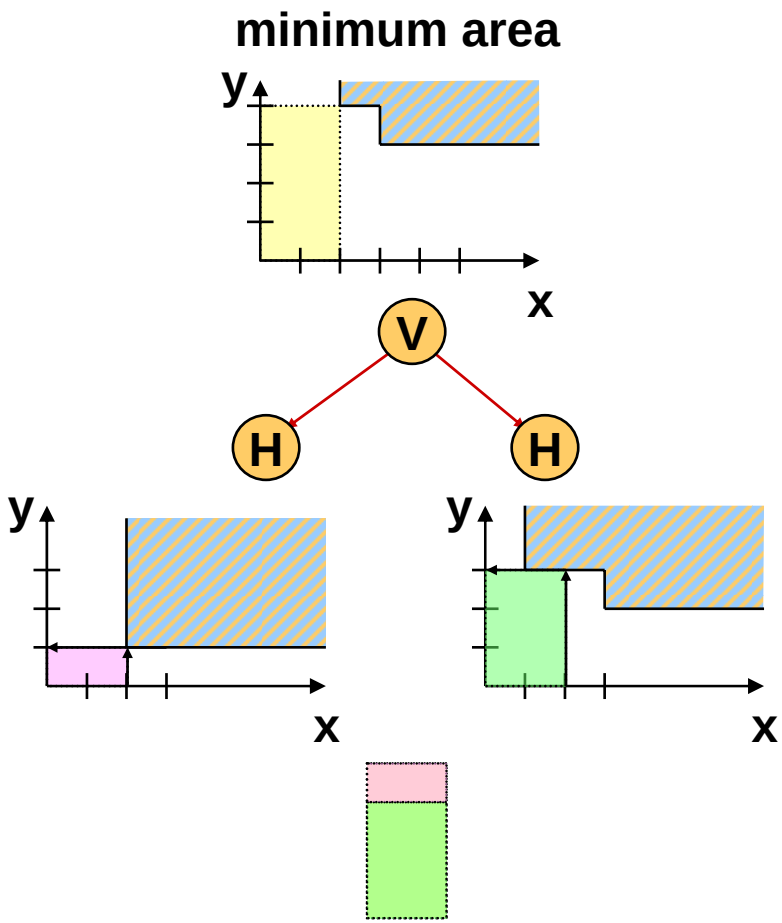
x1=x2=x

Obtain y1 and y2 from their own shape constraints

- **Problem: determine minimum area layout satisfying constraints on aspect ratio, width, and/or height**
- **Solution:**
  - **Step1: shape constraints are composed bottom-up according to the decomposition tree**

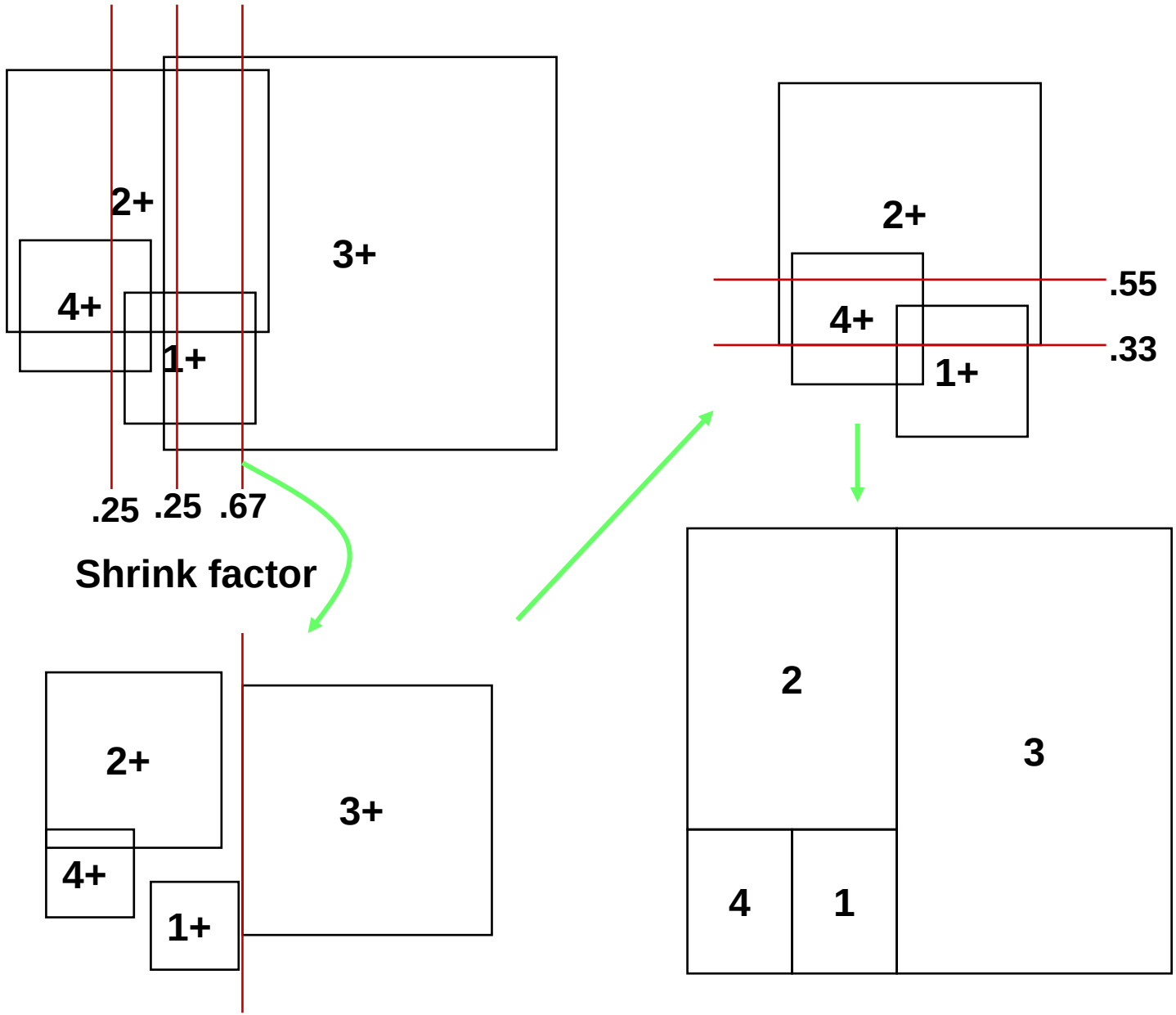
    --> shape constraint of the entire chip
  - **Step2: from the shape constraint, obtain minimum area boundary points satisfying given constraints (chip aspect ratio, width, height, etc.)**
  - **Step3: traversing down from the root of the decomposition tree, determine the module dimension (each child slice inherits one dimension from the ancestor and the other dimension is derived from the shape constraints)**


aspect ratio=1:1
y
x
V
H
H
y
x
y
x


minimum area
y
x
V
H
H
y
x
y
x

- **Generation of slicing structures**
  - **Assume modules are points**
  - **Obtain a placement which optimizes net length (e.g. min-cut, simulated annealing)**
  - **Transform the placement into a slicing structure (consider module size)**
    - **Step1: replace each module by a square whose center is at the center of the module and whose area is proportional to the estimated area of the module**
    - **Step2: shrink uniformly**
    - **Step3: determine slicing lines**
    - **Step4: bisect**

Shape Constraint



Algorithms

- **Constructive**
  - **Some (or all) components are unplaced**
  - **Assign unplaced components to positions**
  - **Used for initial placement**
  - **Cluster growth**

    Min-cut

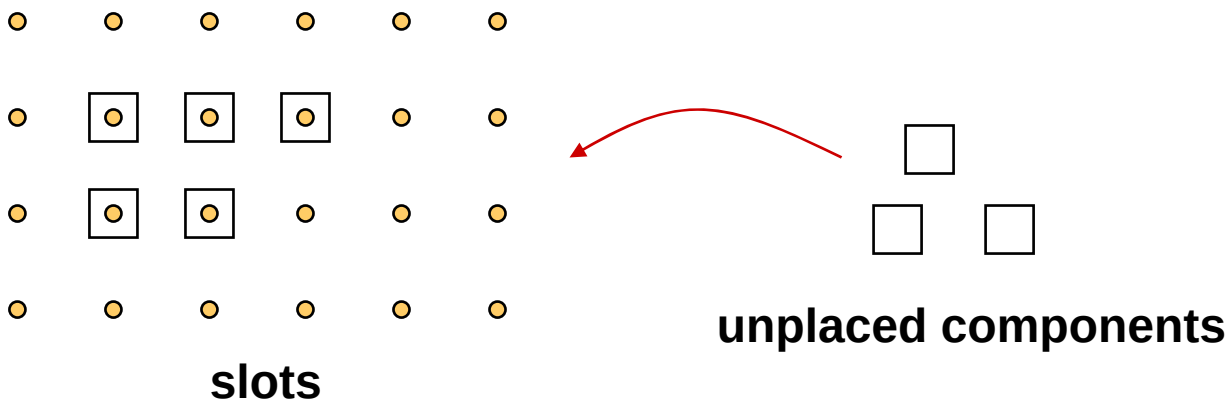
    Global placement (constructive force-directed)

    Branch and bound
- **Iterative**
  - **Start with a complete placement**
  - **Attempt to derive a better placement**
  - **Force-directed**

    Simulated annealing
    Genetic algorithm

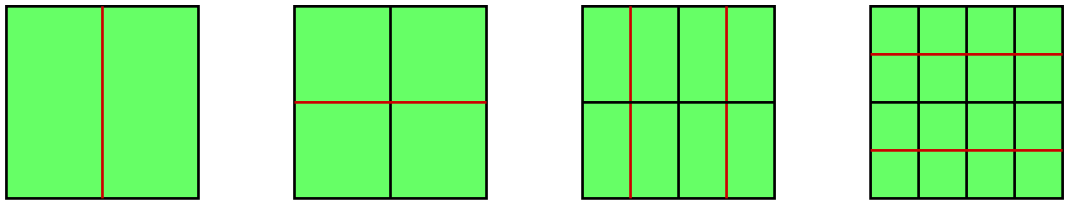
Cluster Growth

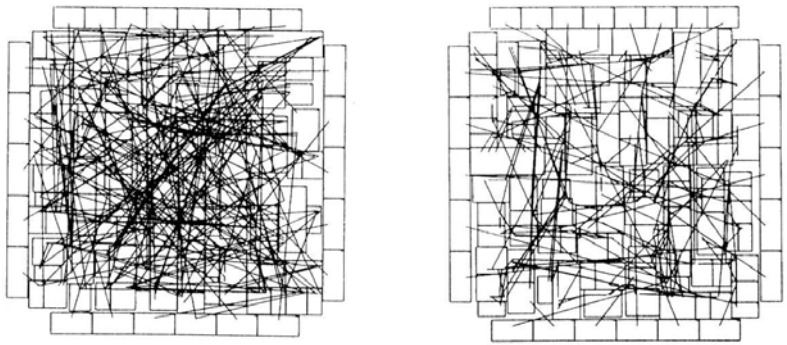
- **Seed component(s) are determined by the user to guide the placement process or determined algorithmically (e.g. one-terminal cell)**
- **Select one of the unplaced components that is most strongly connected to all of the placed components (or to any single placed component)**
- **Place to a position with minimum net length (or as closely to the strongly connected mate as possible)**



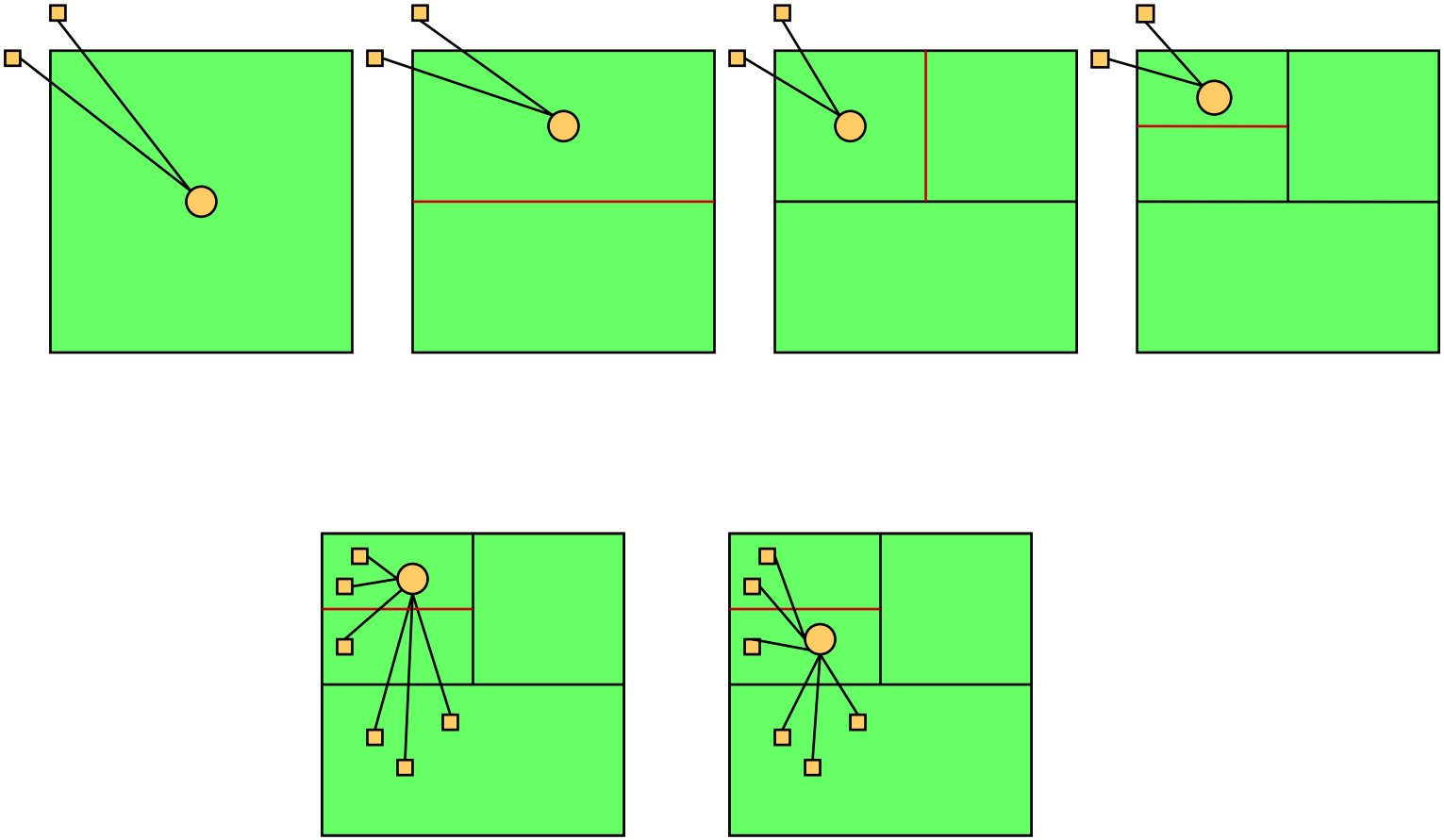
Min-Cut

- Divide modules recursively into subsets with (approximately) equal area and such that the number of nets crossing the cut is minimized
- Use Kernighan-Lin partitioning algorithm
- Leads directly to a slicing structure
- Not optimal because net information is global (not local to a subset)

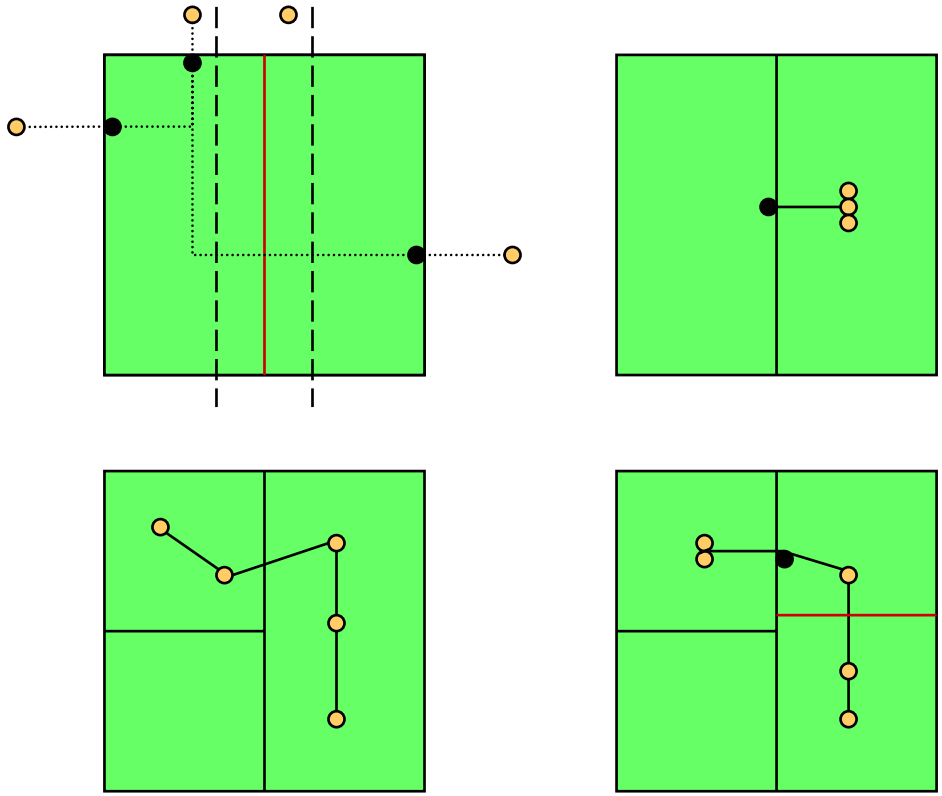




- **Terminal propagation (In-place partitioning)**

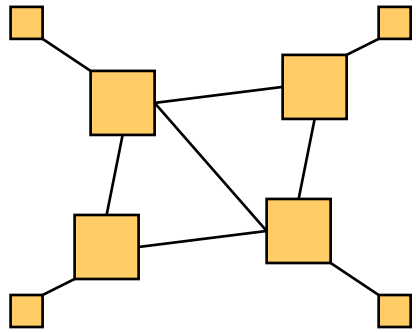


- **Step1: divide the set of all external modules connected to the modules in the set to be partitioned into two sets according to their location**
- **Step2: replace the external modules with internal dummy modules**
- **Step3: partition**



Constructive Force-Directed

- Global placement
- Consider each module as a point
- Replace each net by a spring
- Elastic constant=weight of the net
- Use Hooke's law analogy:
  - force = elastic constant * displacement
- Find equilibrium configuration:
  - Minimum energy
  - Sum of all forces = 0
- Need to consider pins on border (pad) to avoid collapse



- **$C=[c_{ij}]$: interconnection matrix**

  D: diagonal matrix, $d_{ii}=\Sigma_j\ c_{ij}$

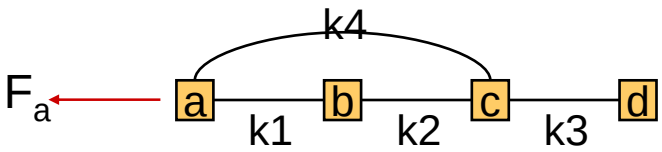
  At equilibrium

  $$F_x=Bx=0$$

  $$F_y=By=0$$

  where B=D-C

- **Example (1-dimensional)**



$$C=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & k1 & k4 & 0 \\ k1 & 0 & k2 & 0 \\ k4 & k2 & 0 & k3 \\ 0 & 0 & k3 & 0 \end{bmatrix} \end{array}$$

$$D=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} k1+k4 & 0 & 0 & 0 \\ 0 & k1+k2 & 0 & 0 \\ 0 & 0 & k2+k3+k4 & 0 \\ 0 & 0 & 0 & k3 \end{bmatrix} \end{array}$$

$$F_a=k1(x_a-x_b)+k4(x_a-x_c)$$
$$=(k1+k4)x_a-k1x_b-k4x_c$$

B is singular

--> $x_a=x_b=x_c=x_d=t$ is a solution

--> collapse

$$F=Bx=(D-C)x=\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} k1+k4 & -k1 & -k4 & 0 \\ -k1 & k1+k2 & -k2 & 0 \\ -k4 & -k2 & k2+k3+k4 & -k3 \\ 0 & 0 & -k3 & k3 \end{bmatrix} \end{array}\begin{bmatrix} x_a \\ x_b \\ x_c \\ x_d \end{bmatrix}=0$$

- **Eigenvalue approach**
  - **Problem: assign modules to a fixed set of positions (slots) minimizing the objective function**
  - **Objective function:**

$$F(x,y) = x'Bx + y'By$$

$$= \frac{1}{2}\sum_{i,j=1}^{n} c_{ij}((x_i - x_j)^2 + (y_i - y_j)^2) \text{ --- convex}$$

$$x'Bx = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} \sum_j c_{1j} & \dots & -c_{1n} \\ \vdots & \ddots & \vdots \\ -c_{n1} & \dots & \sum_j c_{nj} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$= \sum_i \sum_j c_{ij} x_i^{\,2} - \sum_i x_i \sum_j c_{ij} x_j \,, \quad c_{ii} = 0$$

$$= \frac{1}{2}\sum_{i,j=1}^{n} c_{ij}(x_i - x_j)^2$$

where c_{ij} is the number of interconnects between modules i and j

- **B is symmetric**

  $c_{ij}=c_{ji}$

- **B is positive semi-definite**

$$x' Bx = \frac{1}{2}\sum_{i,j=1}^{n} c_{ij}(x_i - x_j)^2 \geq 0 \quad (c_{ij} \geq 0)$$

$$\Rightarrow x' Bx = x'\lambda x \geq 0 \Rightarrow \lambda \geq 0$$

- **rank(B) = n-1**
  - **B is singular--> rank(B)< n**
  - **If the network is not partitioned into disjoint subsets, it collapses into one point. Assume x_n=0. Then**

$$B\begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ 0 \end{bmatrix} = 0$$

$$\Rightarrow x1 = \ldots = xn-1 = 0 \,(\text{collapses to } x = 0)$$

$$\Rightarrow \text{First } n-1 \text{ columns are linearly independent}$$

$$\Rightarrow \text{rank}(B) = n\text{-}1$$

- **Consider one dimension only (F(x) = x'Bx)**
  --> generalize to two dimensions
- **To have a legal placement (i.e. $x_i = l_i$),**

$$\sum x_i = \sum l_i$$
$$\sum x_i^{\,2} = \sum l_i^{\,2}$$
$$\ldots$$
$$\sum x_i^{\,n} = \sum l_i^{\,n}$$

where l_i are legal positions

This can be proved by proving that the condition implies

$$\prod_{i=1}^{n}(x + x_i) = \prod_{i=1}^{n}(x + l_i)$$

- **Problem**
  - **Find x values that minimize F(x)=x'Bx subject to x'x=L=Σl_i^2 (consider only the 2nd constraint)**
- **Solution**
  - **Use Lagrangian Λ= x'Bx - λ(x'x - L)**

    --> (B - λI)x = 0

    --> λ= x'Bx / x'x = F(x) / L

    --> F(x) is minimum for the smallest λ
  - **Solution of the problem is given by the eigenvector of matrix B associated with the smallest eigenvalue**
  - **B has rank n-1**

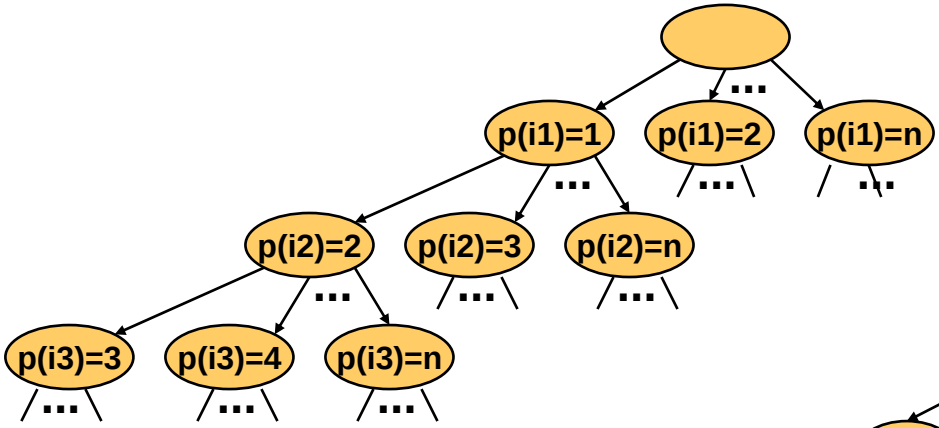
    --> Ignore the smallest eigenvalue λ= 0 (eigen vector [1 1 1 ... 1])

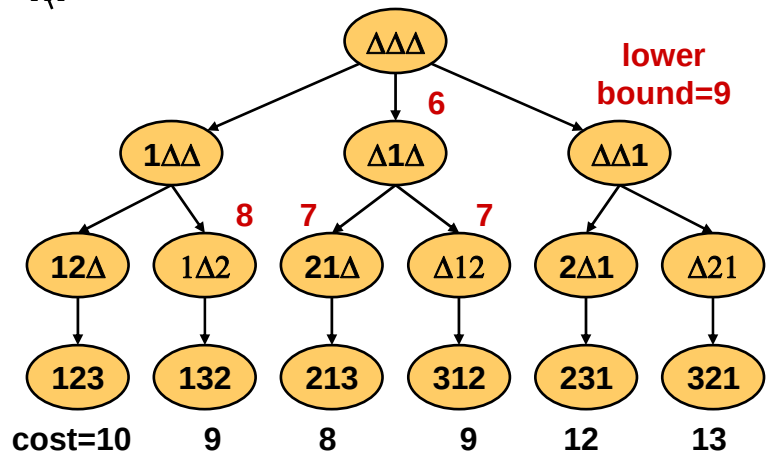
    --> The solution is the eigenvector corresponding to the 2nd smallest eigenvalue.
  - **Two dimensional placement**

    --> use 3rd smallest eigenvalue

Branch-and-Bound

- **Guarantee optimum results**
- **Runtime is excessive**
  - **Small placement problems**
  - **Heuristics are used**
- **Decision tree**





- To obtain lower bounds for $\Sigma c_{ij} d_{p(i)p(j)}$

  Step1: compute the contribution of all placed modules

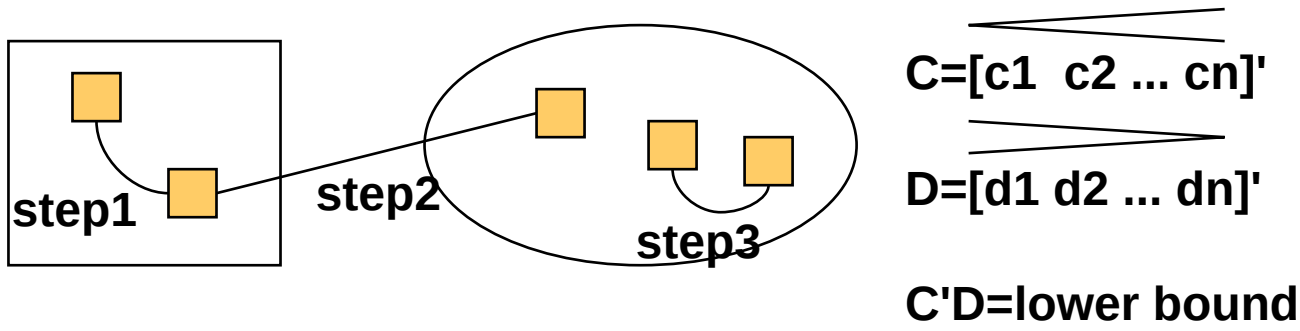
  Step2: compute the lower bound of the contribution due to the interconnection between placed modules and unplaced modules (when permutation is allowed, the minimum dot product of two vectors C and D is obtained by arranging C in ascending order and D in descending order)

  Step3: compute the lower bound of the contribution of the unplaced modules

- More accurate lower bound

  --> sharper pruning

  longer computation time for the lower bound



– Example

	1	2	3	4	5
1	–	7	4	5	8
2	7	–	4	3	0
3	4	4	–	3	7
4	5	3	3	–	4
5	8	0	7	4	–

connection matrix

	1	2	3	4	5
1	–	8	6	6	5
2	8	–	1	2	4
3	6	1	–	3	0
4	6	2	3	–	4
5	5	4	0	4	–

distance matrix

5ΔΔ3Δ

step1: $c_{53}d_{14}=(7)(6)=42$

step2: $(c_{52}\ c_{54}\ c_{51})(d_{12}\ d_{13}\ d_{15})=(0\ 4\ 8)(8\ 6\ 5)=64$

$(c_{34}\ c_{31}\ c_{32})(d_{45}\ d_{43}\ d_{42})=(3\ 4\ 4)(4\ 3\ 2)=32$

total=96

step3: $(c_{24}\ c_{14}\ c_{12})(d_{25}\ d_{23}\ d_{35})=(3\ 5\ 7)(4\ 1\ 0)=17$

lower bound=42+96+17=155

Iterative Force-Directed

- **Algorithm**

  Score current placement

  UNTIL stopping criterion is satisfied DO

  Select component(s) to move

  Move selected component(s) to trial positions

  Score trial placement

  IF trial score <= current score THEN

  current score <-- trial score

  ELSE

  Move selected component(s) to previous positions

  ENDLOOP

- **Force-directed interchange**
  - **Primary module is trial interchanged with its three neighbors (vertical, horizontal, and diagonal) in the direction of its force vector**

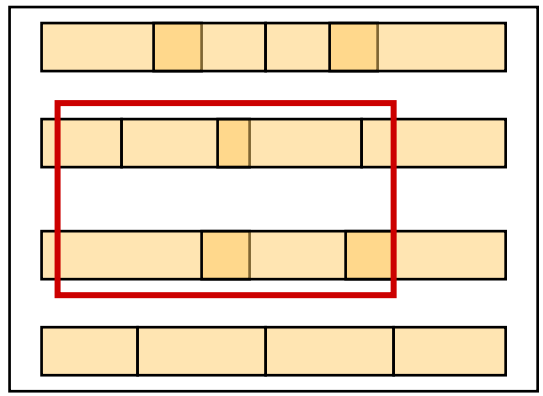
- **Force-directed relaxation**
  - **Select primary component in descending order based on the number of nets connected to the component**
  - **Move primary component to the slot nearest to zero-force point**
  - **The component that was occupying that slot becomes the next primary component**
  - **When moved to an empty slot, the series terminates**

    --> If improved, accept the series

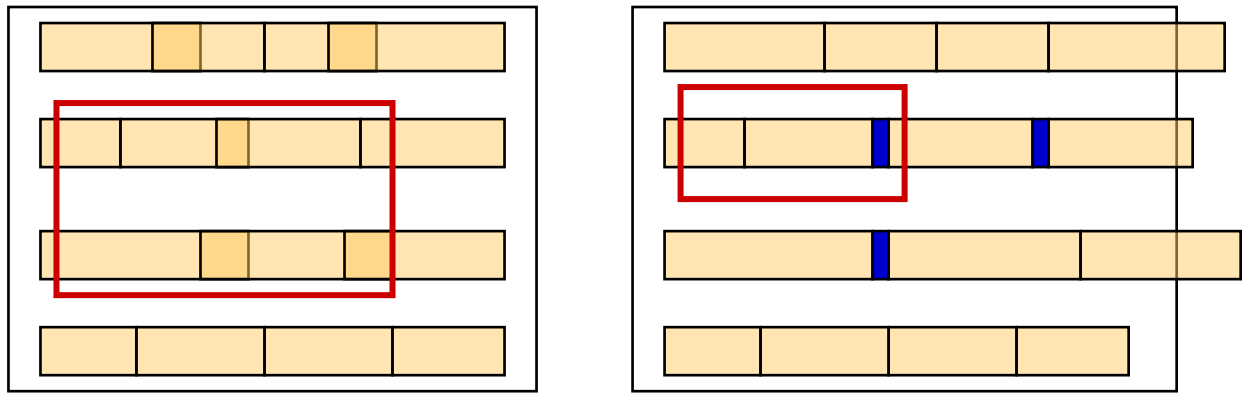
    Otherwise, reject
- **Force-directed pair-wise relaxation**
  - **Find the zero-force target location**
  - **Primary component is trial exchanged with a component in the vicinity of the target location only if the target location is in the vicinity of the primary component**
  - **The exchange is accepted only if the score improves**

Simulated Annealing

- **C.Sechen and A.Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE Journal of Solid-State Circuits*, April 1985.**
- **Standard Cell**
  - **Moves**
    - **M1: displace a module to a new location**
    - **M2: interchange two modules**
    - **M3: change orientation of a module(reflection for standard cells)**
    - **M1:M2 = 5:1**
    - **If M1 is chosen but rejected, try M3**
  - **Range limiter**
    - **Rectangular window**
    - **Shrink as temperature decreases**



- **Stage 1**
  - **Modules are moved between different rows as well as within the same row**
  - **Module overlaps are allowed**
- **Stage 2**
  - **Begins when the window size becomes so small (temperature is reduced) that modules are no longer permitted to switch rows**
  - **All overlaps are removed**
  - **Modules are not moved between different rows**
  - **Insert route-through cells as necessary**

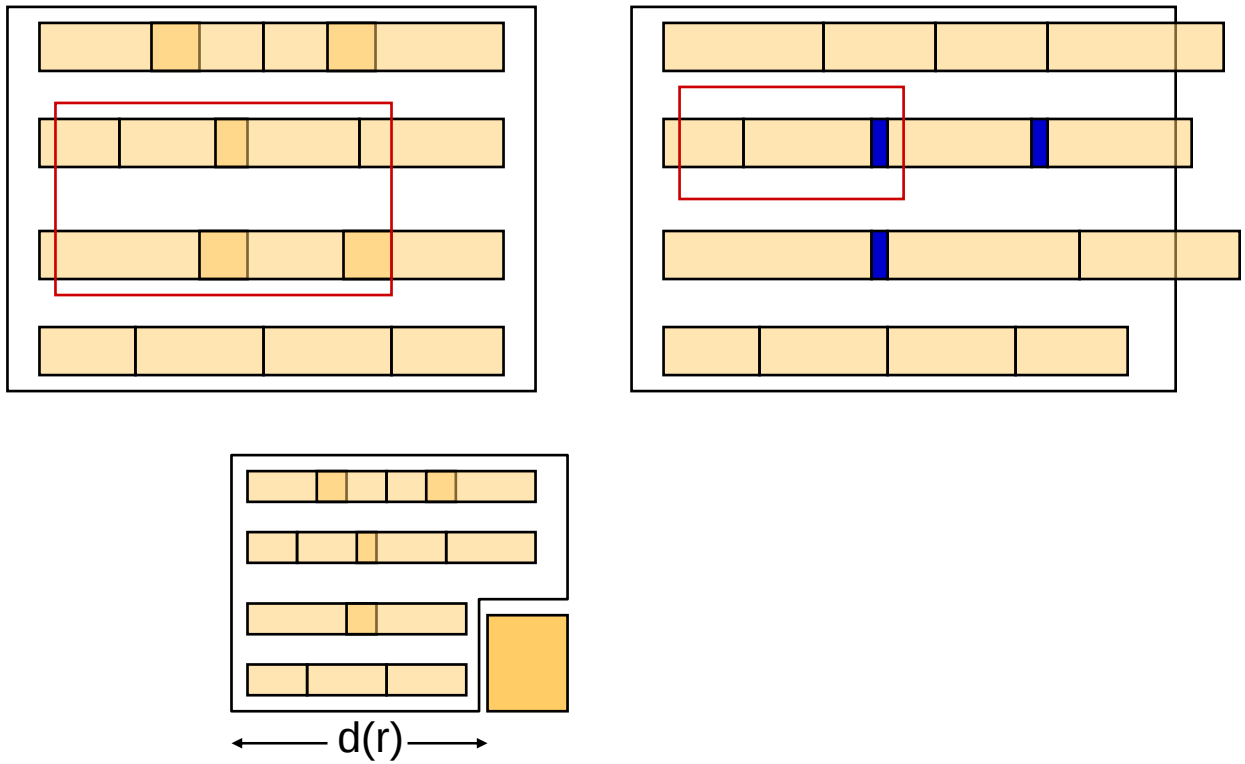


- **Cost = C1 + C2 + C3**
  - **C1 = Σ(Xs(n)Hw(n) + Ys(n)Vw(n))**
  - **C2 = $\Sigma_{i \neq j}$(O(i,j) + α)^2**

    α is to make total overlap --> 0 when T --> 0
  - **C3 = $\Sigma\beta$|l(r)-d(r)|**

    l(r): sum of lengths of cells in row r

    d(r): desired length of row r

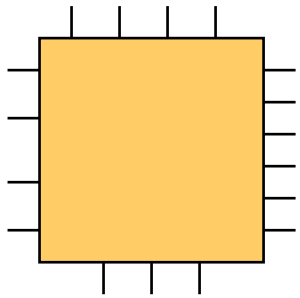


- **Macro/Custom cell**
  - **Macro cell: fixed dimension, pin location**
  - **Custom cell: estimated area, pin list**
  - **Moves**
    - **M1: displacement to a new location**
    - **M2: interchange of two modules**
    - **M3: change of orientation**
    - **M4: aspect ratio change for a flexible module**
    - **M5: assignment of an uncommitted pin(sequence of pins) to a new pin site(s)**
    - **M1:M2=9:1**

      --> If M1 is rejected, try M3

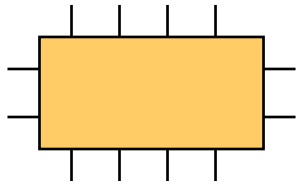
      --> If rejected, try M4 for custom cells

      --> If rejected, try M5



- **Cost function**
  - **C1 = Σ(half perimeters of the bounding rectangles)**
  - **C2 = $\Sigma_{i \neq j}(O(i,j) + \alpha)^2$**
  - **C3 = $\Sigma\beta_i$(pin site capacity overflow)^2**

    β_i:larger for smaller capacity site
- **Range limiter is used**
- **Also applicable to PCB placement**



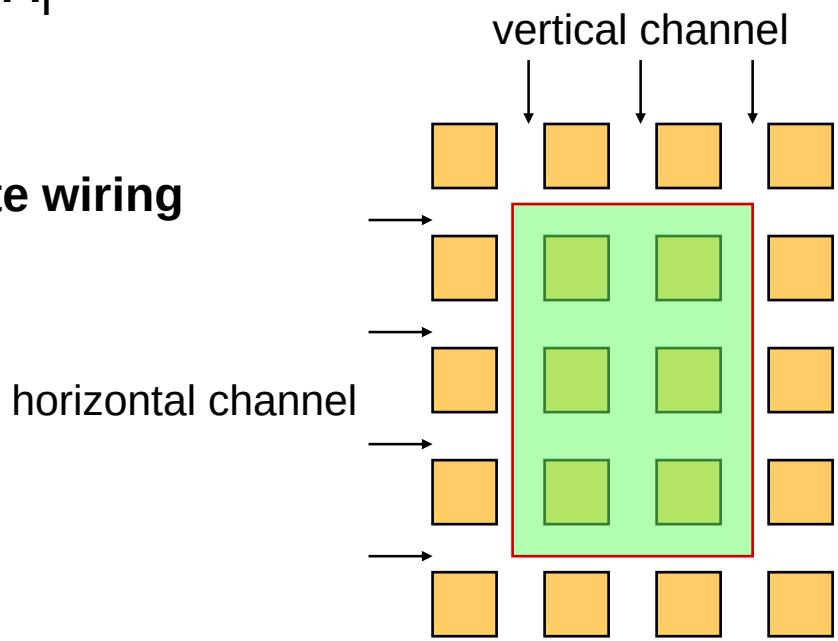
- **Gate Array Placement**
  - **Cost function 1**
    - **l_i: routing channel**
    - **w_i: total number of nets whose bounding rectangles intersect with channel l_i**
    - **Net-crossing histogram: the set of w_i's, one for each channel l_i**
    - **$W = \Sigma w_i$ is an estimation of total wiring length**
    - **t_i: wiring capacity of l_i**

      $\delta_i = w_i - t_i$, if $w_i > t_i$

      0, otherwise

      $\Delta = \Sigma \delta_i^2$ --> distribute wiring

      cost = $W + \alpha\Delta$



– **Cost function 2**

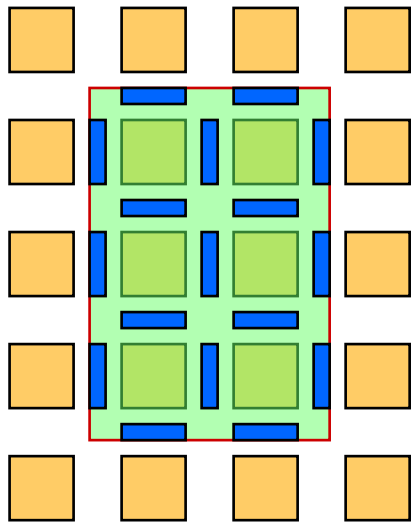
- **s_i: routing channel segment**
- **w_i: total contribution of nets whose bounding boxes intersect with channel segment s_i**
- **Contribution of a net: (half perimeter)/(# segments enclosed by the bounding box)= 5/17**
- **$W = \Sigma w_i$ is an estimation of total wiring length**
- **t_i: wiring capacity of s_i**

  $\delta_i = w_i - t_i$, if $w_i > t_i$

  0, otherwise

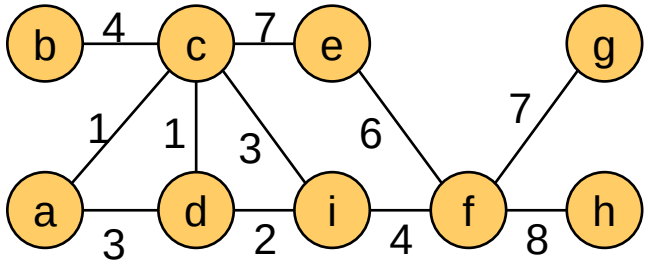
  $\Delta = \Sigma \delta_i^2$ --> distribute wiring

  cost = $W + \alpha\Delta$



Genetic Algorithm

- **Emulation of the biological process of evolution**



6	**7**	**8**
3	**4**	**5**
0	**1**	**2**

A configuration is represented by a string of symbols.
Symbol with coordinates: gene
String of symbols: chromosome
Example: [aghcbidef] (1/85) -- fitness=1/weighted wirelength=1/85

d	**e**	**f**
c	**b**	**i**
a	**g**	**h**

Example of population (4 chromosomes):
[aghcbidef] (1/85)
[bdefigcha] (1/110)
[ihagbfced] (1/95)
[bidefaghc] (1/86)

- **Given initial population,**
  - **Compute the fitness of each individual.**
  - **Two individuals (parents) are selected.**
    - **The probability of selection is proportional to the fitness.**
  - **Apply genetic operators to generate offsprings.**
    - **Crossover**
    - **Mutation**
    - **Inversion**
  - **Select a new generation by selecting some of the parents and some of the offsprings**
  - **After a number of iterations, the most fit individual is selected as the solution.**

- **Crossover**
  - **Main genetic operator**
  - **Generates an offspring by combining chromosome segments of the two parents.**
  - **Order crossover, partially mapped crossover (PMX), cycle crossover**

Naive approach:

```
   [bidef|aghc] (1/86)
   [bdefi|gcha] (1/110)
-->[bidef gcha] (1/63)
```

Infeasible solution:

```
   [bdefi|gcha] (1/110)
   [aghcb|idef] (1/85)
-->[bdefi idef]
```

Order crossover:

```
   [aghcb|idef] (1/85)
   [bdefi|gcha] (1/110)
-->[aghcb defi] (1/82)
```

PMX:

```
   [aghcb|idef] (1/85)
   [bdefi|gcha] (1/110)
-->[fiedb gcha] (1/122)
```

Cycle crossover:

```
   [aghcb|idef] (1/85)
   [bdefi|gcha] (1/110)
-->[ag cb id f]
   [agecb idhf] (1/90)
```

– **Order crossover**

- **P1 [aghcb|idef] (1/85)**
  P2 [bdefi|gcha] (1/110)

- **copy left substring of P1**
  [aghcb]

- **copy right substring of P1 in the order they appear in P2**
  [aghcb defi] (1/82)

– **Partially mapped crossover (PMX)**

- **P1 [aghcb|idef] (1/85)**
  P2 [bdefi|gcha] (1/110)

- **copy right substring of P2**
  [gcha]

- **copy genes in the left substring of P1 if not copied yet**
  [b gcha]

- **for the genes already copied, use the ones in the right substring of P1 (in the same position)**
  [fiedb gcha] (1/122)

– **Cycle crossover**

- **P1 [aghcb|idef] (1/85)**
  P2 [bdefi|gcha] (1/110)

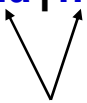
- **generate a cycle starting from P1 and copy genes in P1 and in the cycle**
  [ag cb id f]

- **repeat starting from P2**
  [agecb idhf] (1/90)

- **repeat until done**

- **Mutation**
  - **Makes random changes in the offsprings.**
  - **Pairwise interchange is commonly used.**
  - **Generates new genes.**
- **Inversion**
  - **A chromosome segment is flipped.**
  - **Does not modify the solution but modifies only the representation.**
  - **Reduce the probability of splitting up symbols that is to be placed together.**

```
    [bid|efgch|a]
--> [bid|hcgfe|a]
```



less likely to be splitted