Elective II: VLSI Design

Code: CISM 402

Pritha Banerjee

Courtsey for slides: Yao Wen Chang, National Taiwan University, Taiwan

Placement

- Books:
  - Chapter 5 of Naveed A. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers
  - Chapter 2 (2.3) of M. Sarafzadeh and C. K. Wong, *An introduction to VLSI Physical Design*, The McGraw Hill Companies, Inc.

Physical Design Flow

Circuit Partitioning

↓

Floorplanning & Placement

↓

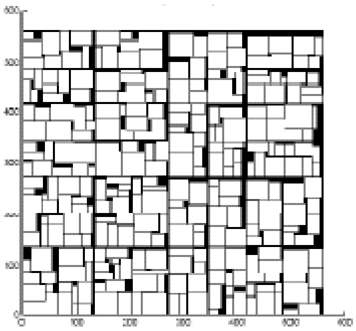
Routing

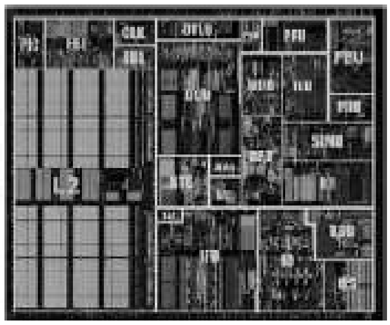
↓

Layout Compaction

↓

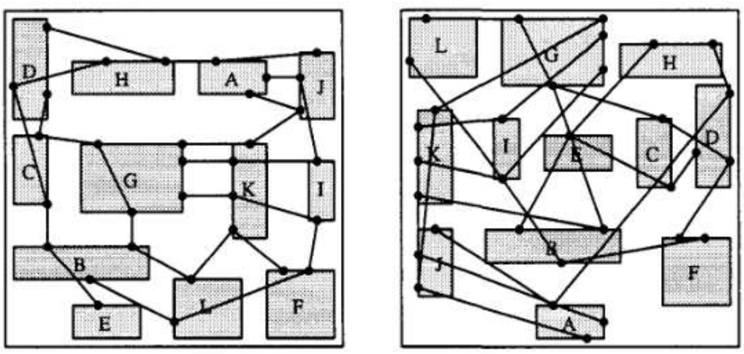
Extraction and Verification





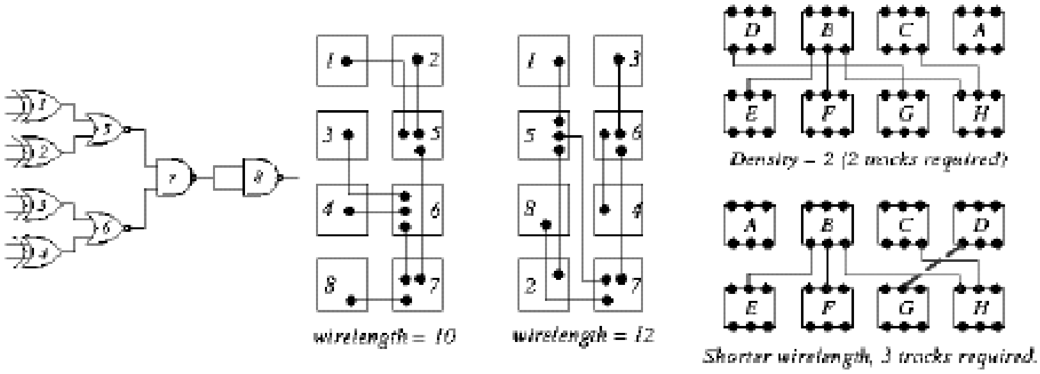
Placement

- Problem:
  - Given a netlist of fixed blocks with fixed pin locations, construct a layout of blocks (find positions) s.t
    - all nets are routable (congestion driven placement)
    - Area is minimized (area /wirelength driven placement)
    - Length of longest path (critical path length) is minimized : performance driven (timing driven placement)
- Placement problem is NP complete



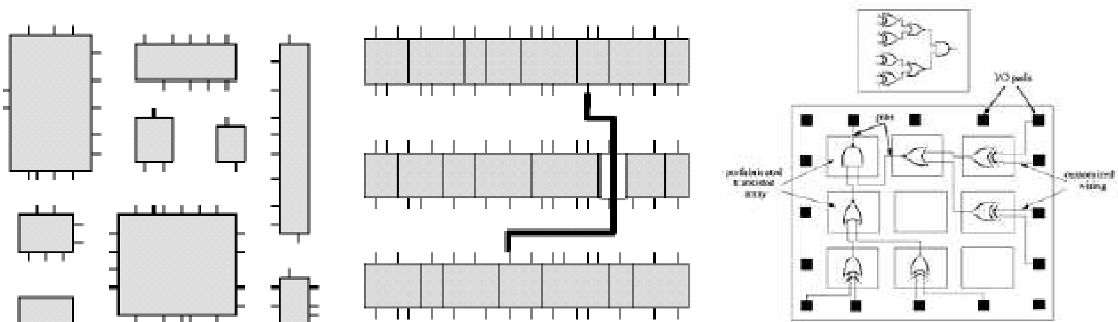
Placement

- Inputs: A set of **fixed** cells/modules, a netlist.
- Goal: Find the best position for each module on the chip according to appropriate cost functions.
  - Considerations: **routability/channel density**, **wirelength**, power, timing, thermal, I/O pads, manufacturability, etc.



Design style specific placement problems

- Full Custom: blocks have different size and shape, minimize area and critical path length
- Standard Cells: cells have same height, minimize channel height, width of widest row
- Gate arrays: assign gates to slots on the gate array



Basic wirelength Models

- **Half-perimeter wirelength (HPWL):** Half the perimeter of the bounding rectangle that encloses all the pins of the net to be connected. Most widely used approximation!
- **Squared Euclidean distance:** squares of all pairwise terminal distances in a net using a quadratic cost function

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\gamma_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2]$$

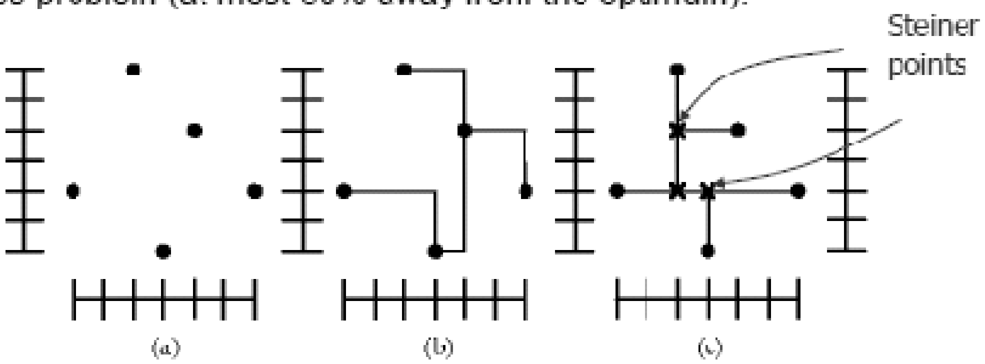
- **Steiner-tree approximation:** Computationally expensive.
- **Minimum spanning tree**: Good approximation to Steiner trees.

Minimum spanning Tree

- Given an edge weighted undirected graph G(V,E), find the minimum set of edges that spans all the vertices of G
- Kruskal's Algorithm: O(E log E)
  - Sort edges in non decreasing order, have |V| sets
  - add edge (u,v) in tree if u, v belongs to different set and create new set having u and v, else
  - edge is discarded
- Prim's Algorithm: O(E log E)
  - Start with random vertex, pick least weighted edge that do not form a cycle

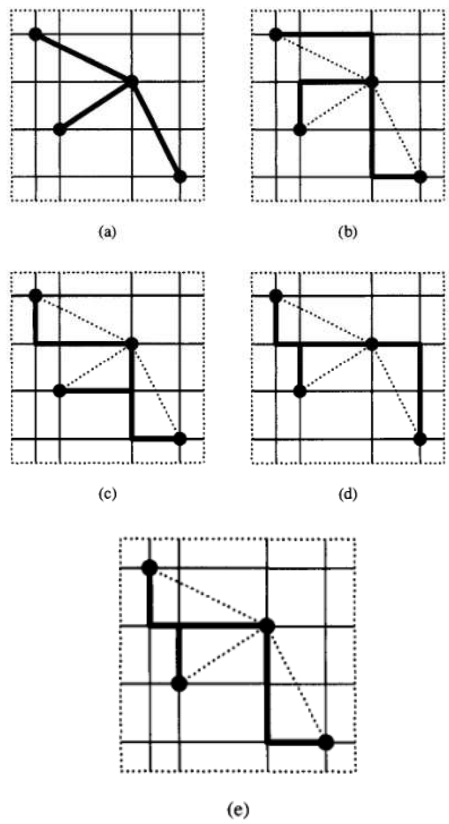
Spanning Tree Vs. Steiner Tree

- **Manhattan distance:** If two points (nodes) are located at coordinates (x_1, y_1) and (x_2, y_2), the Manhattan distance between them is given by $d_{12} = |x_1-x_2| + |y_1-y_2|$ (a.k.a. λ-1 metric)
- **Rectilinear spanning tree:** a spanning tree that connects its nodes using Manhattan paths (Fig. (b) below).
- **Steiner tree:** a tree that connects its nodes, and additional points (**Steiner points**) are permitted to be used for the connections.
- The minimum rectilinear spanning tree problem is in P, while the minimum rectilinear Steiner tree (Fig. (c)) problem is NP-complete.
  - The spanning tree algorithm can be an *approximation* for the Steiner tree problem (at most 50% away from the optimum).

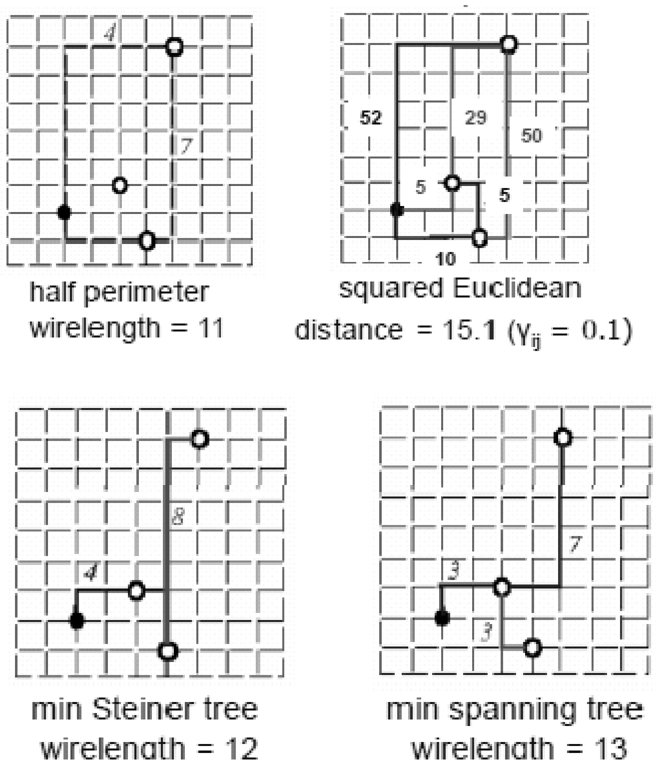


Spanning Tree Vs. Steiner Tree

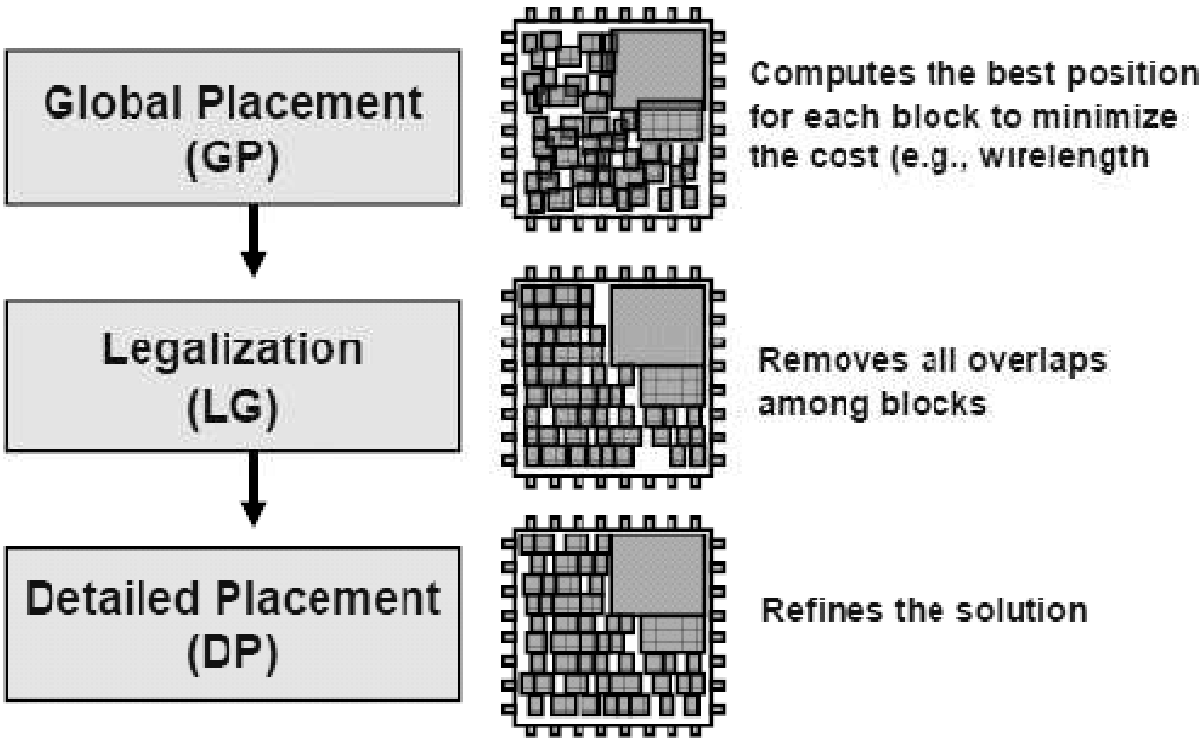
- Find Minimum Spanning tree: cost(MST)
- Apply local modification to obtain Rectilinear Steiner tree: cost (RSMT)
- Cost(MST)/Cost (RSMT) <= 3/2



Basic wirelength Models



Typical Placement Flow



Global Placement Methods

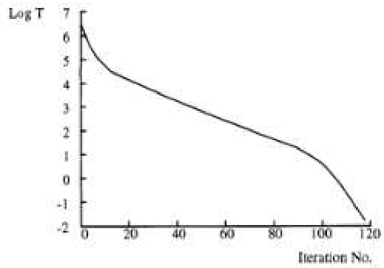
- The placement problem is NP-complete.
- Popular placement algorithms:
  - **Constructive algorithms:** once the position of a cell is fixed, it is not modified anymore.
    - Cluster growth, linear assignment, min cut, QP, etc.
  - **Iterative algorithms:** intermediate placements are modified in an attempt to improve the cost function.
    - Force-directed method, etc
  - **Nondeterministic approaches:** simulated annealing, genetic algorithm, etc.
- Can combine multiple elements:
  - Constructive algorithms are used to obtain an initial placement.
  - The initial placement is followed by iterative improvement.
  - The results can further be improved by simulated annealing.

Placement Methods

- Simulated annealing based placement for standard cells
- Partition based placement for custom design
- Analytical placement : force directed
- Regular Placement : Assignment problem
- Placement by Genetic algorithm

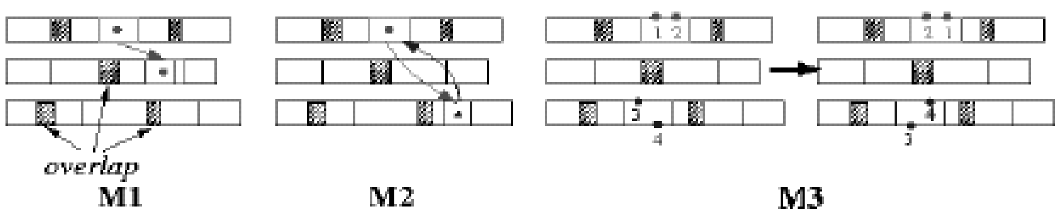
Placement : Simulated annealing based

- Timber-Wolf 3.2 for standard cell placement
  - Initial temperature : 4000000
  - Final temperature : 0.1
  - Cooling Schedule (T): α(T). T
    - α(T): low when T high: 0.8, temperature decreases rapidly
    - α(T): 0.95, when T medium, temperature changes slowly
    - α(T): 0.8 when T low, temperature changes faster
  - Inner loop criteria: # of trials per temperature
    - Depends on # of blocks in circuit



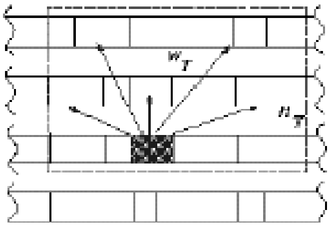
Placement : Simulated annealing based

- Timber-Wolf 3.2 for standard cell placement
  - Moves to generate new configuration
    - **Solution Space:** All possible arrangements of the modules into rows, possibly with overlaps.
    - **Neighborhood Structure:** 3 types of moves
      - M_1: Displace a module to a new location.
      - M_2: Interchange two modules.
      - M_3: Change the orientation of a module.



Placement : Simulated annealing based

- Timber-Wolf 3.2 for standard cell placement
  - Move selection
    - TimberWolf first tries to select a move between M_1 and M_2: $Prob(M_1) = 0.8$, $Prob(M_2) = 0.2$.
    - If a move of type M_1 is chosen and it is rejected, then a move of type M_3 for the same module will be chosen with probability 0.1.
    - Restrictions: (1) what row for a module can be displaced? (2) what pairs of modules can be interchanged?
    - **Key: Range Limiter**
      - At the beginning, (W_T, H_T) is big enough to contain the whole chip.
      - Window size shrinks as the temperature decreases. Height, width $\propto log(T)$.
      - Stage 2 begins when window size is so small that no inter-row module interchanges are possible.

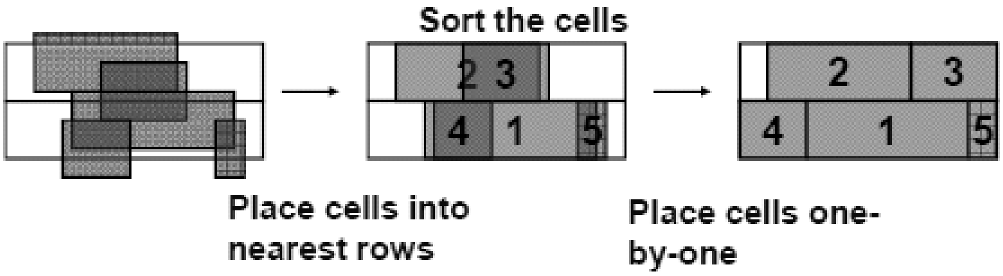


Placement : Simulated annealing based

- Timber-Wolf 3.2 for standard cell placement
  - Cost function
    - Cost function: $C = C_1 + C_2 + C_3$.
    - C_1: **total estimated wirelength.**
      - $C_1 = \Sigma_{i \in Nets}(\alpha_i w_i + \beta_i h_i)$
      - α_i, β_i are horizontal and vertical weights, respectively. ($\alpha_i=1$, $\beta_i=1$ $\Rightarrow$(1/2) $\times$ perimeter of the bounding box of Net i.)
      - Critical nets: Increase both α_i and β_i .
      - If vertical wirings are "cheaper" than horizontal wirings, use smaller vertical weights: $\beta_i < \alpha_i$.
    - C_2: **penalty function for module overlaps.**
      - $C_2 = \gamma \Sigma_{i \neq j} O^2_{ij}$, γ: penalty weight.
      - O_{ij}: amount of overlaps in the x-dimension between modules i and j.
    - C_3: **penalty function that controls the row length.**
      - $C_3 = \delta \Sigma_{r \in Rows}|L_r - D_r|$, δ : penalty weight.
      - D_r: desired row length.
      - L_r: sum of the widths of the modules in row r.

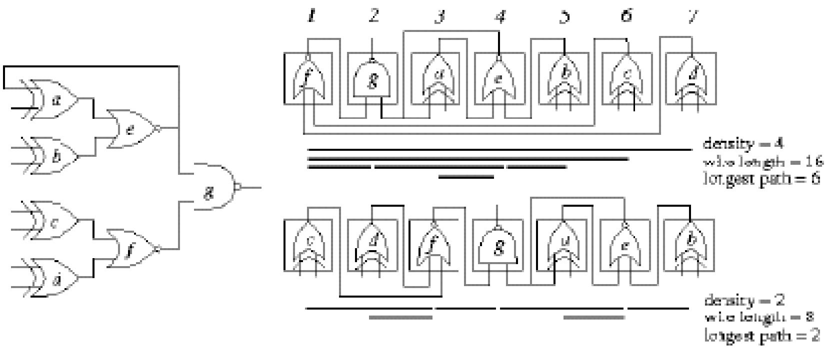
Placement : Simulated annealing based

- Remove Overlap (Legalization)
  - Place all cells in the rows to obtain a feasible solution
    1) Place cells into their nearest rows
    2) Sort all standard cells according to their sizes, from the largest to the smallest
    3) Assign the x-coordinates for all cells according to the sorted order. If overlap occurs, we will find a nearest empty slot to place the cell



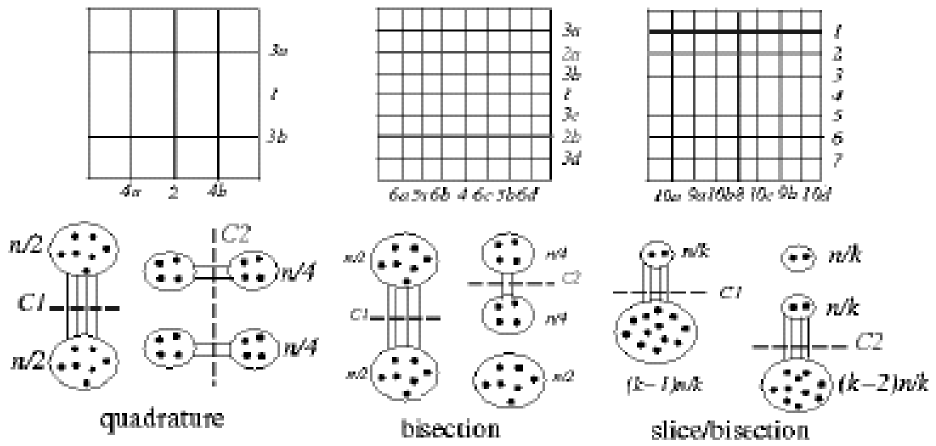
Placement : Partition based Placement

- top-down min-cut based
  – Reduce number of nets cut across partition on chip
  – Objectives
    - Total net cut: Sum of all nets cut by vertical and horizontal cut line; minimize semi perimeter wirelength
    - Min-max cut value: for std. cell, gate arrays, need to reduce # of nets routed through channel; cut-line is the channel between rows



Placement : Partition based

- top-down min-cut based
  - Breuer, "A class of min-cut placement algorithms," DAC-77.
  - **Quadrature:** suitable for circuits with high density in the center.
  - **Bisection:** good for standard-cell placement.
  - **Slice/Bisection:** good for cells with high interconnection on the periphery.



Placement : Partition based

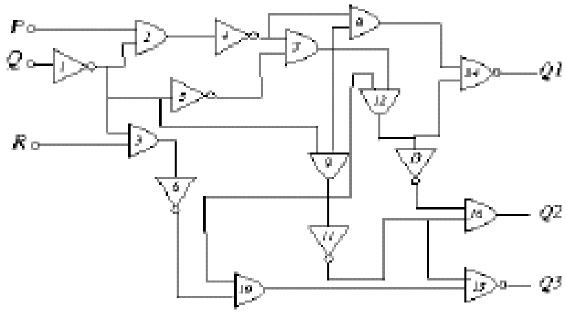
- top-down min-cut based

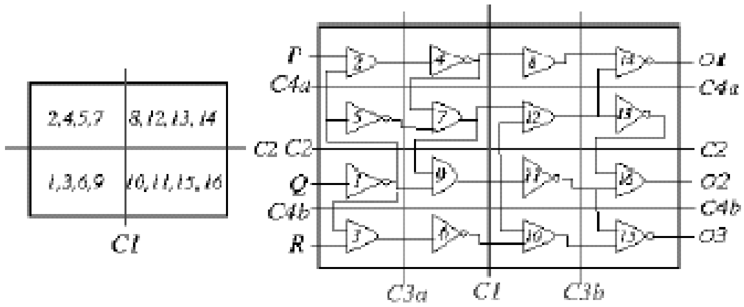
Algorithm: Min_Cut_Placement(N, n, C)
/* N: the layout surface */
/* n : # of cells to be placed */
/* n_0: # of cells in a slot */
/* C: the connectivity matrix */

begin
if ($n \le n_0$) **then** PlaceCells(N, n, C)
else
    $(N_1, N_2) \leftarrow$ CutSurface(N);
    $(n_1, C_1), (n_2, C_2) \leftarrow$ Partition(n, C);
  Call Min_Cut_Placement(N_1, n_1, C_1);
  Call Min_Cut_Placement(N_2, n_2, C_2);
end

Placement : Partition based

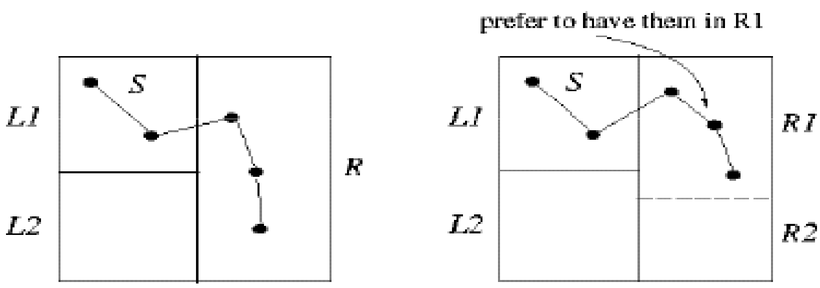
- Quadrature placement example
  - Apply the F-M or K-L heuristic to partition + Quadrature Placement: Cost $C_1 = 4$, $C_{2L} = C_{2R} = 2$, etc.



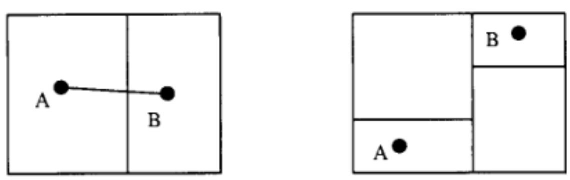


Min cut placement with terminal propagation

- Drawback of the original min-cut placement: Does not consider the positions of terminal pins that enter a region.
  - What happens if we swap {1, 3, 6, 9} and {2, 4, 5, 7} in the previous example?

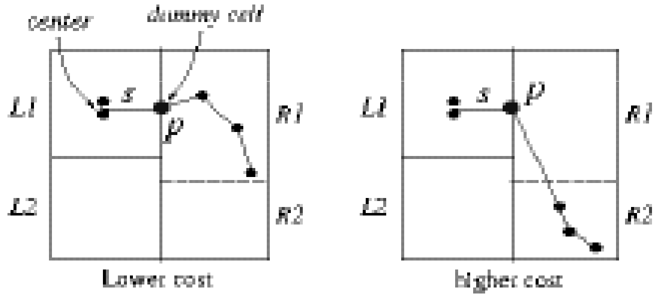


- Increases wirelength
- Increases congestion



Min cut placement with terminal propagation

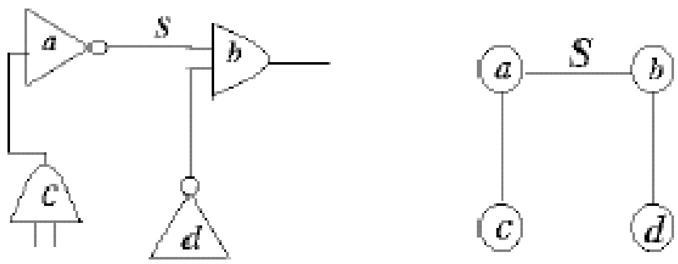
- Introduce dummy terminal at the nearest boundary, fore partitioner not to put the blocks in the other partition during next level of partition
  - We should use the fact that s is in L_1!

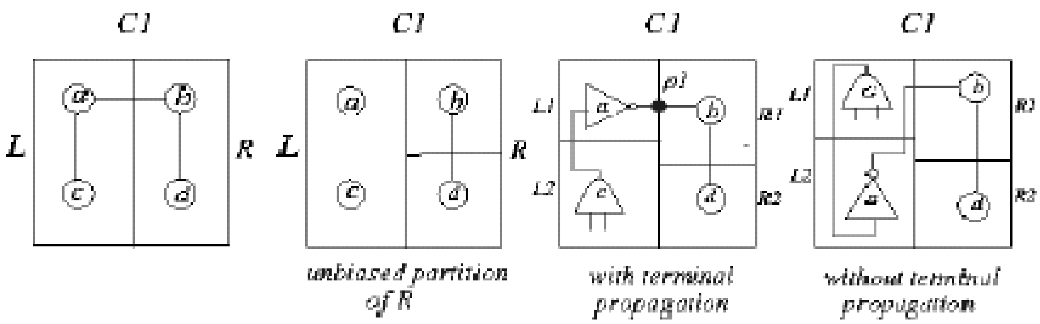


P will stay in R1 for the rest of partitioning!

Min cut placement with terminal propagation

- Partitioning must be done breadth-first, not depth-first.





Regular Placement: Assignment problem

- Possible positions (targets) of modules are known
- Assign each module to a target s.t wirelength is minimized
  - Relaxed placement (may have overlaps)
  - Overlap removal
- Relaxed placement : solve an LPP
- Overlap removal: solve minimum weighted perfect matching

Regular Placement: Relaxed Placement

- LPP formulation

For a net N_i, define three variables,

$X_{l,i}$: the leftmost position of net N_i,
$X_{r,i}$: the rightmost position of net N_i, and
X_v : possible location of module M_v, where module M_v is connected to net N_i.

$$X_l \le X_{l,i} \le X_v \le X_{r,i} \le X_r, \tag{2.32}$$

where X_l and X_r are the values of the left and right boundaries of the chip, respectively. For fixed modules, $X_{v1}, X_{v2}, ..., X_{vk}$,

$$X_{vi} = X_i. \tag{2.33}$$

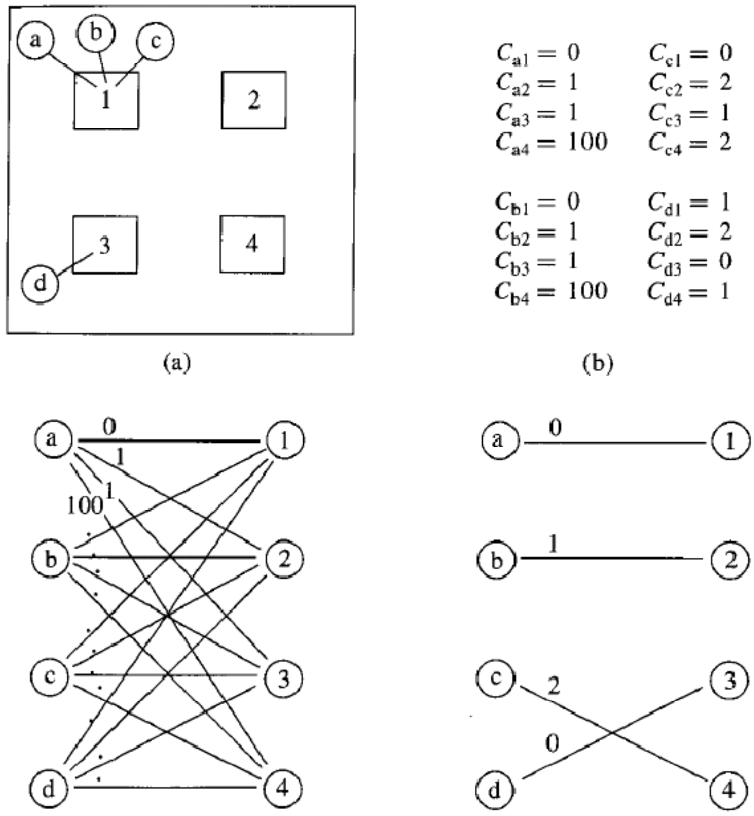
minimize

$$\sum_{N_i \in \mathcal{N}} W(i)(X_{r,i} - X_{l,i}).$$

- Similar equation for Yi

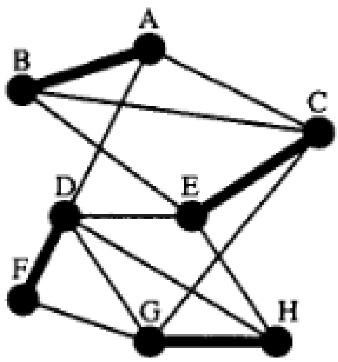
Regular Placement: Relaxed Placement

- Overlap removal
- Cij = Cost(Mi,Hj)
- Cij = distance of current position of Mi to Hj
- Define complete bipartite graph
  - LHS: modules
  - RHS: targets
  - Edge weight = cij
- Find a minimum weighted perfect matching:O(n^3)



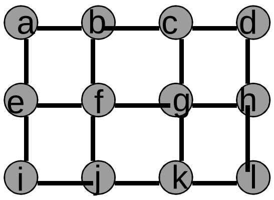
Matching

- G=(V,E) be a connected undirected graph
- Matching in G is a subset of edges M s.t no two edges in M have a vertex in common
- e is matched if it is in M else unmatched or free
- v is matched if it is incident to matched edge else unmatched or free
- A maximum matching: matching of maximum cardinality. |M| is maximum
- A perfect matching: every vertex is matched
- Min wt matching: sum of weights of M is minimum

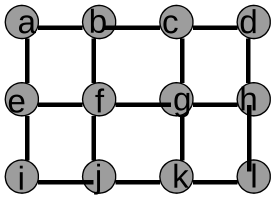


Matching

- Alternating path p w.r.t given M in undirected graph: a simple path (without repeating vertices) consisting of alternating matched and unmatched edges
- Augmenting path w.r.t M: if all the matched edges in p are in M & its endpoints are free (odd number of edges)



|M| = 4



|M| = 4 + 1

•Y: unmatched, W: mathched

•p : abcgfe: aug. path

•Reverse edges (matched becomes Unmatched and vice–versa)

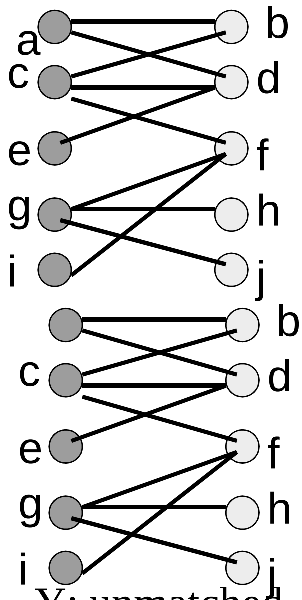
•M is maximum iff G has no aug. path w.r.t M

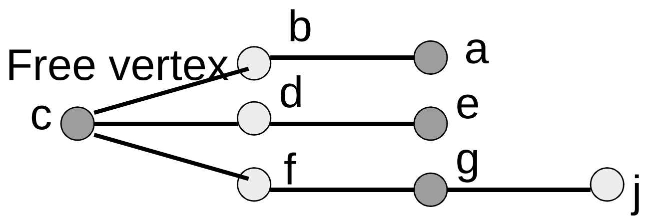
Bipartite Matching Algorithm

- Input : Bipartite graph G=(X U Y, E)
- Output: maximum matching M in G
- Initialize M to any matching (empty)
- While there exists a free x-vertex and free y vertex
  - Let r be a free x-vertex. Use BFS, grow alternating path tree T rooted at r
  - If T is Hungarian tree then G = G – T {remove T}
  - Else find aug. path p in T and M = M $\oplus$ p
- End while

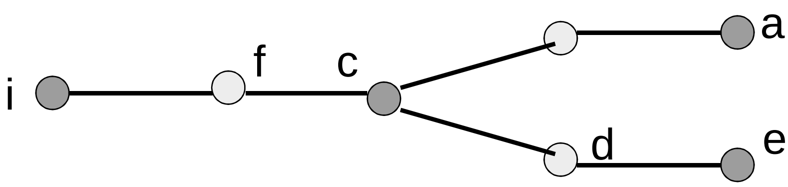
- Hungarian Tree: if no free y vertex starting from r
- M $\oplus$ p : augment matching by reversing edge roles

Bipartite Matching algorithm





Alternating path tree



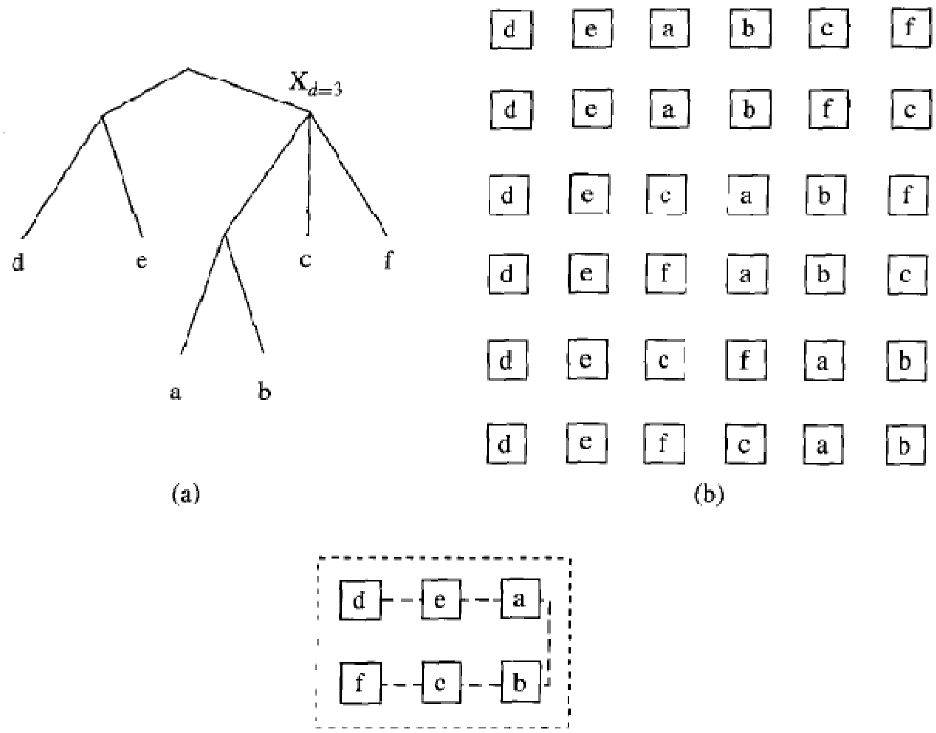
Hungarian tree

- Y: unmatched, W: mathched
- Construction of alternating path tree: O(|E|) using breadth first search
- O(|V|) trees are constructed
- Total running time : $O(|V||E|) = O(n^3)$, n= |V|

Linear Placement

- One dimensional placement problem
  - Min-cut or total length based
  - Optimal linear arrangement is NP-hard
- Use it to obtain initial placement
- Clustering / partition tree created
- Nodes are placed according to leaf sequence
- May be improved by considering all possible d! positions if a node has degree d.
- d : max degree of a node, d! time at each internal node, for each permutation O(n) is cost evaluation time; so, O(n.d!) at each internal node
- Atmost n inner nodes; So total time complexity: $O(n^2\ d!)$
- If d = n, $O(n^2\ n!)$, good quality but expensive w.r.t time
- If d = 2, $O(n^2\ )$, sacrifice quality, good run time

Linear Placement



Analytical Placement

- Force Directed placement

- Key: Solve a relaxed placement problem "optimally"
  - Ignore overlaps (fixed at later stages)
  - Adopt linear or nonlinear wirelength estimation
  - Need pads to pull the cells outwards
- Approaches for overlap removal
  - Cell spreading
  - Legalization
- Pro: Obtain very good quality for existing benchmarks
- Con: hard to handle cell rotation, big macros, region constraints, etc.

Analytical Placement

- Force Directed placement

- Key: Solve a relaxed placement problem "optimally"
  - Ignore overlaps (fixed at later stages)
  - Adopt linear or nonlinear wirelength estimation
  - Need pads to pull the cells outwards
- Approaches for overlap removal
  - Cell spreading
  - Legalization
- Pro: Obtain very good quality for existing benchmarks
- Con: hard to handle cell rotation, big macros, region constraints, etc.

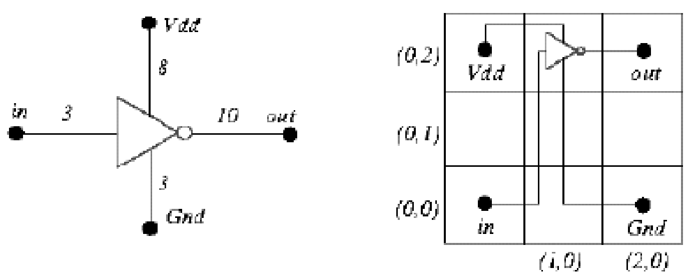
Analytical Placement

- Find zero-force target location
  - Cell i connects to several cells j's at distances d_{ij}'s by wires of weights w_{ij}'s. Total force: $F_i = \sum_j w_{ij} d_{ij}$
  - The zero-force target location ($\hat{x}_i$, $\hat{y}_i$) can be determined by equating the x- and y-components of the forces to zero:

$$\sum_j w_{ij} \cdot (x_j - \hat{x}_i) = 0 \quad \Rightarrow \quad \hat{x}_i = \frac{\sum_j w_{ij} x_j}{\sum_j w_{ij}}$$

$$\sum_j w_{ij} \cdot (y_j - \hat{y}_i) = 0 \quad \Rightarrow \quad \hat{y}_i = \frac{\sum_j w_{ij} y_j}{\sum_j w_{ij}}$$

  - In the example, $\hat{x}_i = \frac{8\times0+10\times2+3\times0+3\times2}{8+10+3+3} = 1.083$ and $\hat{y}_i$ = 1.50.



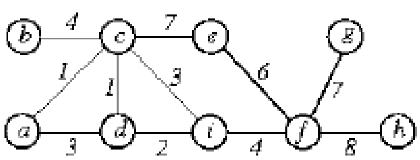
Analytical Placement

- Force Directed placement : approaches

- Approach I (constructive):
  - Start with an initial placement.
  - Compute the zero-force locations for all cells.
  - Apply **linear assignment (matching)** to determine the "ideal" locations for the cells.
- Approach II (can be constructive or iterative):
  - Start with an initial placement.
  - Select a "most profitable" cell *p* (e.g., maximum *F*, critical cells) and place it in its zero-force location.
  - "Fix" placement if the zero-force location has been occupied by another cell *q*.
- Popular options to fix:
  - **Ripple move:** place *p* in the occupied location, compute a new zero-force location for *q*, …
  - **Chain move:** place *p* in the occupied location, move *q* to an adjacent location, …
  - Move *p* to a free location close to *q*.

Placement by Genetic Algorithm

- **Genetic algorithm:** A search technique that emulates the biological evolution process to find the optimum.
- Generic approaches:
  - Start with an initial set of random configurations (**population**); each individual is a string of symbol (symbol string ↔ **chromosome**: a solution to the optimization problem, symbol ↔ **gene**).
  - During each iteration (**generation**), the individuals are evaluated using a **fitness** measurement.
  - Two fitter individuals (**parents**) at a time are selected to generate new solutions (**offsprings**).
  - Genetic operators: **crossover, mutation, inversion**
- In the example, string = [*aghcbidef*]; fitness value = $1/\sum_{(i,j)\in E} w_{ij}\, d_{ij}$ = 1/85.



6	7	8
3	4	5
0	1	2

d	e	f
c	b	i
a	g	h

Placement by Genetic Algorithm

Genetic Operator : Crossover

- Main genetic operator: Operate on two individuals and generates an offspring.
  - $[bidef|aghc](\frac{1}{86}) + [bdefi|gcha](\frac{1}{110}) \rightarrow [bidefgcha](\frac{1}{63})$.
  - Need to avoid repeated symbols in the solution string!
- **Partially mapped crossover** for avoiding repeated symbols:
  - $[bidef|gcha](\frac{1}{86}) + [aghcb|idef](\frac{1}{85}) \rightarrow [bgcha|idef]$.
  - Copy *idef* to the offspring; scan [*bidef*|*gcha*] from the left, and then copy all unrepeated genes.

Placement by Genetic Algorithm

Genetic Operator : Mutation and Inversion

- **Mutation:** prevents loss of diversity by introducing new solutions.
  - Incremental random changes in the offspring generated by the crossover.
  - A commonly used mutation: pairwise interchange.
- **Inversion:** [*bid*|*efgch*|*a*] $\rightarrow$ [*bid*|*hcgfe*|*a*].
- Apply mutation and inversion with probability P_μ and P_i respectively.

Placement by Genetic Algorithm

```
Algorithm: Genetic_Placement(Np, Ng, No, Pi, Pμ)
/* Np: population size; */
/* Ng: # of generation; */
/* No: # of offsprings; */
/* Pi : inversion probability; */
/* Pμ : mutation probability; */

begin
ConstructPopulation(Np); /* randomly generate the initial population */
for j ← 1 to Np
   Evaluate Fitness(population(j));
for i ← 1 to Ng
   for j ← 1 to No
      (x, y) ← ChooseParents; /* choose parents with probability ∝ fitness value */
      offspring(j) ← GenerateOffspring(x, y); /* perform crossover to generate offspring */
     for h ← 1 to Np
        With probability Pμ, apply Mutation(population(h));
     for h ← 1 to Np
        With probability Pi, apply Inversion(population(h));
     Evaluate Fitness(offspring(j));
   population ← Select(population, offspring, Np);
return the highest scoring configuration in population;
end
```

Placement by Genetic Algorithm

Experimental parameters

- Termination condition: no improvement in the best solution for 10,000 generations.
- Population size: 50. (Each generation: 50 unchanged throughout the process.)
- Each generation creates 12 offsprings.
- Comparisons with simulated annealing:
  - Similar quality of solutions and running time.