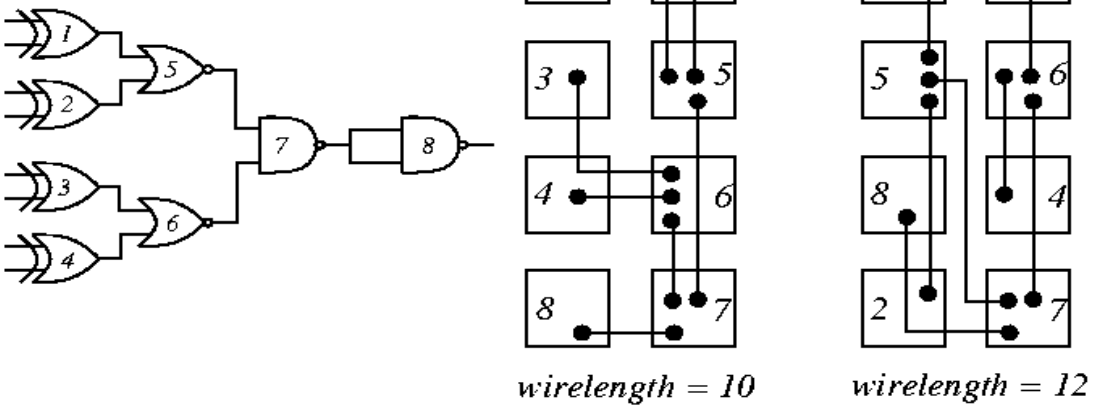
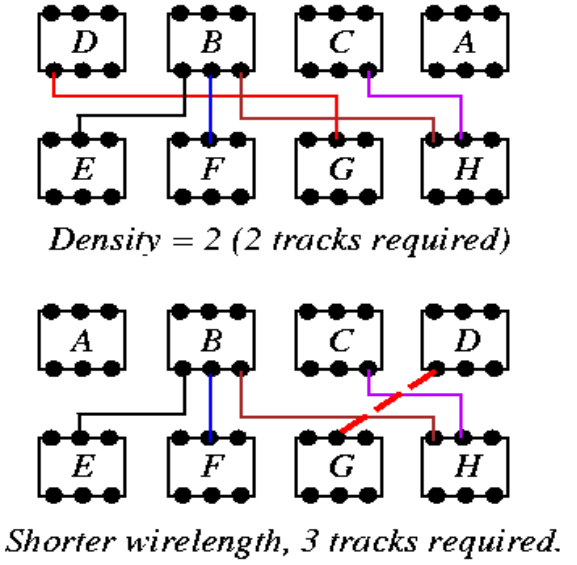
Placement (Part I)

Placement

- The process of arranging the circuit components on a layout surface.
- Inputs: A set of pre-placed/movable cells/modules and a netlist.
- Goal: Find the best position for each movable cell/module on the chip (or placement region) according to appropriate cost functions.
  - wirelength, routability/channel density, cut size, timing, power, thermal, I/O pads, manufacturability, etc.





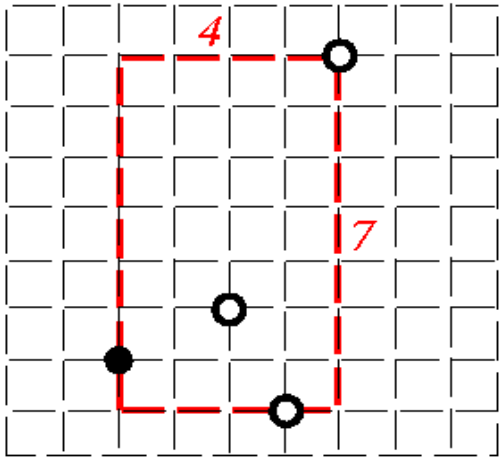
Placement for Different Design Styles

- Standard cell placement
- Gate array / FPGA placement
- Macro block placement
  - Typically called floorplanning
- Mixed-size placement
  - A few large macro blocks
  - A large number of small cells

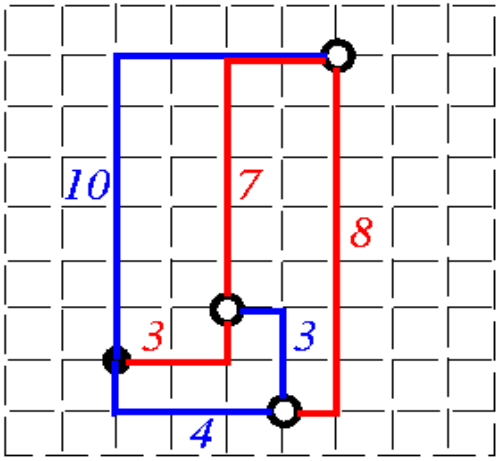
Estimation of Wirelength

- **Half-perimeter wirelength (HPWL)**: Half the perimeter of the smallest bounding rectangle that encloses all the pins of the net to be connected. Most widely used approximation!
- **Complete graph**: Since #edges ($\frac{n(n-1)}{2}$) in a complete graph is $\frac{n}{2}$x # of tree edges (n - 1), *wirelength* $\approx \frac{2}{n} \sum_{(i,j) \in net} dist(i,j)$.
- **Minimum chain**: Start from one vertex and connect to the closest one, and then to the next closest, etc.
- **Source-to-sink connection**: Connect one pin to all other pins of the net.
- **Steiner-tree (approximation)**
- **Minimum spanning tree**

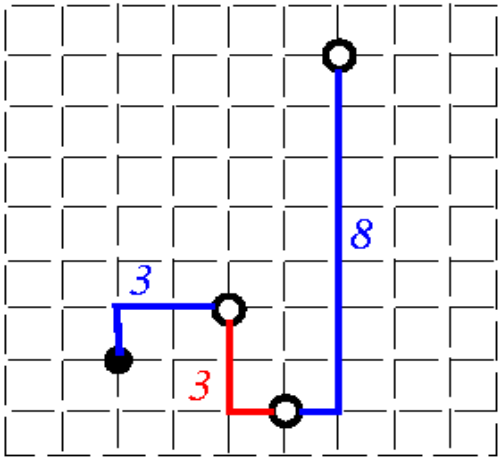
Estimation of Wirelength (cont'd)



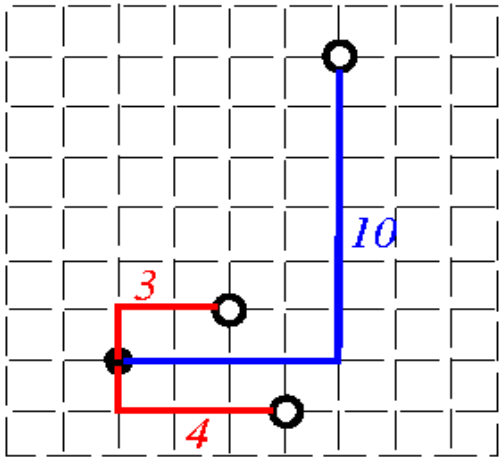
semi−perimeter len = 11



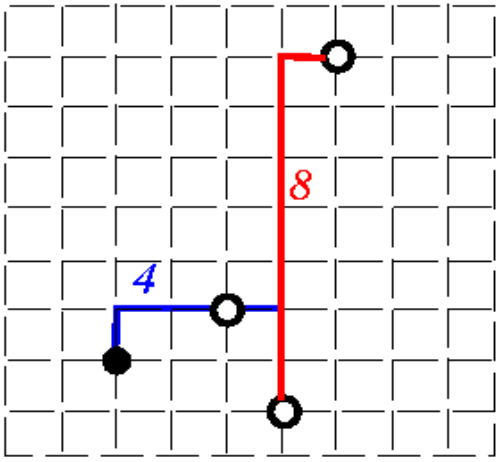
*complete graph len * 2/n = 17.5*



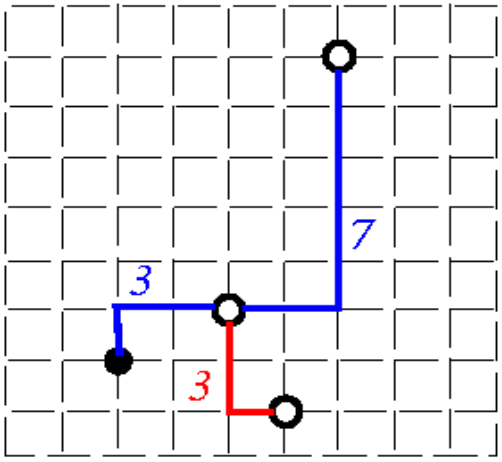
chain len = 14



source−to−sink len = 17

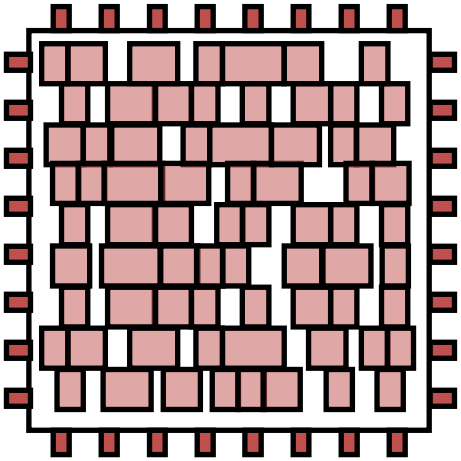


Steiner tree len = 12



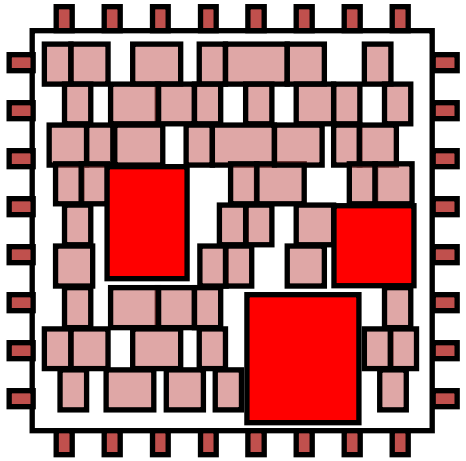
Spanning tree len = 13

Wirelength-Driven Standard Cell Placement



- Given:
  - A netlist
    - Standard cells C_1, ... , C_n with given shapes and pin locations
    - Nets N_1, ... , N_m
  - A placement region with given cell rows
- Place cells in the placement region:
  - Determine coordinates (x_i , y_i) for cell C_i
  - No overlap among cells
  - The total wirelength is minimized

Wirelength-Driven Mixed-Size Placement



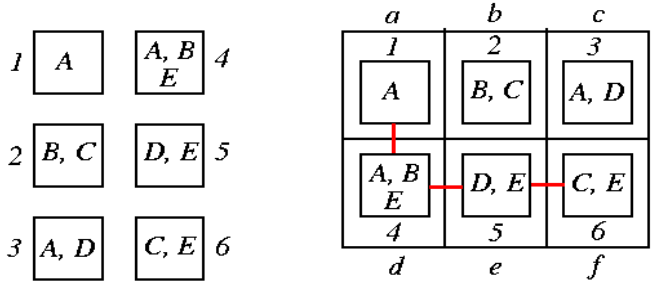
- Given:
  - A netlist
    - Standard cells C_1, ... , C_n and macros M_1, ... , M_r with given shapes and pin locations
    - Nets N_1, ... , N_m
  - A placement region with given cell rows
- Place cells/modules in the placement region:
  - Determine coordinates (x_i , y_i) for C_i / M_i
  - No overlap among cells and macros.
  - The total wire length is minimized.

Placement Is Very Hard

- Even this simplified placement problem is NP-complete:
  - 1-D placement
  - 2-pin nets only
  - All modules are of the same size
  - Wirelength as the only cost function
- Realistic placement problems:
  - 2-D placement
  - Multi-pin nets
  - Modules of different sizes
  - Other cost functions in addition to total WL
  - Different constraints for different design styles
  - Huge problem size

Placement by Linear Assignment

- Akers, "On the use of linear assignment algorithm in module placement," DAC'81.
- Total # of signal adjacencies for the placement = 3 (Signal *A*: between modules 1 and 4; Signal *E*: modules 4 and 5; Signal E: modules 5 and 6)



J:

	1	2	3	4	5	6
1	0	0	1	1	0	0
2	0	0	0	1	0	1
3	1	0	0	1	1	0
4	1	1	1	0	1	1
5	0	0	1	1	0	1
6	0	1	0	1	1	0

A:

	a	b	c	d	e	f
1	1	1	0	0	1	1
2	1	0	1	0	2	0
3	1	2	0	2	1	1
4	1	3	2	2	2	2
5	1	1	1	0	2	1
6	2	1	1	1	2	1

- J_{ij}: # of common signals between modules *i* and *j*.
- A_{ix}: # of signal adjacencies between module *i* and its neighbors if we move it to location *x*.
- Example: $A_{6a} = J_{62} + J_{64} = 2$; $A_{4e} = J_{42} + J_{44} + J_{46} = 2$.

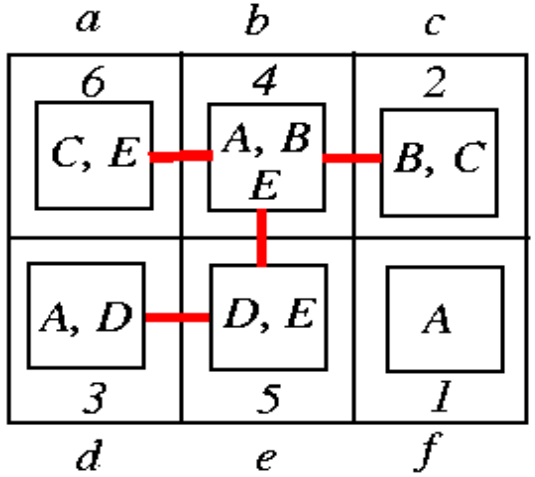
Linear Assignment (cont'd)

A:

	a	b	c	d	e	f
1	1	1	0	0	1	**1**
2	1	0	**1**	0	2	0
3	1	2	0	**2**	1	1
4	1	**3**	2	2	2	2
5	1	1	1	0	**2**	1
6	**2**	1	1	1	2	1

max linear assignment
= 1+1+2+3+2+2 = 11

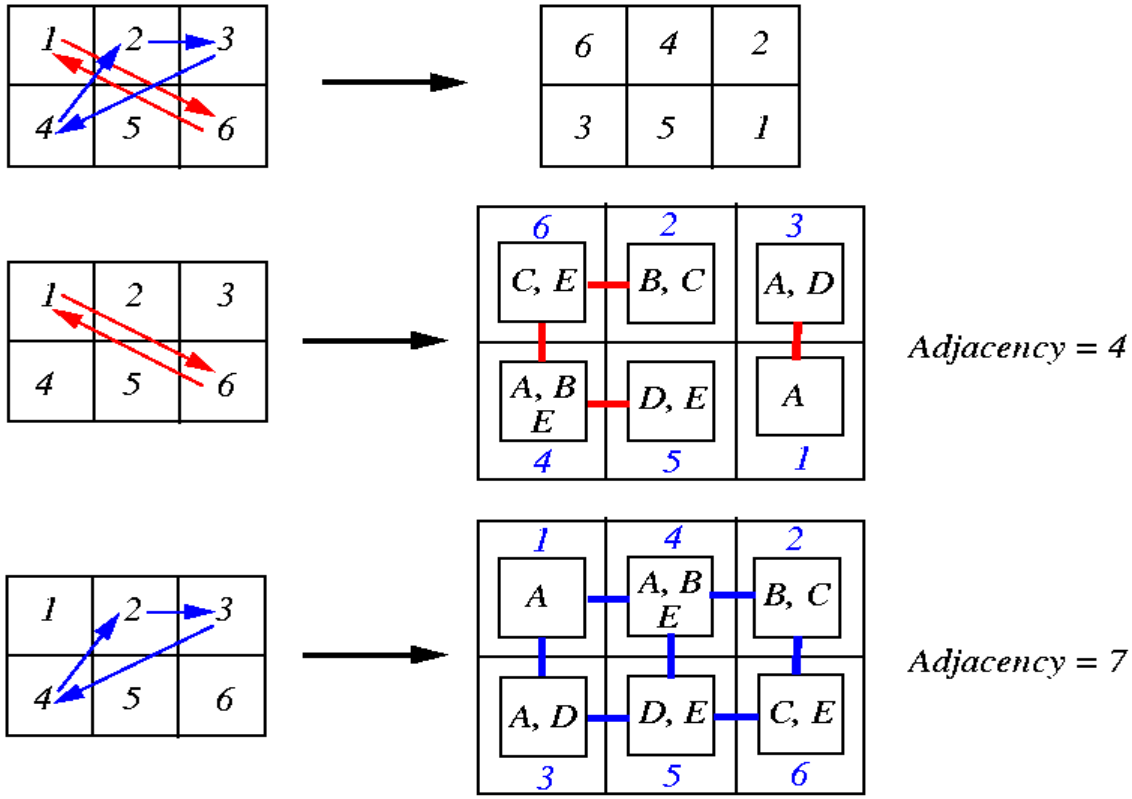
Bad approach: Rearrange the placement according to this solution



Adjacency = 4

- Minor improvement.
  - Reason: When a module reached a "good" position, the module previously adjacent to that position had gone elsewhere!

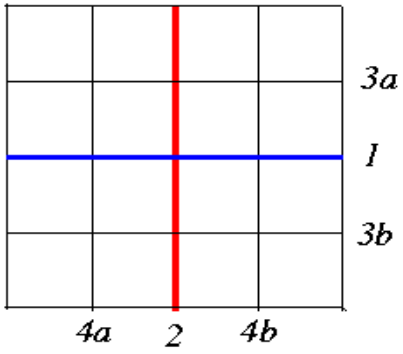
Linear Assignment: Better Approach

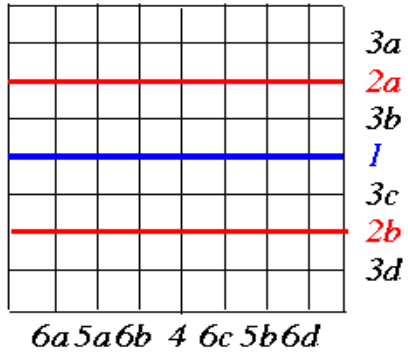


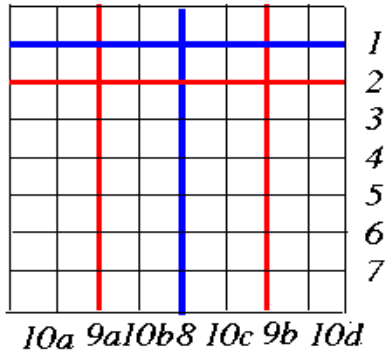
- Modify the placement based on a cycle that yields maximum improvement.
- Repeat the process all over again.

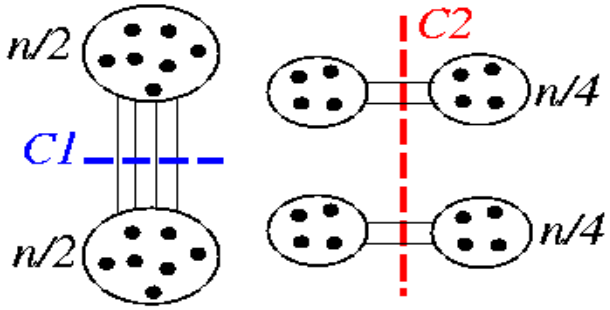
Min-Cut Placement

- 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.

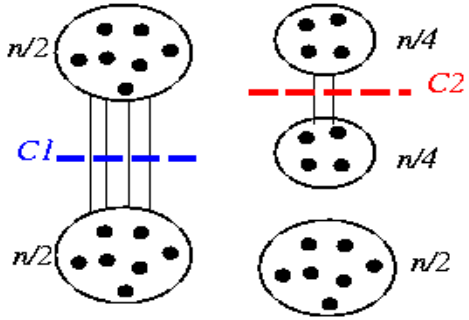




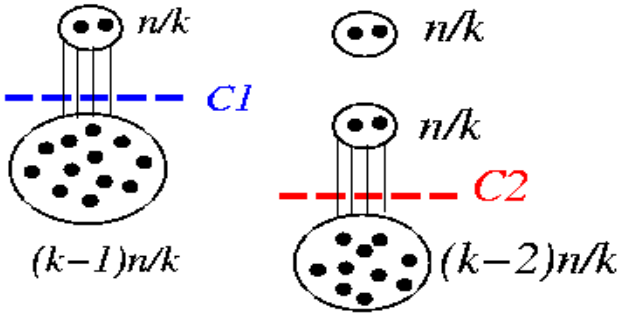




quadrature



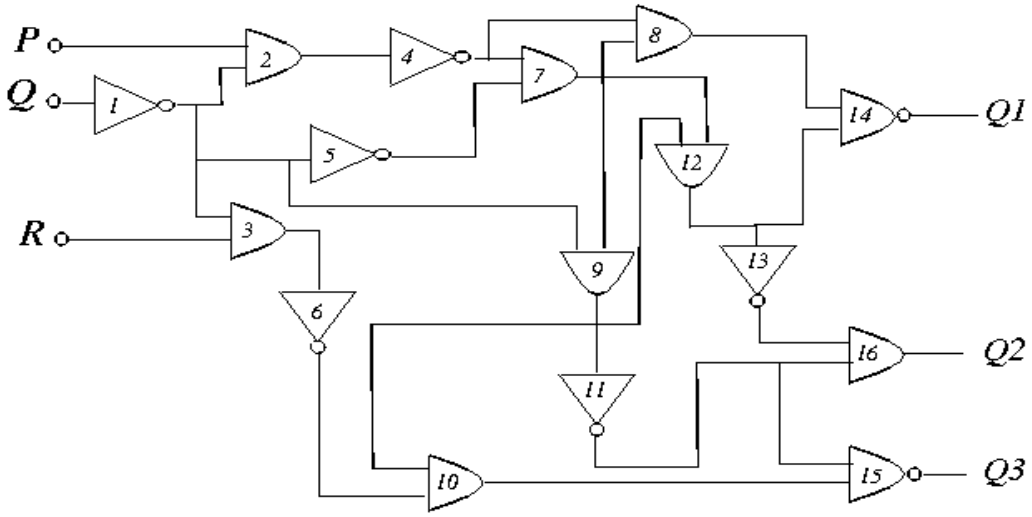
bisection

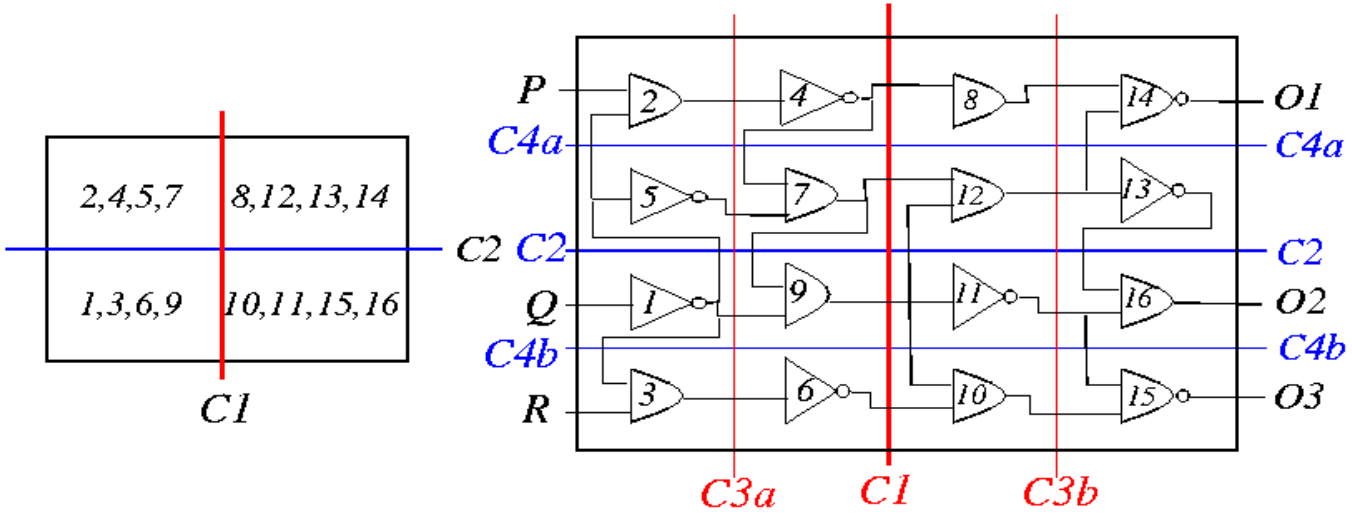


slice/bisection

Quadrature Placement Example

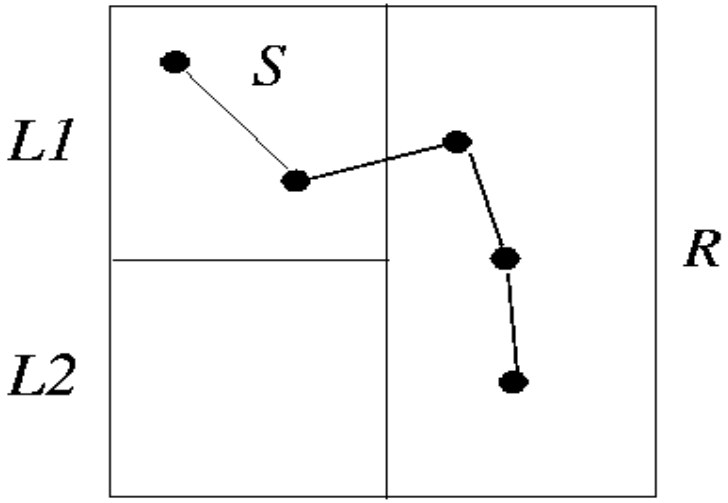
- Apply K-L heuristic to partition + Quadrature Placement: Cost $C_1 = 4$, $C_{2L} = C_{2R} = 2$, etc.

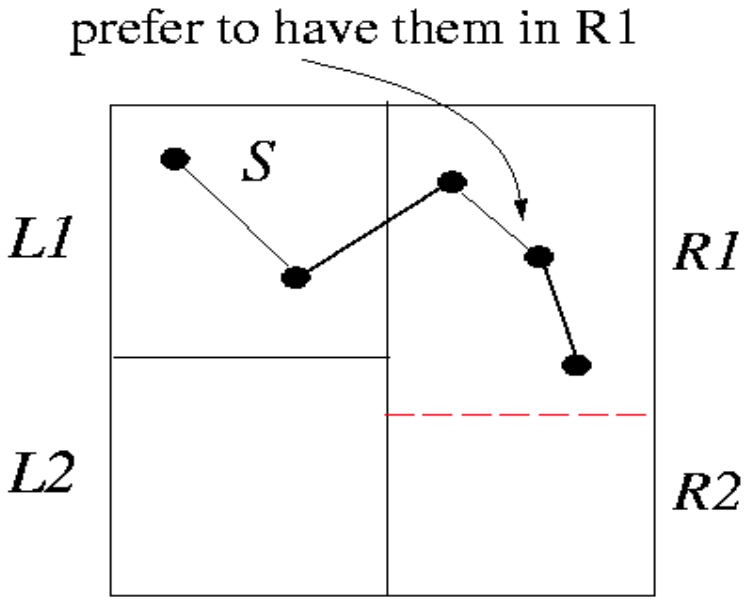




Min-Cut Placement with Terminal Propagation

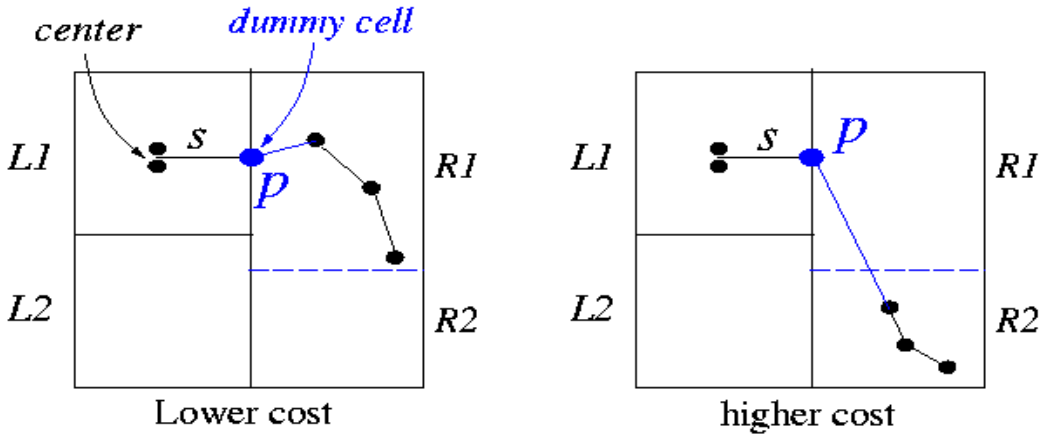
- Dunlop & Kernighan, "A procedure for placement of standard-cell VLSI circuits," *IEEE TCAD*, Jan. 1985.
- Drawback of the original min-cut placement: Does not consider the positions of pins that enter a region.
  - What happens if we swap {1, 3, 6, 9} and {2, 4, 5, 7} in the previous example?





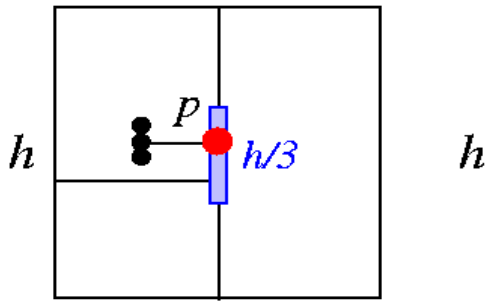
Terminal Propagation

- We should use the fact that *s* is in L_1!

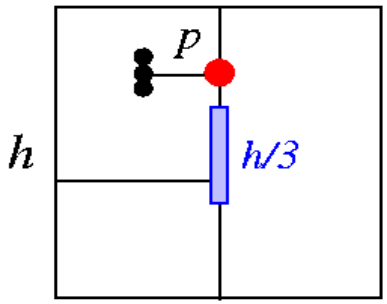


P will stay in R1 for the rest of partitioning!

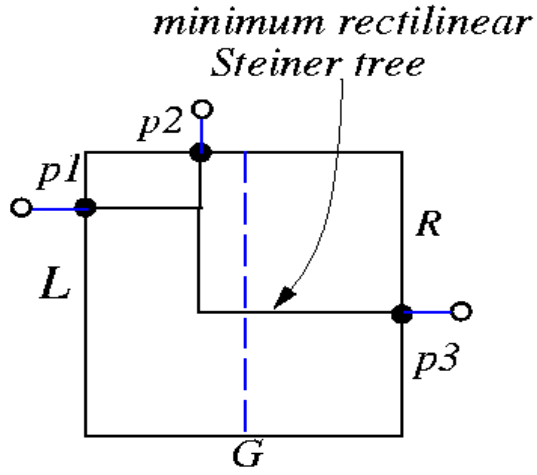
- When not to use *p* to bias partitioning? Net *s* has cells in many groups?



Don't use p to bias the solution in either direction!

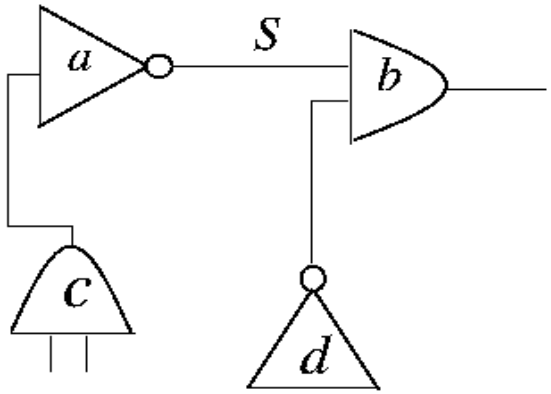


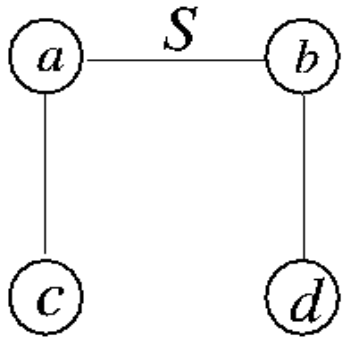
Use p!

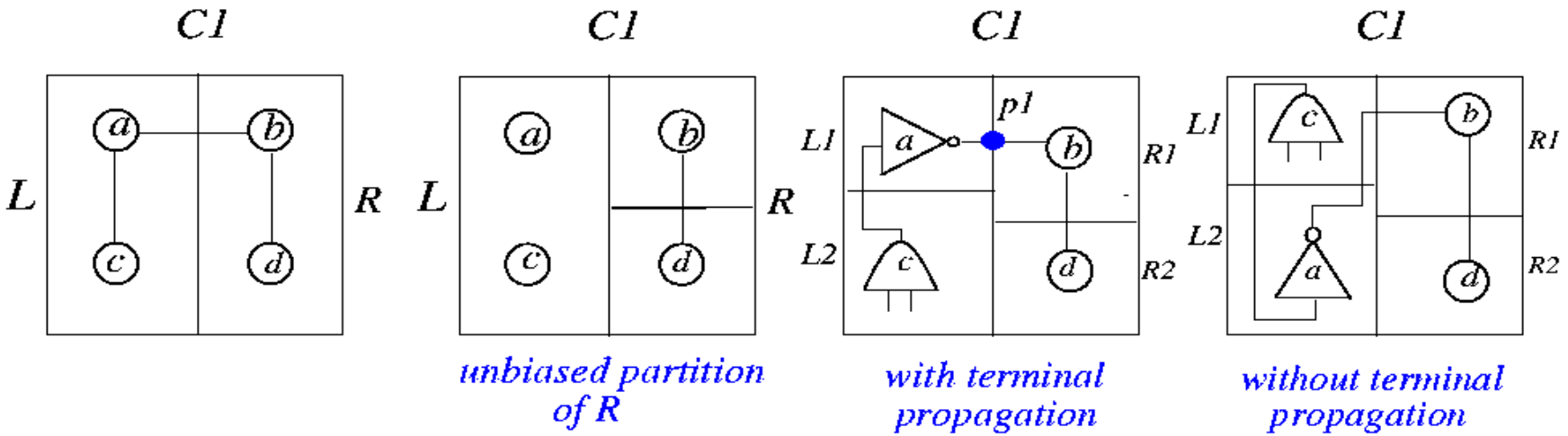


Terminal Propagation Example

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





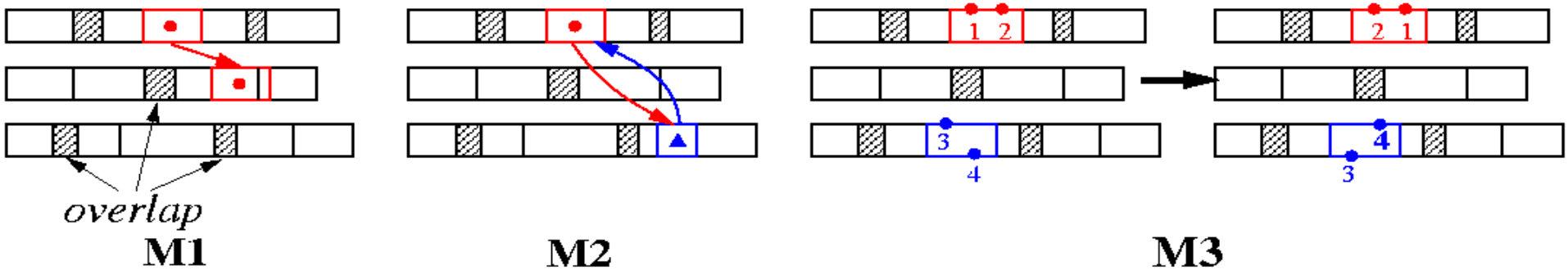


Placement by Simulated Annealing

- Sechen and Sangiovanni-Vincentelli, “The TimberWolf placement and routing package,” *IEEE J. Solid-State Circuits*, Feb. 1985; “TimberWolf 3.2: A new standard cell placement and global routing package,” DAC’86.
- TimberWolf: Stage 1
  - Modules are moved between different rows as well as within the same row.
  - Modules overlaps are allowed.
  - When the temperature is reached below a certain value, stage 2 begins.
- TimberWolf: Stage 2
  - Remove overlaps.
  - Annealing process continues, but only interchanges adjacent modules within the same row.

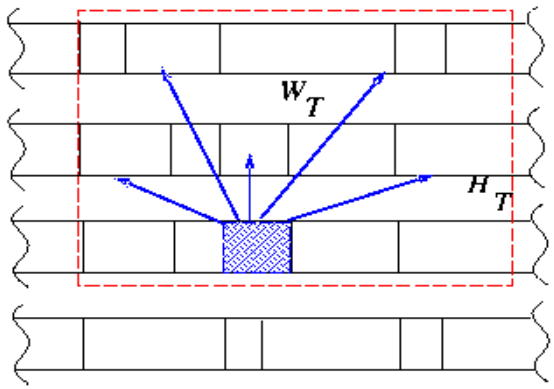
Solution Space & Neighborhood Structure

- **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.



Neighborhood Structure

- 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 very large, big enough to contain the whole chip.
  - Window size shrinks slowly as the temperature decreases. Height and width $\propto log(T)$.
  - Stage 2 begins when window size is so small that no inter-row module interchanges are possible.



Cost Function

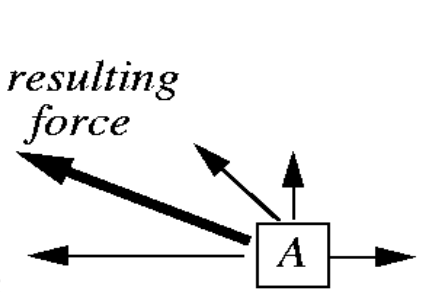
- Cost function: $C = C_1 + C_2 + C_3$.
- C_1 : **total estimated wirelength.**
  - $C_1 = \sum_{i \in Nets} (a_i w_i + b_i h_i)$
  - a_i, b_i are horizontal and vertical weights, respectively.
    - a_i=1, b_i=1 => $\frac{1}{2}$ x perimeter of the bounding box of Net i.
    - Critical nets: Increase both a_i and b_i.
    - If vertical wirings are "cheaper" than horizontal wirings, use smaller vertical weights: $b_i < a_i$.
- C_2: penalty function for module overlaps.
  - $C_2 = \gamma \sum_{i \neq j} O_{ij}^2$ 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.
  - $C_2 = \delta \sum_{r \in Rows} |L_r - D_r|$ δ: penalty weight.
  - D_r : desired row length.
  - L_r : sum of the widths of the modules in row r.

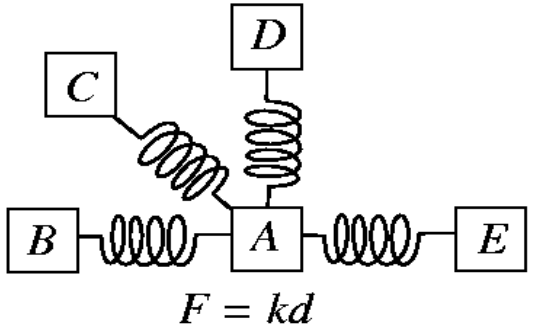
Annealing Schedule

- $T_k = r_k T_{k-1}$, $k = 1,2,3,\ldots$
- r_k increases from 0.8 to max value 0.94 and then decreases to 0.8.
- At each temperature, a total # of nP attempts is made. n: # of modules; P: user specified constant.
- Termination: $T < 0.1$.

Placement by the Force-Directed Method

- Hanan & Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems*, Breuer, Ed, 1972.
- Quinn, Jr. & Breuer, "A force directed component placement procedure for printed circuit boards," *IEEE Trans. Circuits and Systems*, June 1979.
- Reducing the placement problem to solving a set of simultaneous linear equations to determine equilibrium locations for cells.
- Analogy to Hooke's law: $F = kd$, F: force, k: spring constant, d: distance.
- Goal: Map cells to the layout surface.





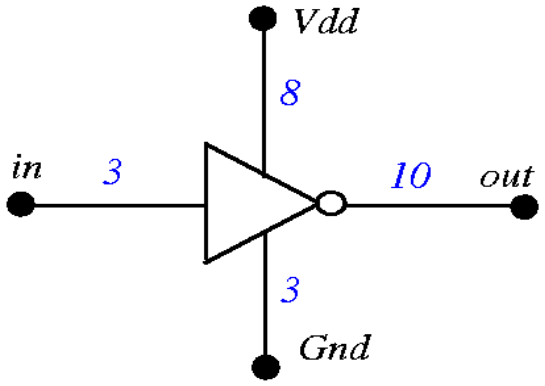
Finding the Zero-Force Target Location

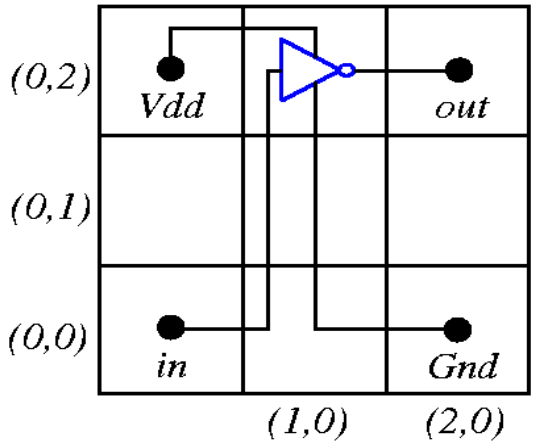
- Cell *i* connects to several cells *j*'s at distances d_{ij}'s by wires of weights w_{ij}'s. Total force: $F_i = \sum_j w_{ij} d_{ij}$
- The zero-force target location $(\hat{x}_i, \hat{y}_i)$ can be determined by equating the *x*- and *y*-components of the forces to zero:

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

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

- In the example, $\hat{x}_i = \frac{8*0+10*2+3*0+3*2}{8+10+3+3} = 1.083$ and $\hat{y}_i = 1.50$.





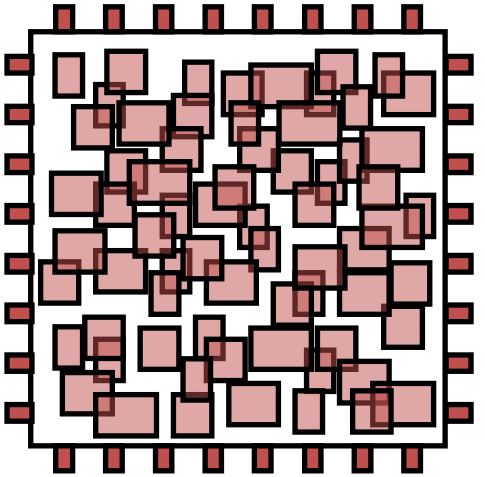
Force-Directed Placement

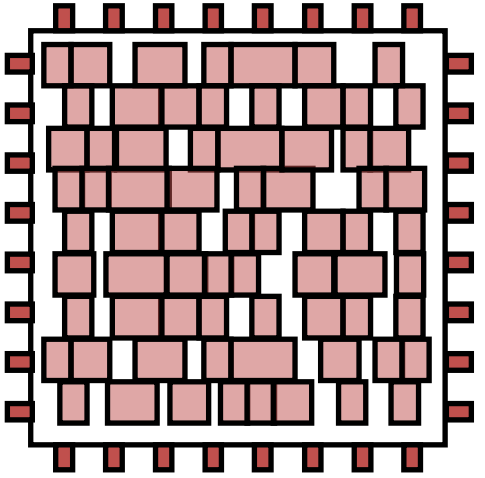
- An iterative improvement approach:
  - Start with an initial placement.
  - Repeat the following until convergence.
    - Select a "most profitable" cell *p* (e.g., maximum *F*, critical cells) and place it in its zero-force location.
    - "Fix" placement if the zero-location has been occupied by another cell *q*.
      - Options:
        - **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*.

Steps for Modern Circuit Placement

- Placement is usually divided into several more manageable steps:

1. **Global placement**
   - Generating a rough placement solution that may violate some placement (e.g., non-overlapping) constraints
2. **Legalization**
   - Modifying the rough placement to satisfy all constraints by local module movement
3. **Detailed placement**
   - Further improving the legalized solution by some local techniques





Placement Objectives

- Total wirelength
- Routability
  - Congestion-driven placement
- Performance
  - Timing-driven placement
- Power consumption
  - Power-driven placement
- Heat distribution
  - Thermal-driven placement

Underlying Issues for All Formulations

- Underlying issues for all variations of placement formulations are the same:
  - **Wirelength needs to be minimized** for different objectives
  - **Modules have to be properly distributed** for different design styles and for thermal-driven placement
- Will focus on total wirelength (HPWL) minimization

Other Works in Min-Cut Placement

- **Capo** [DAC-00]
  - http://vlsicad.eecs.umich.edu/BK/PDtools/Capo/
  - Standard cell placement, fixed-die context
  - Pure recursive bisectioning placer
  - Several techniques to produce good bisections
  - Produce good results mainly because
    - Breakthroughs in mincut bisection
    - Pay attention to details in implementation
  - Implementation with good interface (LEF/DEF and GSRC bookshelf)
  - A lot of extensions and improvements since the first implementation
- **Fengshui** [GLSVLSI-01,DAC-01,ICCAD-03]

Capo

- Recursive bisection framework
  - Multi-level FM for instances with >200 cells
  - Flat FM for instances with 35-200 cells
  - Branch-and-bound for instances with <35 cells
- Careful handling of partitioning tolerance
  - Uncorking: prevent large cells from being the first modules in a bucket of the FM algorithm
  - Repartitioning: several FM calls with decreasing tolerance
  - Block splitting heuristics: Higher tolerance for vertical cut
  - Hierarchical tolerance computation: instance with more whitespace can have a bigger partitioning tolerance

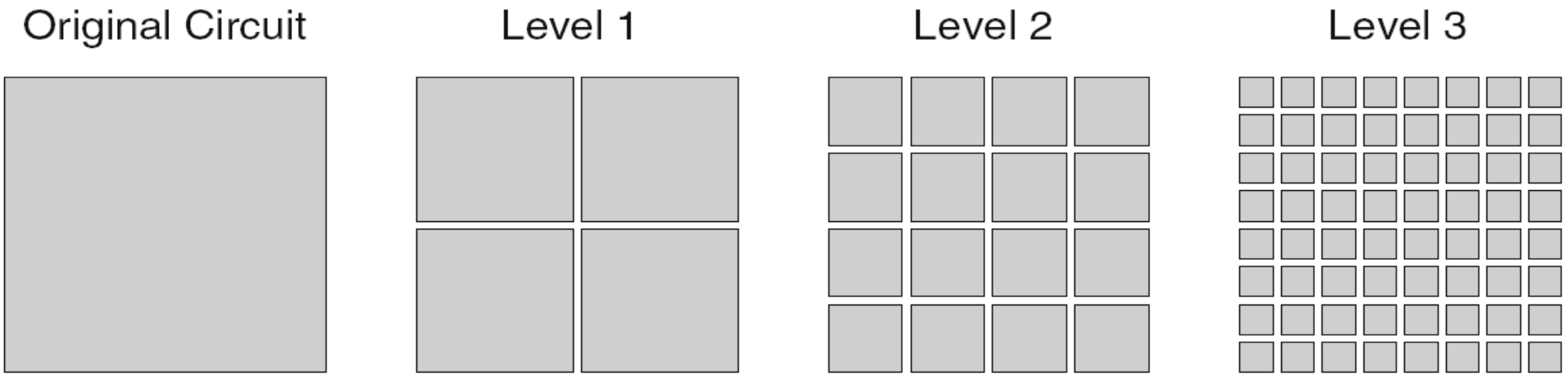
Hybrid Approach: Partitioning + Simulated Annealing

- Simulated annealing based placement produces good solution for small circuits
- But it becomes slow and poor in quality for larger circuits

- **Dragon** [ICCAD-00] takes a hybrid approach that combines simulated annealing and partitioning
  - Recursive partitioning to reduce the problem size
  - Simulated annealing to refine the solution generated by partitioning

Dragon

- Top-down hierarchical placement
  - hMetis to recursively quadrisect into 4^h bins at level h
  - Swapping of bins at each level by SA to minimize WL
  - Terminate when each bin contains < 7 cells
  - Then at the final stage of GP, switch single cells locally among bins by SA to further minimize WL
- Detailed placement is done by a greedy algorithm



Analytical Approach

- Write the placement objective and possibly placement constraints as *analytical* functions of module coordinates
- Therefore, formulate the placement problem as a mathematical program
- Many variations
  - Different ways to formulate the problem
  - Different ways to solve the resulting mathematical program

HPWL as Analytical Function

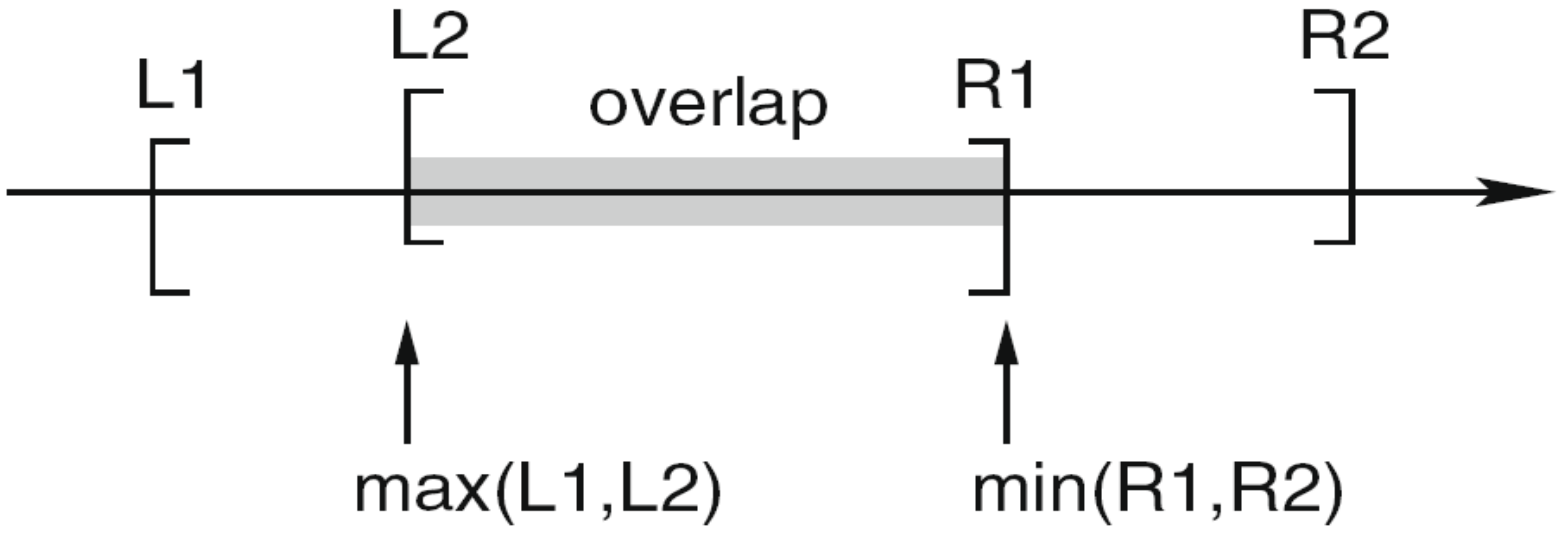
- HPWL of net e

$$\begin{aligned}\mathrm{HPWL}_e(x_1, \ldots, x_n, y_1, \ldots, y_n) &= \left(\max_{i \in e}\{x_i\} - \min_{i \in e}\{x_i\}\right) \\ &+ \left(\max_{i \in e}\{y_i\} - \min_{i \in e}\{y_i\}\right)\end{aligned}$$

Overlapping Area as Analytical Function

- First, define:

$$\Theta([L1, R1], [L2, R2]) = [\min(R1, R2) - \max(L1, L2)]^+$$



Overlapping Area as Analytical Function (cont'd)

- Overlapping area between modules *i* & *j*:

$$\text{Overlap}_{ij}(x_i,\ y_i,\ x_j,\ y_j)$$

$$= \Theta\left(\left[x_i - \frac{\omega_i}{2},\ x_i + \frac{\omega_i}{2}\right], \left[x_j - \frac{\omega_j}{2},\ x_j + \frac{\omega_j}{2}\right]\right)$$

$$\Theta\left(\left[y_i - \frac{h_i}{2},\ y_i + \frac{h_i}{2}\right], \left[y_j - \frac{h_j}{2},\ y_j + \frac{h_j}{2}\right]\right)$$

An Exact (but Impractical) Formulation

Minimize $\sum_{e \in E} c_e \times \text{HPWL}_e(x_1, \ldots, x_n, y_1, \ldots, y_n)$

Subject to $\text{Overlap}_{ij}(x_i, y_i, x_j, y_j) = 0$ for all $i, j \in V$ s.t. $i \neq j$

$$0 \leq x_i - \frac{\omega_i}{2}, \; x_i + \frac{\omega_i}{2} \leq W \text{ for all } i \in V$$

$$0 \leq y_i - \frac{h_i}{2}, \; y_i + \frac{h_i}{2} \leq H \text{ for all } i \in V$$

Problems of the Exact Formulation

- The functions $Overlap_{ij}(x_i, y_i, x_j, y_j)$ for $i, j \in V$ are highly nonconvex and not differentiable
- The functions $HPWL_e(x_1, \ldots, x_n, y_1, \ldots, y_n)$ for $e \in E$ are not differentiable (although convex)
- There are $O(n^2)$ constraints
  - n is up to (tens of) millions in modern designs

Practical Analytical Placement Algorithms

- Global placement
  - Wirelength is approximated by some differentiable and convex functions
  - Nonoverlapping constraints are usually replaced by some simpler constraints to make the module distribution roughly even
- Legalization is performed to eliminate module overlaps
- Detailed placement is applied to refine the solution with a more accurate wirelength metric

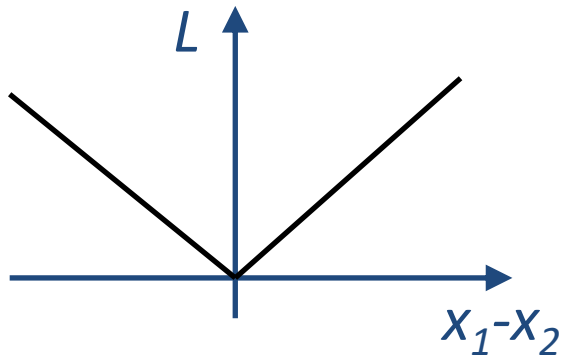
Techniques for Analytical Placement

- **Quadratic** techniques
  - Transformed into a sequence of convex quadratic programs
    - convex quadratic program: a mathematical program with a convex and quadratic objective function and linear constraints
- **Non-quadratic** techniques
  - Transformed into a single general mathematical program

Quadratic Wirelength

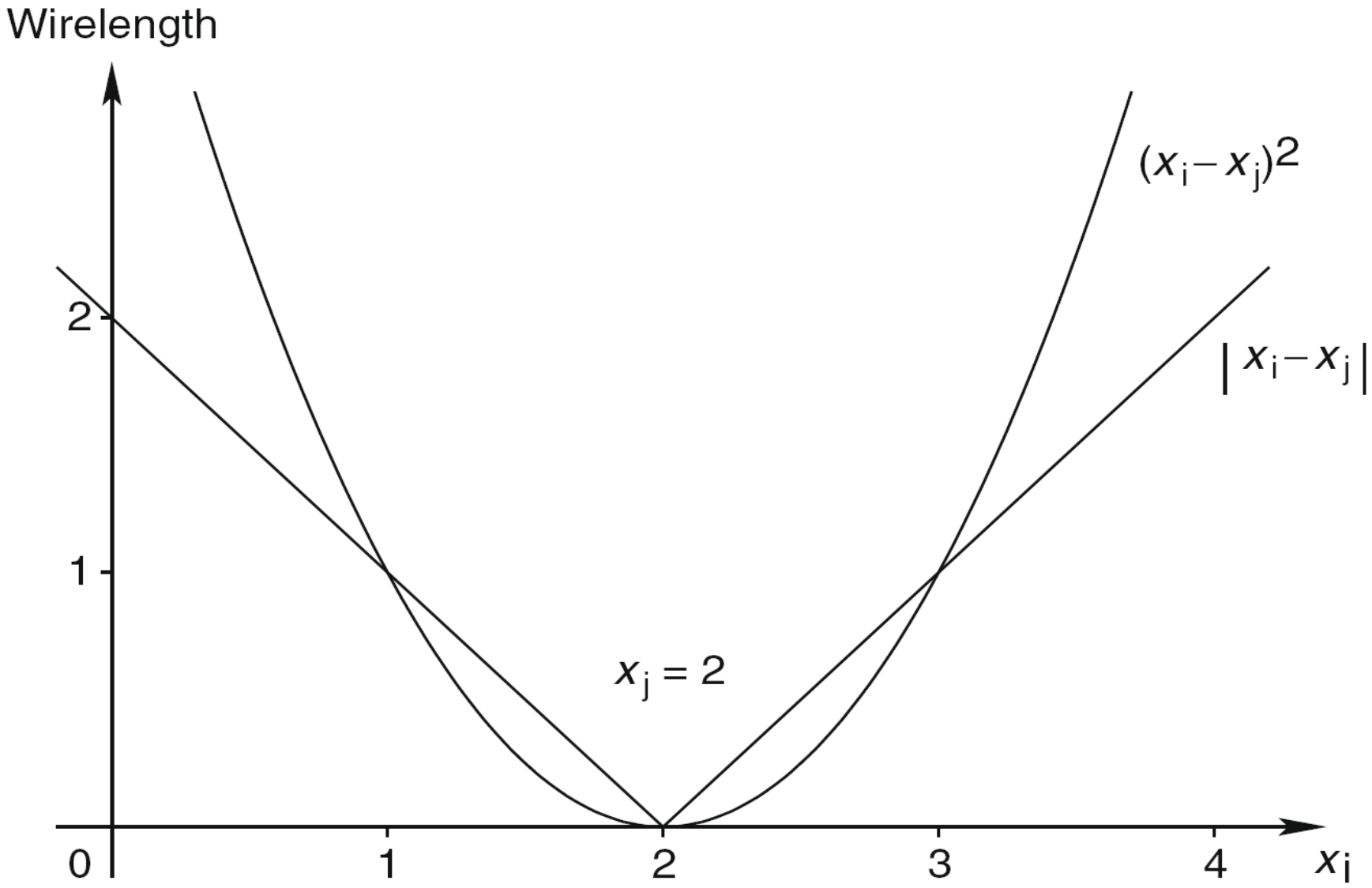
- WL (for 2-pin nets) can be written as a piece-wise linear function: $L = |x_1 - x_2|$ (in x direction)
- WL minimization can be written as a LP

$$\begin{aligned} &\text{min. } L \\ &\text{s.t.} \quad x_1 - x_2 \le L \\ &\qquad\; x_2 - x_1 \le L \end{aligned}$$



- However, quadratic WL minimization is more common: $\tilde{L} = (x_1 - x_2)^2$
  - Smooth function
  - Convex function → easy to minimize
  - Correlates well with linear WL
  - Often called **quadratic placement**

Quadratic WL vs. Linear WL



Cost Function of Quadratic Placement

Let (x_i, y_i) = Coordinates of the center of cell i

c_{ij} = Weight of the net between cell i and cell j

$\mathbf{x}, \mathbf{y}$ = Solution vectors

Cost of the net between cell i and cell j

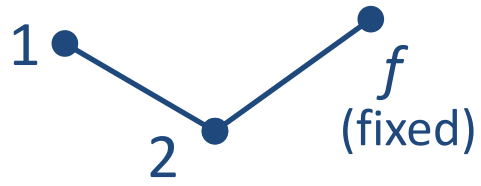
$$\tilde{L}_{\{i,j\}} = \frac{1}{2} c_{ij} \left((x_i - x_j)^2 + (y_i - y_j)^2 \right)$$

$$\text{Total cost } \tilde{L} = \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + d_x^T \mathbf{x} + \frac{1}{2} \mathbf{y}^T Q \mathbf{y} + d_y^T \mathbf{y} + \text{const}$$

Horizontal cost

$$= \frac{1}{2} c_{12} (x_1 - x_2)^2 + \frac{1}{2} c_{2f} (x_2 - x_f)^2$$

$$= \frac{1}{2} \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} c_{12} & -c_{12} \\ -c_{12} & c_{12} + c_{2f} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ -c_{2f} x_f \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \frac{1}{2} c_{2f} x_f^2$$



Solution of Quadratic Placement

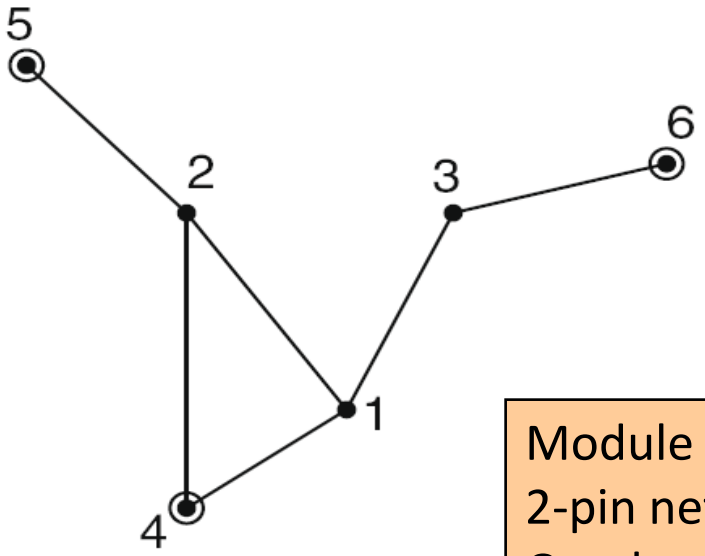
$$\text{Total cost } \tilde{\tilde{L}} = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \boldsymbol{d}_x{}^T\mathbf{x} + \frac{1}{2}\mathbf{y}^T Q\mathbf{y} + \boldsymbol{d}_y{}^T\mathbf{y} + \text{const}$$

- The problems in x- and y-directions can be separated and solved independently
  - *Ignore non-overlapping and other constraints*
  - Minimize $\tilde{\tilde{L}}_x = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \boldsymbol{d}_x{}^T\mathbf{x}$
  - Q can be proved to be positive and definite => the cost function is convex
  - Minimum solution can be found by setting derivatives to 0:

$$Q\mathbf{x} + \boldsymbol{d}_x{}^T = \mathbf{0}$$

Force Interpretation of Quadratic WL

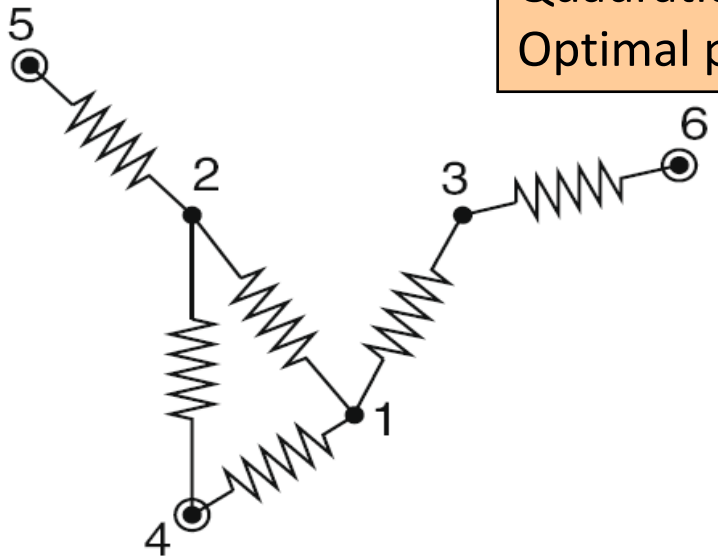
Circuit



◉ Fixed module
• Movable module

Module → Object
2-pin net → Spring
Quadratic WL → Spring potential energy
Optimal placement → Force equilibrium

Spring system



◉ Fixed object
• Movable object

Force Calculation

- Hooke's Law:
  - Force = Spring Constant x Displacement
- Can consider forces in x- and y-direction separately

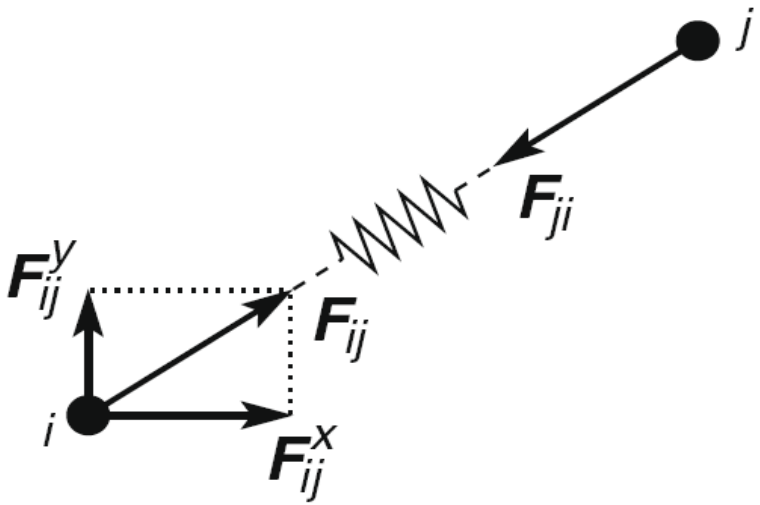
$$\text{Distance } d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

$$\text{Net Cost } c_{\{i,j\}}$$

$$\left|\mathbf{F_{ij}}\right| = c_{\{i,j\}} \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

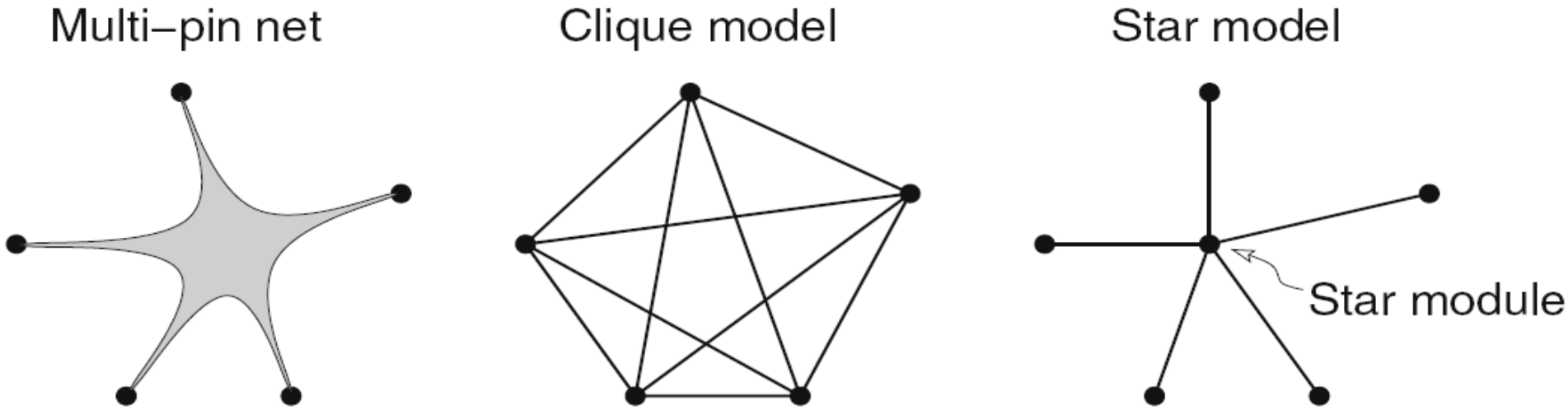
$$\left|\mathbf{F_{ij}^x}\right| = c_{\{i,j\}} \times \left|x_j - x_i\right|$$

$$\left|\mathbf{F_{ij}^y}\right| = c_{\{i,j\}} \times \left|y_j - y_i\right|$$



Net Models for Multi-Pin Nets

- Multi-pin net is modeled as several 2-pin nets



Hybrid net model (best)

# pins	Net Model
2	Clique
3	Clique
4	Star
5	Star
6	Star
…	…

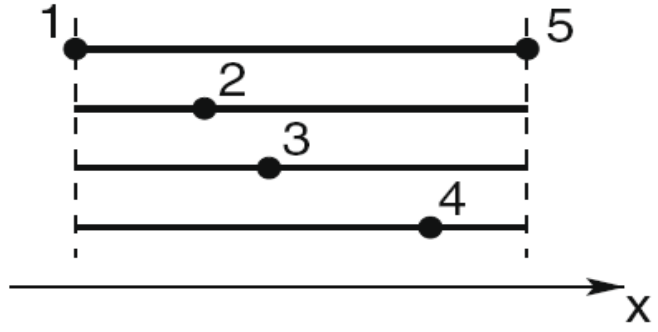
Linearization Method in GORDIAN-L [DAC-91]

- Linear WL (star model): $L^{star} = \sum_{e \in E} \sum_{i \in e} |x_i - x_e|$

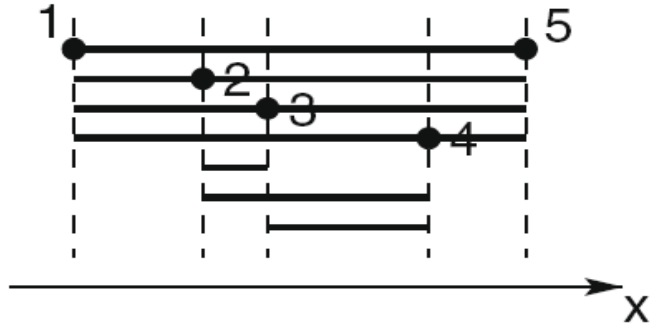
- Consider the function: $\tilde{L}^{star} = \sum_{e \in E} \sum_{i \in e} \frac{(x_i - x_e)^2}{g_{ie}}$
  - Exact if $g_{ie} = |x_i - x_e|$
  - Quadratic if g_{ie}'s are set to constant
  - L^{star} can be optimized iteratively by setting g_{ie} in current iteration according to the coordinates of previous iteration
  - In practice, set $g_{ie} = \sum_{i \in e} |x_i - x_e|$ for all $i \in e$

BoundingBox Net Model in Kraftwerk [ICCAD-06]

- Make use of preceding linearization idea
- Accurately model HPWL
- For a *k*-pin net:



BoundingBox



Clique

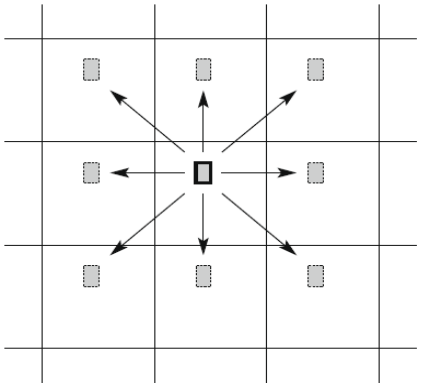
$$\tilde{L}^{BB} = \frac{1}{2} \sum_{\{i,j\} \in N} \omega_{\{i,j\}} \times (x_i - x_j)^2 \quad \text{where} \quad \omega_{\{i,j\}} = \frac{2}{k-1} \times \frac{1}{l_{\{i,j\}}}$$

If $l_{\{i,j\}}$ is set to $|x_i - x_j|$ for all $\{i, j\} \in N$, $\tilde{L}^{BB} = |x_1 - x_k|$

Iterative Local Refinement (ILR) in FastPlace [ISPD-04]

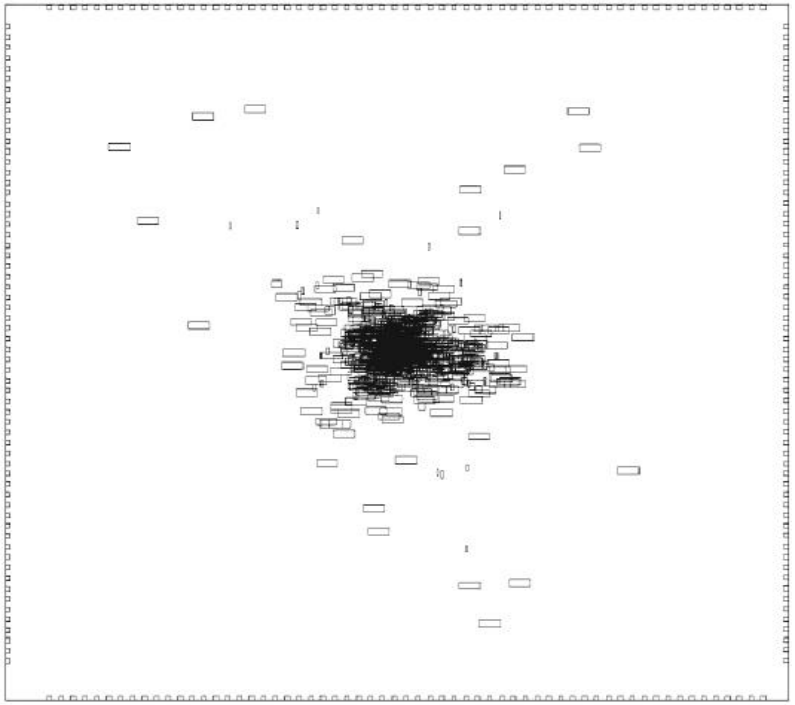
- Mitigate the inaccuracy of quadratic WL by refining global placement with accurate WL metric
  - Divide the placement region into bins by a regular grid
  - Examine modules one by one
  - Tentatively move a module

    to its eight adjacent bins
  - Compute a score for each tentative move
    - HPWL reduction
    - Cell densities at the source and destination bins
  - Take the move with the highest positive score (no move if all scores are negative)
  - Repeat until there is no significant improvement



Ignoring Nonoverlapping Constraints

- Consider WL minimization alone
  - If no fixed pins, a trivial solution is to place all modules at the same place
  - If there are fixed pins (e.g., I/O pins at boundary), it tends to get a lot of overlaps at the center of the placement

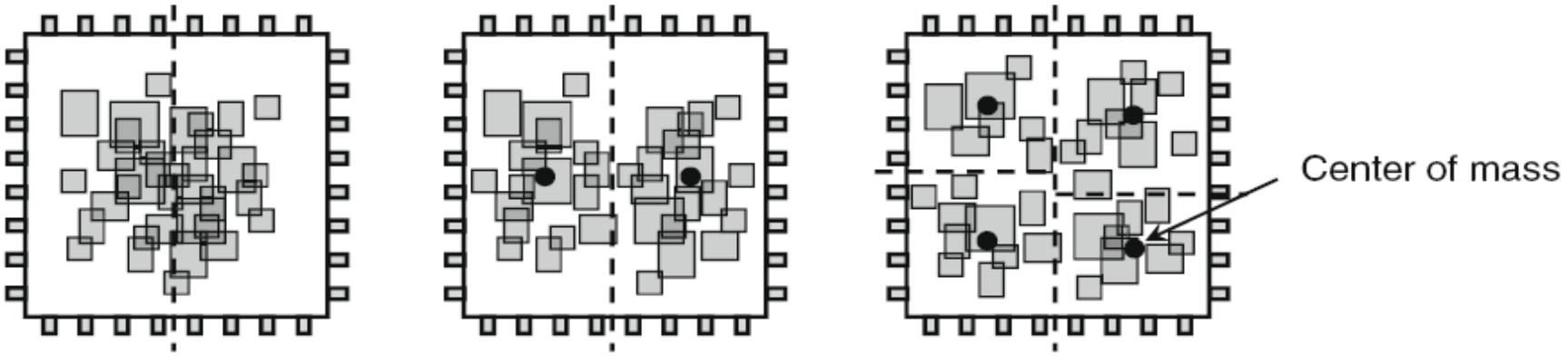


Handling Nonoverlapping Constraints

- Ways to make module distribution more even in quadratic placement
  - Adding center-of-mass constraints
  - Adding forces to pull modules from dense regions to sparse regions
- Constraints/forces are added in **an iterative manner** to gradually spread out the modules
- Transformed into a sequence of convex quadratic programs

Center-of-Mass Constraints in GORDIAN [TCAD-91]

- Given an uneven global placement solution
  - Find a good cut direction and position
    - Improve the cut value using FM
  - For each partition, add constraints that the center of gravity of cells should be in the center of region
    - The constraints are linear
  - Then perform quadratic placement again
    - Therefore, solving a single convex QP



Density-based Force by Kraftwerk [ICCAD-06]

- Pull cells away from dense to sparse regions
- Definitions:
  - $\boldsymbol{x}'$= vector of **current** placement positions
  - $\boldsymbol{x}$= vector of **new** placement positions to be determined
  - $\hat{\boldsymbol{x}}$= vector of **target** placement positions
    - Based on module density $D(x,y)$

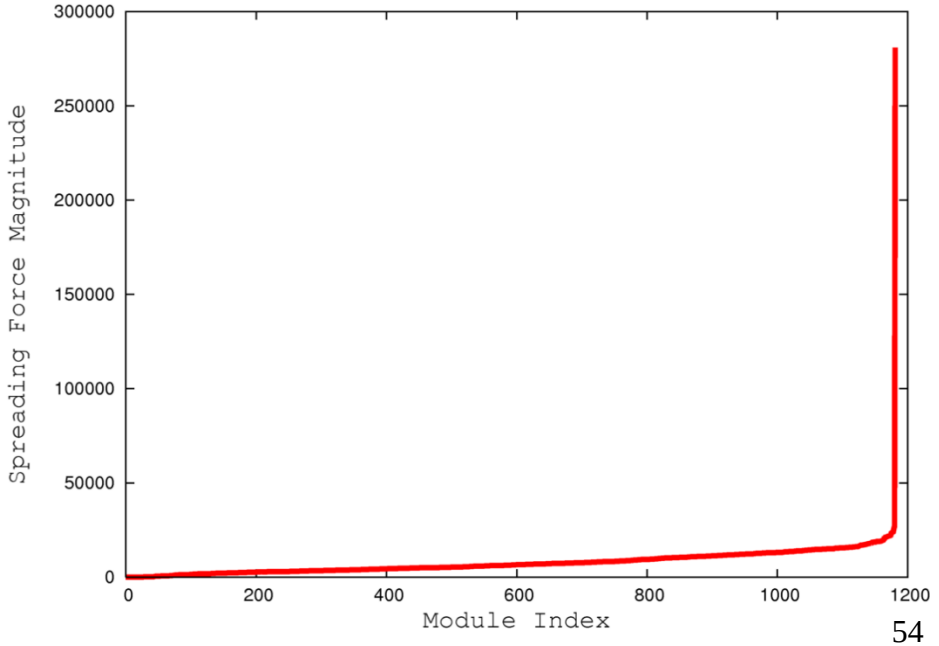
$$\Delta\phi = -D(x,\ y) \quad \hat{x}_i = x'_i - \left|\frac{\partial\Phi}{\partial x}\right|_{(x'_i,\ y'_i)}$$

- **Hold force**: $\boldsymbol{F}_x^{hold} = -(\boldsymbol{Q}\boldsymbol{x}' + \boldsymbol{d}_x)$
- **Move force**: $\boldsymbol{F}_x^{move} = \hat{\boldsymbol{Q}}(\boldsymbol{x} - \hat{\boldsymbol{x}})$ here $\hat{\boldsymbol{Q}} = \mathrm{diag}(\hat{c}_i)$
- **Force equilibrium**:

$$(\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x) - (\boldsymbol{Q}\boldsymbol{x}' + \boldsymbol{d}_x) + \hat{\boldsymbol{Q}}(\boldsymbol{x} - \hat{\boldsymbol{x}}) = 0$$

Force-Vector Modulation in RQL [DAC-07]

- Some additional spreading forces are huge
  - A module is pulled far away from its natural position
  - Causes significant increase in WL
  - Only a few percent of all additional forces are huge
- Idea: Nullifies the huge forces before next QP
  - Correcting mistakes made during spreading
  - Significant WL reduction
  - Minor effect in spreading



Non-Quadratic Techniques

- Formulate the placement problem as a single non-linear & non-quadratic program

$$\text{Minimize} \sum_{e \in E} c_e \times \text{WL}_e(x, y)$$

$$\text{Subject to } D_b(x, y) <= T_b \text{ for all bin } b$$

  - $WL_e()$ is continuously differentiable and more accurate in approximating HPWL than quadratic WL
  - Placement region is divided into **bins** such that non-overlapping constraints are replaced by **bin density constraints**

Choices of Wirelength Functions

HPWL $$\max_{v_i,v_j \in e, i<j} |x_i - x_j| + \max_{v_i,v_j \in e, i<j} |y_i - y_j|$$

Quadratic $$\sum_{e \in E} \Big(\sum_{v_i,v_j \in e, i<j} w_{ij} |x_i - x_j|^2 + \sum_{v_i,v_j \in e, i<j} w_{ij} |y_i - y_j|^2 \Big)$$

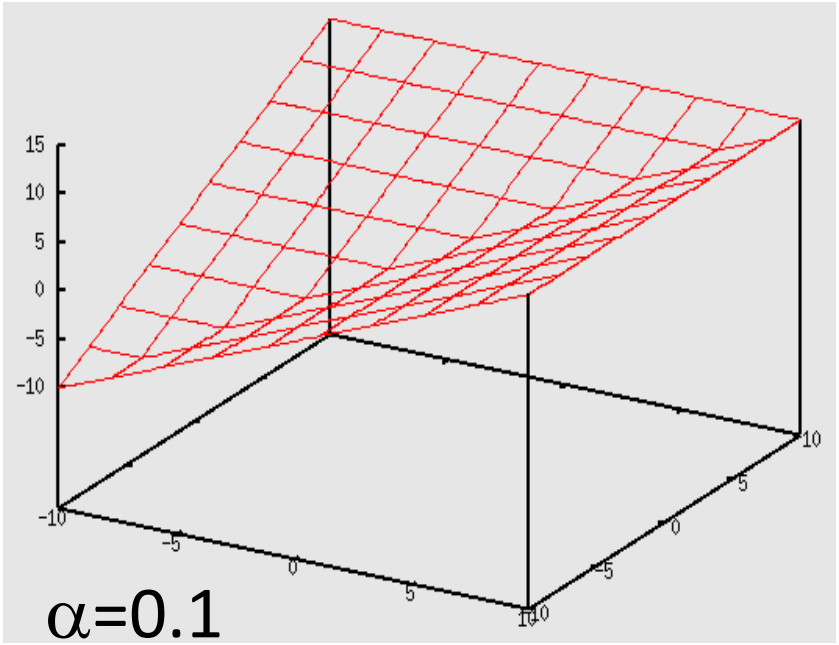
Log-Sum-Exp $$\eta \sum_{e \in E} \Big(\log \sum_{v_k \in e} \exp(x_k/\eta) + \log \sum_{v_k \in e} \exp(-x_k/\eta) + \log \sum_{v_k \in e} \exp(y_k/\eta) + \log \sum_{v_k \in e} \exp(-y_k/\eta) \Big)$$

Most common

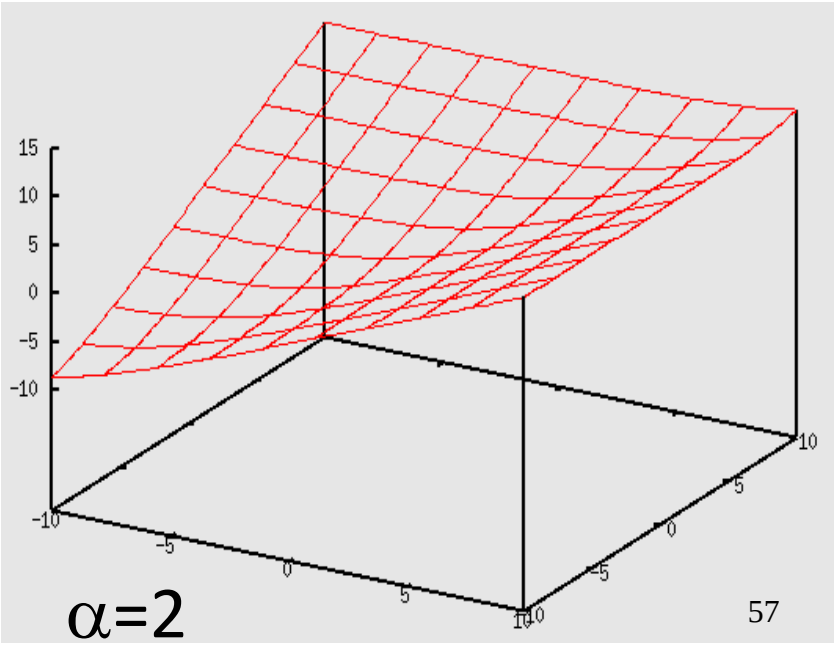
L_p-norm $$\sum_{e \in E} \Big(\big(\sum_{v_k \in e} x_k^p \big)^{\frac{1}{p}} - \big(\sum_{v_k \in e} x_k^{-p} \big)^{-\frac{1}{p}} + \big(\sum_{v_k \in e} y_k^p \big)^{\frac{1}{p}} - \big(\sum_{v_k \in e} y_k^{-p} \big)^{-\frac{1}{p}} \Big)$$

Log-Sum-Exponential (LSE) Function

- An approximation of the maximum function:
  - $\mathrm{LSE}_\alpha(z_1, \ldots, z_n) = \alpha \times \left(\log \left(\sum_{i=1}^{n} e^{z_i/\alpha} \right) \right) \approx \max(z_1, \ldots, z_n)$
  - Strictly convex and continuously differentiable
  - α : smoothing parameter (exact when $\alpha \rightarrow 0$)



α=0.1



α=2

Expression of HPWL using LSE Function

- HPWL in terms of max:

$$\begin{aligned}&\mathrm{HPWL}_e(x_1,\ldots,x_n,\ y_1,\ldots,y_n)\\ &= \left(\max_{i\in e}\{x_i\} - \min_{i\in e}\{x_i\}\right) + \left(\max_{i\in e}\{y_i\} - \min_{i\in e}\{y_i\}\right)\\ &= \left(\max_{i\in e}\{x_i\} + \max_{i\in e}\{-x_i\}\right) + \left(\max_{i\in e}\{y_i\} + \max_{i\in e}\{-y_i\}\right)\end{aligned}$$

- After approximating max by LSE:

$$\begin{aligned}&\mathrm{LSEWL}_{e,\alpha}(x_1,\ldots,x_n,\ y_1,\ldots,y_n)\\ &= \alpha \times \left(\log\left(\sum_{i\in e} e^{x_i/\alpha}\right) + \log\left(\sum_{i\in e} e^{-x_i/\alpha}\right)\right.\\ &\qquad \left. + \log\left(\sum_{i\in e} e^{y_i/\alpha}\right) + \log\left(\sum_{i\in e} e^{-y_i/\alpha}\right)\right)\end{aligned}$$

Density Constraint Smoothing

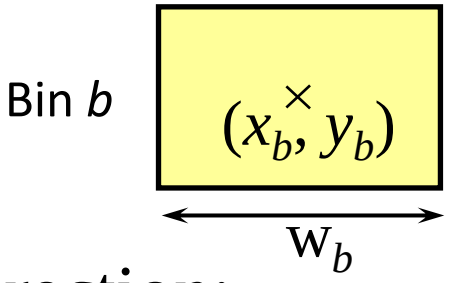
$$\text{Minimize} \sum_{e \in E} c_e \times \text{WL}_e(\boldsymbol{x}, \boldsymbol{y})$$

$$\text{Subject to } D_b(\boldsymbol{x}, \boldsymbol{y}) <= T_b \text{ for all bin } b$$

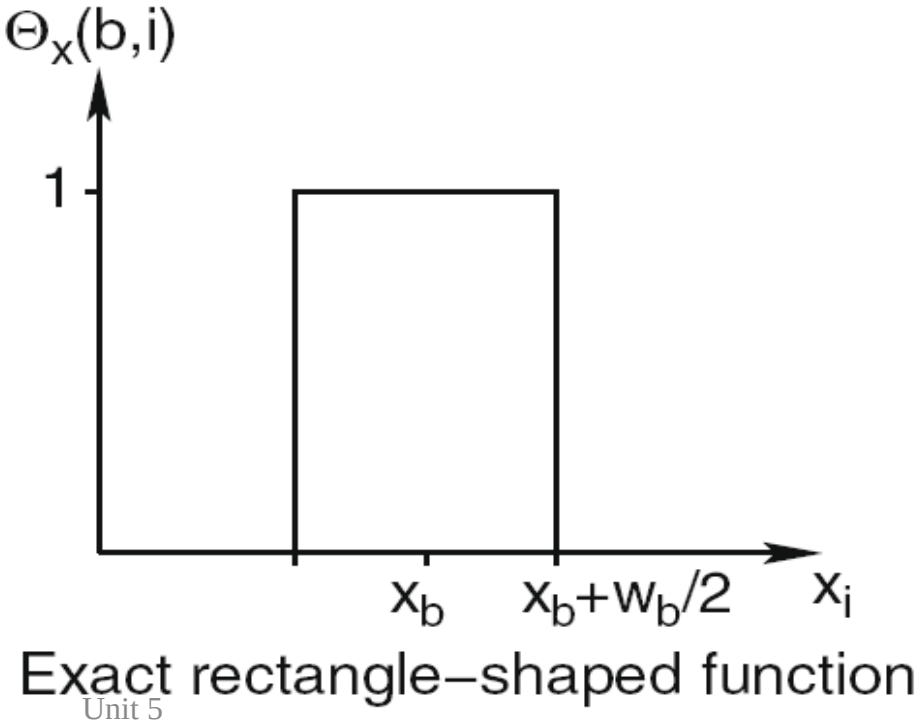
- $D_b(x,y)$ gives the density of bin b with respect to placement solution $\boldsymbol{x}$ and $\boldsymbol{y}$
- Need to be smoothed
- A smoothing technique
  - Bell-shaped function

Smoothing by Bell-Shaped Function

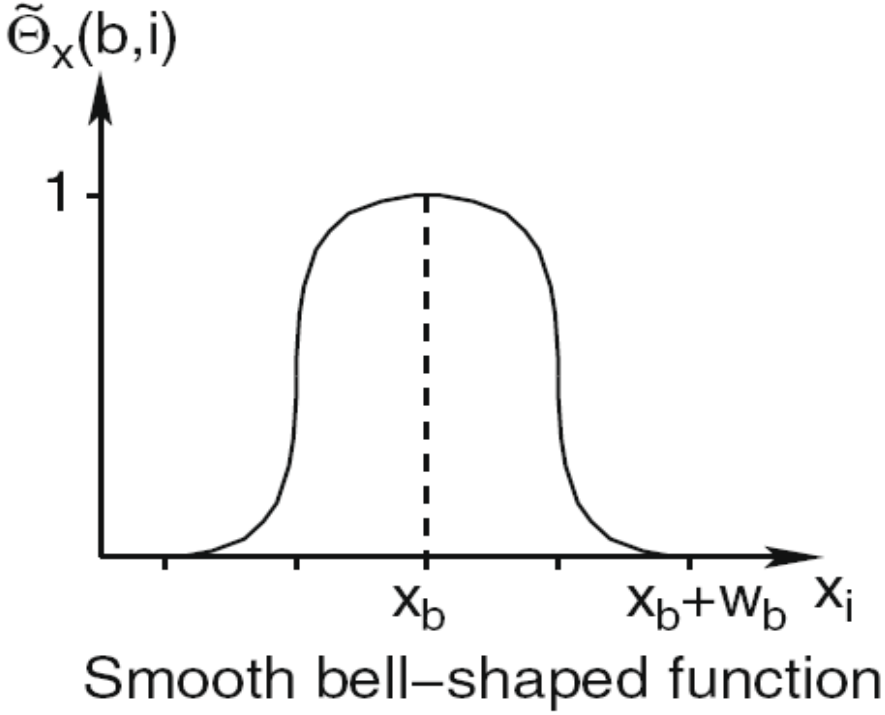
- Assume
  - Each module is much smaller than bins
  - Each module has a unit area



- Overlap between bin *b* and module *i* in x-direction:



Exact rectangle–shaped function



Smooth bell–shaped function

Equations for Bell-Shaped Function

- Let $d_x = |x_i - x_b|$

$$\tilde{\Theta}_x(b,i) = \begin{cases} 1 - 2 \times d_x^2/\omega_b^2 & \text{if } 0 \le d_x \le \omega_b/2 \\ 2 \times (d_x - \omega_b)^2/\omega_b^2 & \text{if } \omega_b/2 \le d_x \le \omega_b \\ 0 & \text{if } \omega_b \le d_x \end{cases}$$

$$D_b(x,\ y) = \sum_{i \in V} C_i \times \tilde{\Theta}_x(b,i) \times \tilde{\Theta}_y(b,i)$$

- Extension for large modules:

$$\tilde{\Theta}_x(b,i) = \begin{cases} 1 - a \times d_x^2 & \text{if } 0 \le d_x \le w_b/2 + w_i/2 \\ b \times (d_x - w_b - w_i/2)^2 & \text{if } w_b/2 + w_i/2 \le d_x \le w_b + w_i/2 \\ 0 & \text{if } w_b + w_i/2 \le d_x \end{cases}$$

$$\text{where } a = 4/((w_b + w_i)(2w_b + w_i))$$
$$b = 4/(w_b(2w_b + w_i))$$

Algorithms for Nonlinear Programs

- **Quadratic penalty method**
  - Convert into a sequence of unconstrained minimization problems
  - Each unconstrained problem can be solved by the **conjugate gradient method**

$$\text{Minimize} \sum_{e \in E} c_e \times \text{WL}_e(x, y) + \beta \times \sum_b (D_b(x, y) - T_b)^2$$

Extension to Multilevel Placement

- Three phases
  1. A hierarchy of coarser netlists is constructed
  2. An initial placement of the coarsest netlist is generated
  3. The netlist is successively unclustered, and placement at each level is refined

Clustering Technique: First Choice

- Traverse modules in an arbitrary order
- Each module *i* is clustered with an unclustered neighbor *j* with the largest affinity:

$$r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{|e| - 1}$$

- To reduce variation in cluster size:

$$r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{(|e| - 1) \times \text{area}(e)}$$

- To consider proximity information in initial placement:

$$r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{(|e| - 1) \times \text{area}(e) \times \text{dist}(i,\, j)}$$

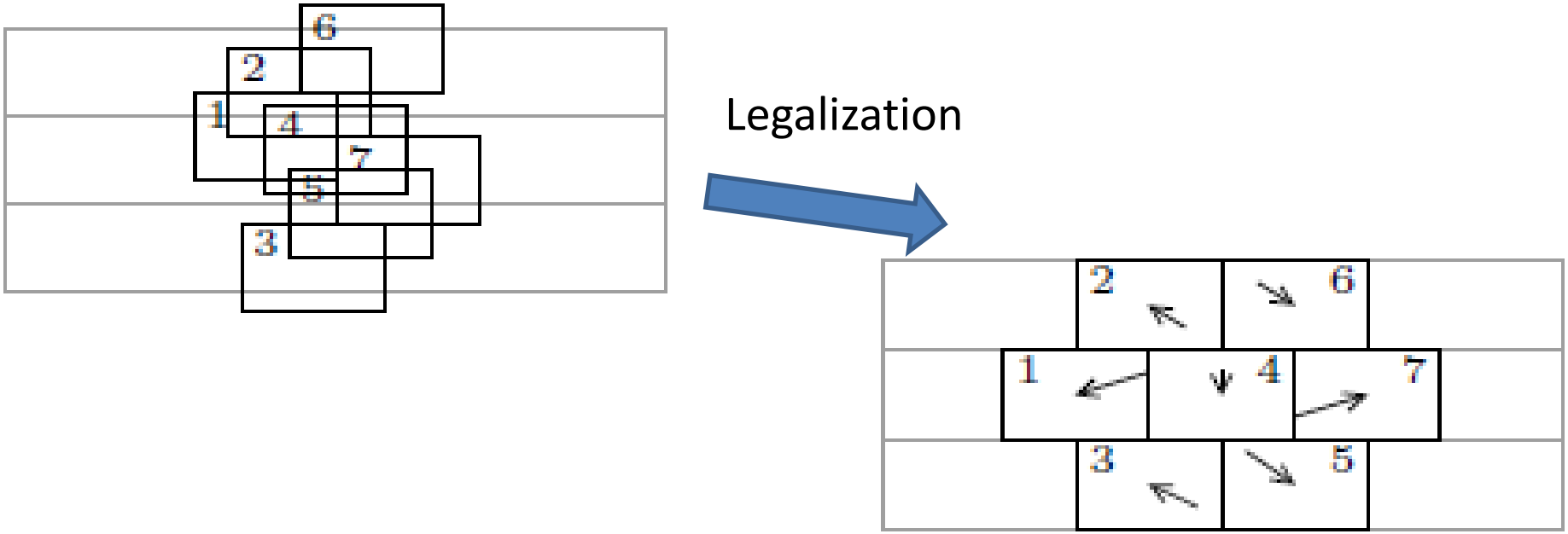
Clustering Technique: Best Choice

- Affinity: $r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{|e| \times (\text{area}(i) \times \text{area}(j))}$
- Always select the globally best pair of modules for clustering (and, in principle, update netlist immediately)
- In practice, **lazy updating technique** is proposed
  - Affinities affected by previous clustering are marked as invalid and are updated only after they have been selected for clustering
- Impose a hard upper limit for cluster size

Placement (Part II)

Legalization

- Legalization is a process to eliminate all overlaps (and assign cells to rows) by perturbing cells/modules as little as possible

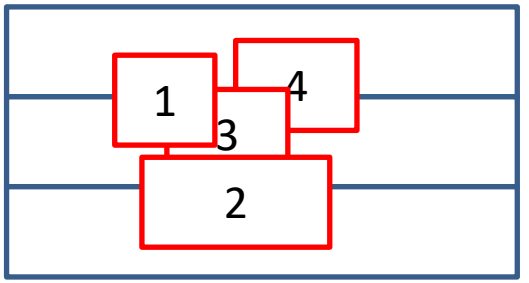


Greedy Methods

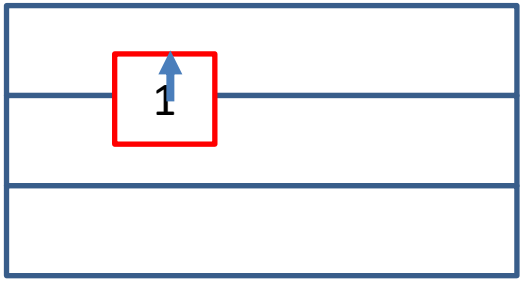
- Legalize one cell at a time
- **Tetris** [Hill, US Patent 6370673, 2002]
  - Sort cells in ascending x-coordinates
  - Pack cells to the left one at a time into a row so as to minimize the displacement
  - Already legalized cells are never moved again
  - Pros: Commonly used and extremely fast
  - Cons: May result in very uneven row length and may fail to pack all cells inside the placement region
- **Abacus** [Spindler, ISPD 2008]
  - Similar to Tetris
  - **Already legalized cells can be moved**
  - Get better results than Tetris

Example for Tetris

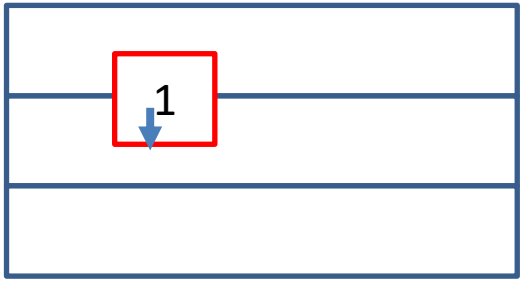
Global Placement

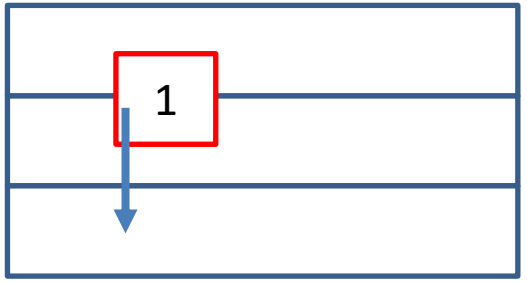


Legalize cell 1:

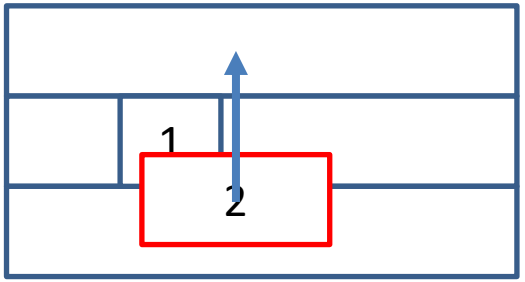


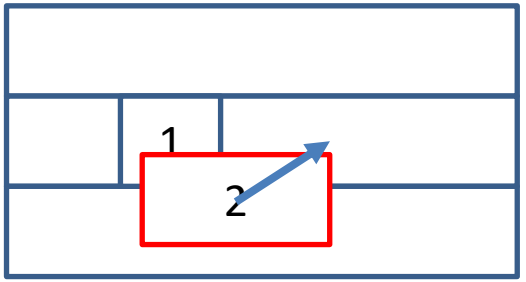
Pick Nearest Movement

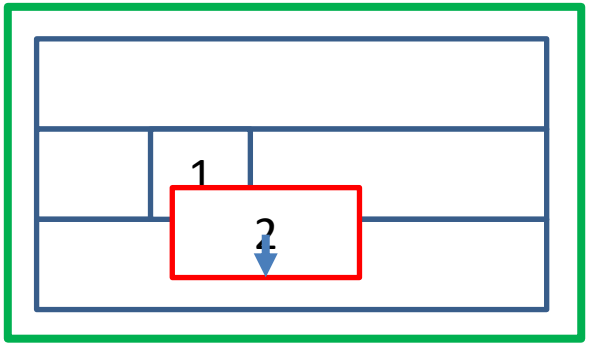




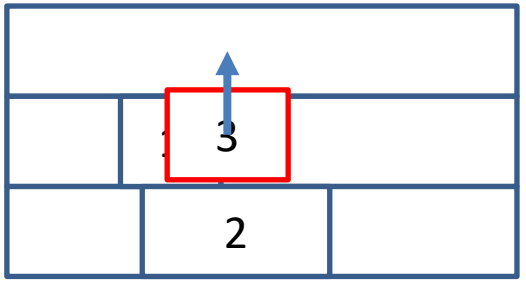
Legalize cell 2:

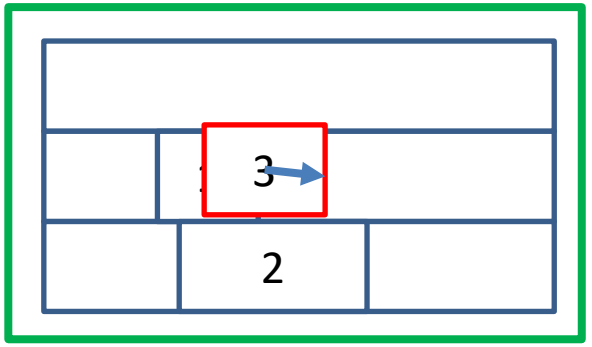


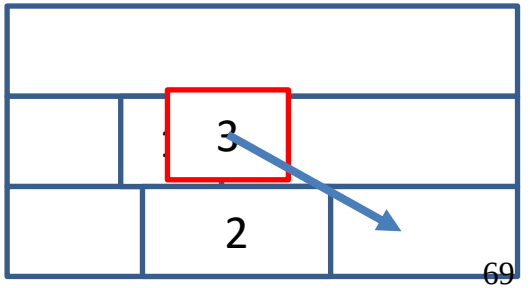




Legalize cell 3:



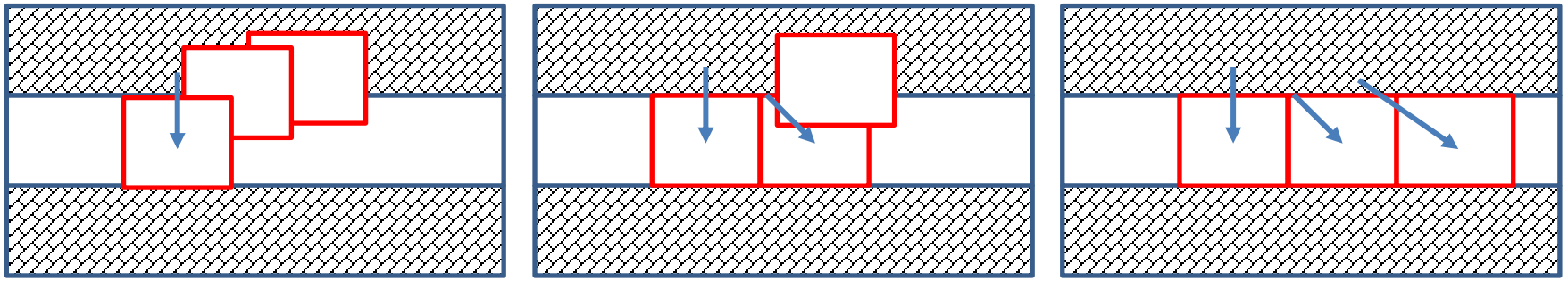




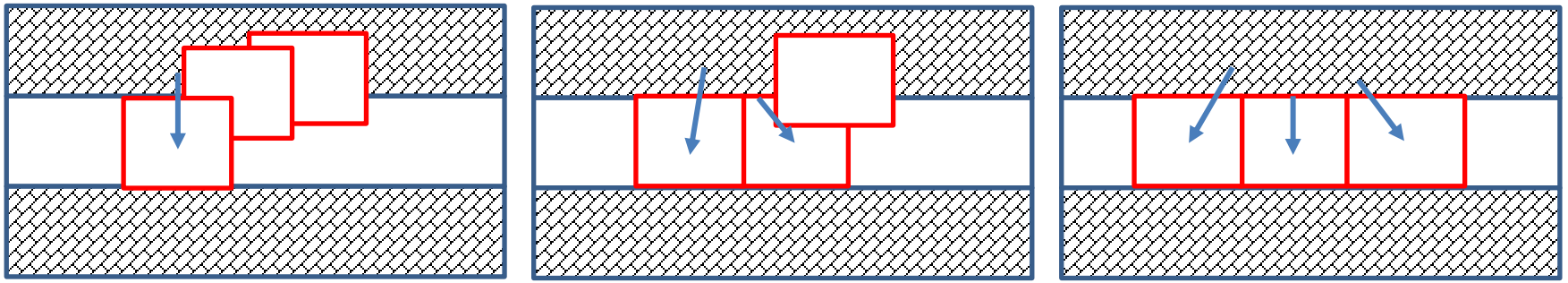
Abacus

- Similar to Tetris: sort cells and legalize one cell at a time
- Legalization of one cell: move the cell over the rows, and place the cell to the best/nearest row
- **PlaceRow** (difference from Tetris): move already legalized cells within one row to minimize total movement

Tetris:



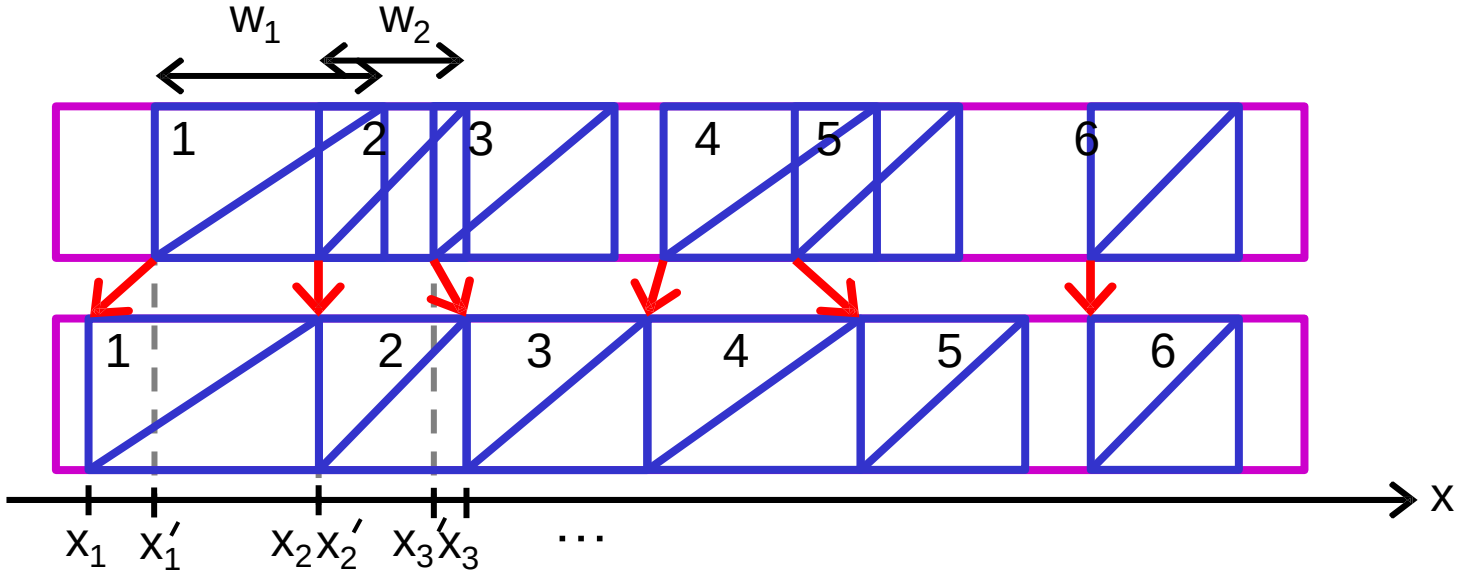
Abacus:



PlaceRow

Input: one row with N cells, x-pos of cells: global placement (x_i')

Output: new (legal) x-pos of cells (x_i) such that the overlap is removed and the total quadratic movement is minimized



QP:

$$\min \sum_{i=1}^{N} e_i \left(x_i - x_i'\right)^2 \quad \text{s.t.} \quad \underbrace{x_i \geq x_{i-1} + w_{i-1}}_{\text{no overlap}} \leftarrow \text{width}$$

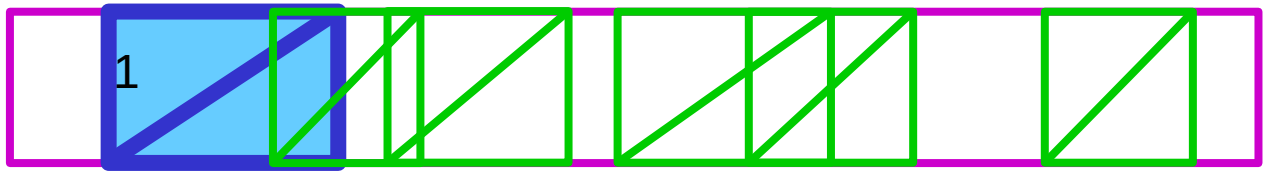
weight: e_i ↑ legal x-pos: x_i ↑ global x-pos: x_i'

PlaceRow: Dynamic Programming

PlaceRow:

- Solve QP by dynamic programming approach:
  solve sub problems optimally to obtain final solution
- Process cells from left to right

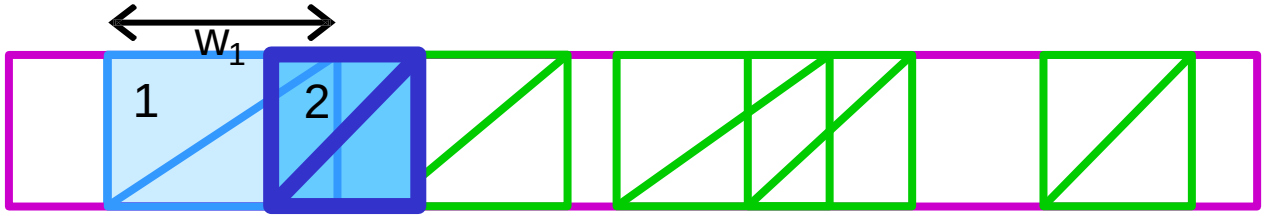
Cell 1:



first cell → do not move

PlaceRow: Cell 2

Cell 2:



overlap with previous cell? **yes** → **cluster** with previous cell

Clustering process:

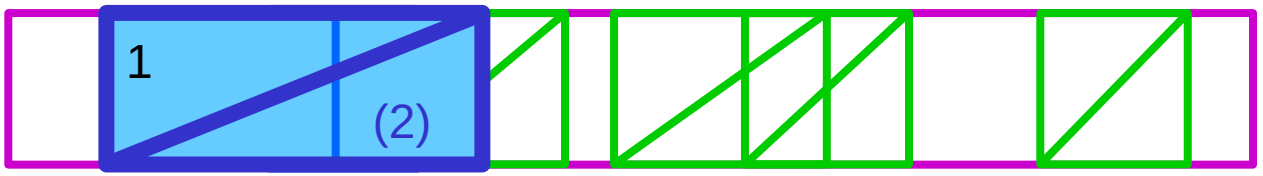
$$x_2 = x_1 + w_1$$

$$\min e_1(x_1 - x_1')^2 + e_2(x_2 - x_2')^2 \Rightarrow \min \underbrace{(e_1 + e_2)}_{\text{new } e_1}\left(x - \underbrace{\frac{e_1 x_1' + e_2(x_2' - w_1)}{e_1 + e_2}}_{\text{new } x_1'}\right)^2$$

Update x_1', e_1, and w_1:

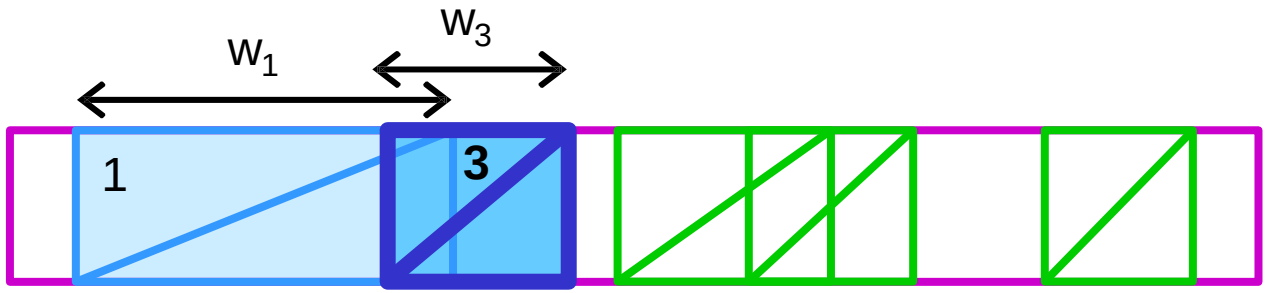
$$x_1' \leftarrow \frac{e_1 x_1' + e_2(x_2' - w_1)}{e_1 + e_2} \qquad e_1 \leftarrow e_1 + e_2 \qquad w_1 \leftarrow w_1 + w_2$$

Result: $\min e_1(x_1 - x_1')^2 \rightarrow x_1 = x_1'$



PlaceRow: Cell 3

Cell 3:



Overlap with previous cell? **Yes** → **cluster** with previous cell

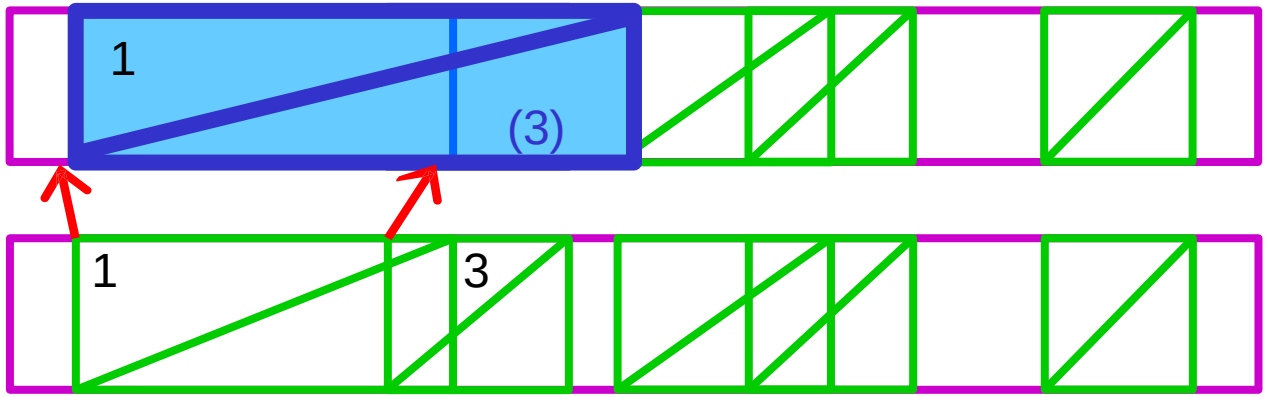
Update x_1', e_1, and w_1:

$$x_1' \leftarrow \frac{e_1 x_1' + e_3 (x_3' - w_1)}{e_1 + e_3} \qquad w_1 \leftarrow w_1 + w_3 \qquad e_1 \leftarrow e_1 + e_3$$

Move cell 1: $\min e_1 (x_1 - x_1)^2 \longrightarrow x_1 = x_1'$

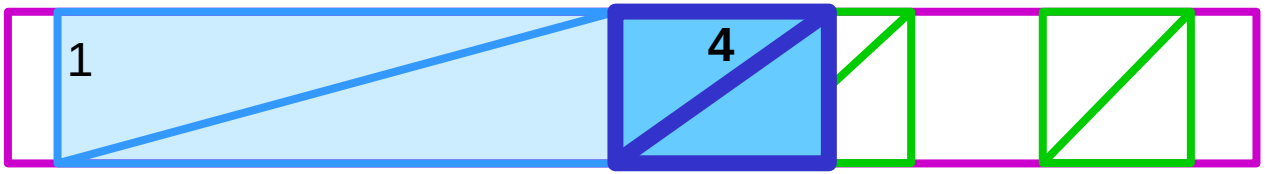
Result:

Prev:



PlaceRow: Cell 4

Cell 4:

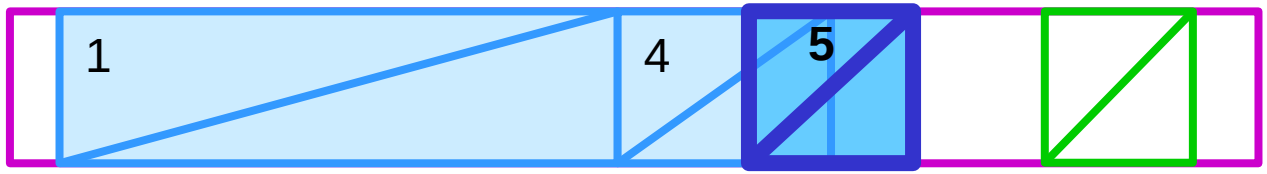


Overlap with previous cell? **No**

→ **no clustering, no movement**

PlaceRow: Cell 5

Cell 5:

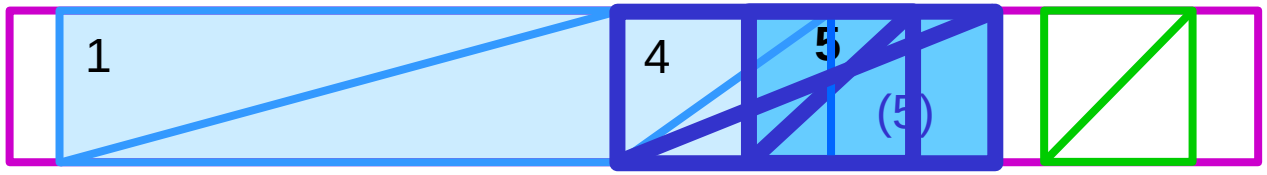


Overlap with previous cell? **yes** → **cluster** with previous cell 4

Update x_4', e_4, and w_4:

$$x_4' \leftarrow \frac{e_4 x_4' + e_5 (x_5' - w_4)}{e_4 + e_5} \qquad e_4 \leftarrow e_4 + e_5 \qquad w_4 \leftarrow w_4 + w_5$$

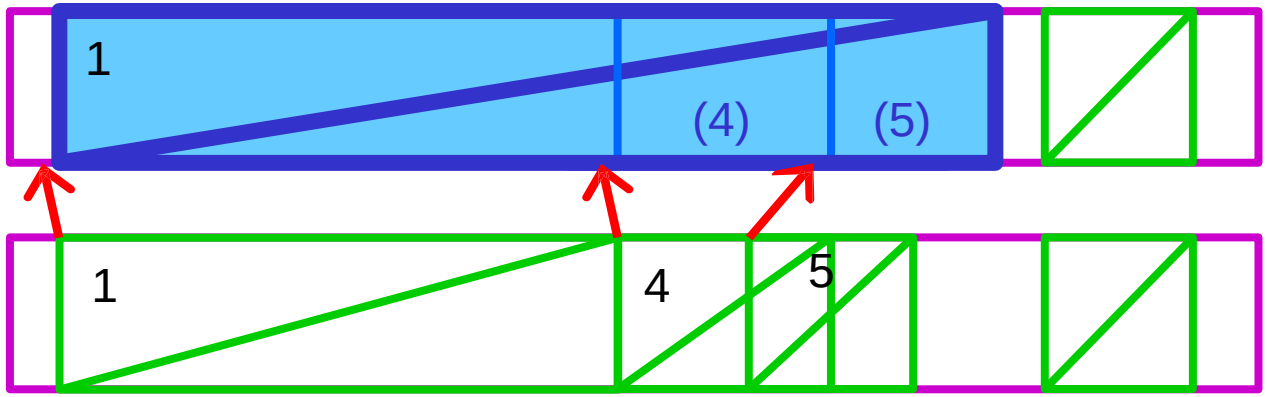
Move cell 4: $\min e_4 (x_4 - x_4')^2 \rightarrow x_4 = x_4'$



Overlap with previous cell? **yes** → **cluster** with previous cell 1

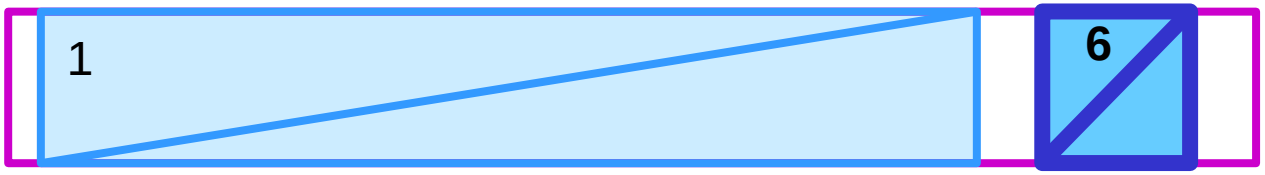
Result:

Prev:



PlaceRow: Cell 6

Cell 6:



Overlap with previous cell? **No**
→ **no clustering, no movement**

Last cell → done, PlaceRow finished

PlaceRow: Summary

PlaceRow:

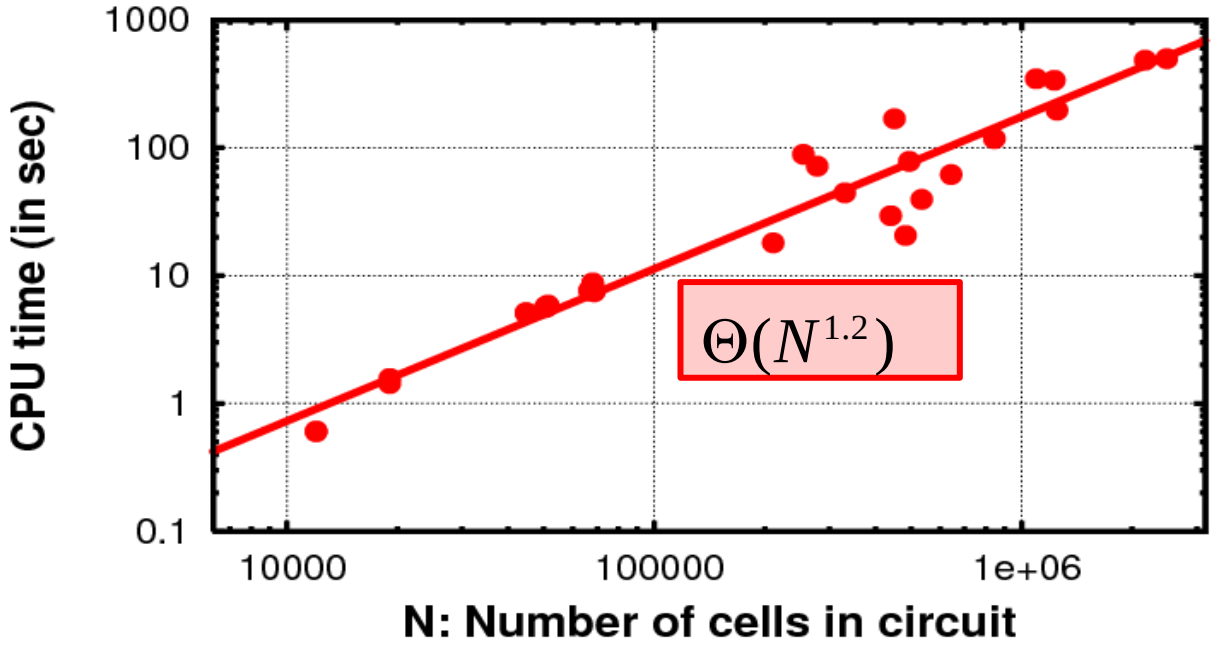
- Called several times for legalizing one cell
- Places cells aligned to one row:
minimize quadratic movement → quadratic program (QP)
- Solves QP by dynamic programming:
  - Process cells from left to right
  - If cell overlaps with previous cell:
    clustering → movement → further checks with left cells
  - Clustering: update width, weight, and global x-pos of cell
    constant execution time
- Linear worst-case complexity: O(N) N: number of cells in the row

At most N-1 clustering operations for N cells

Complexity

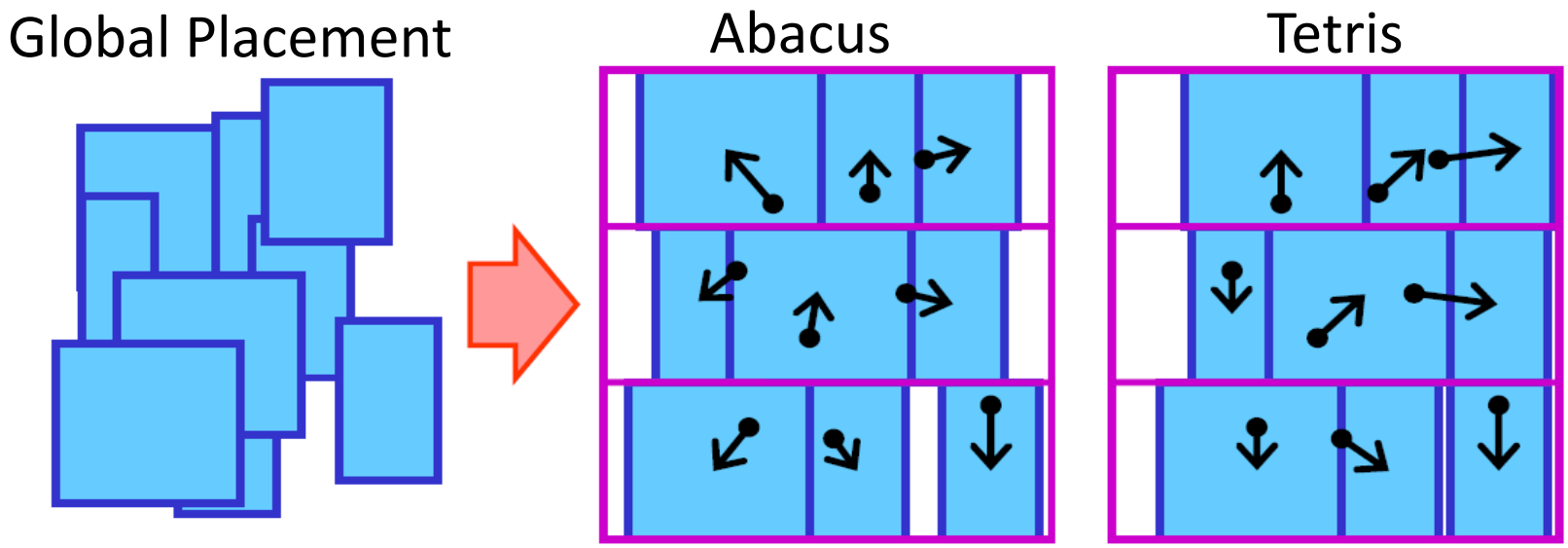
Worst-case for a complete circuit with N cells: $O(N^2)$

Average-case (experimental results):



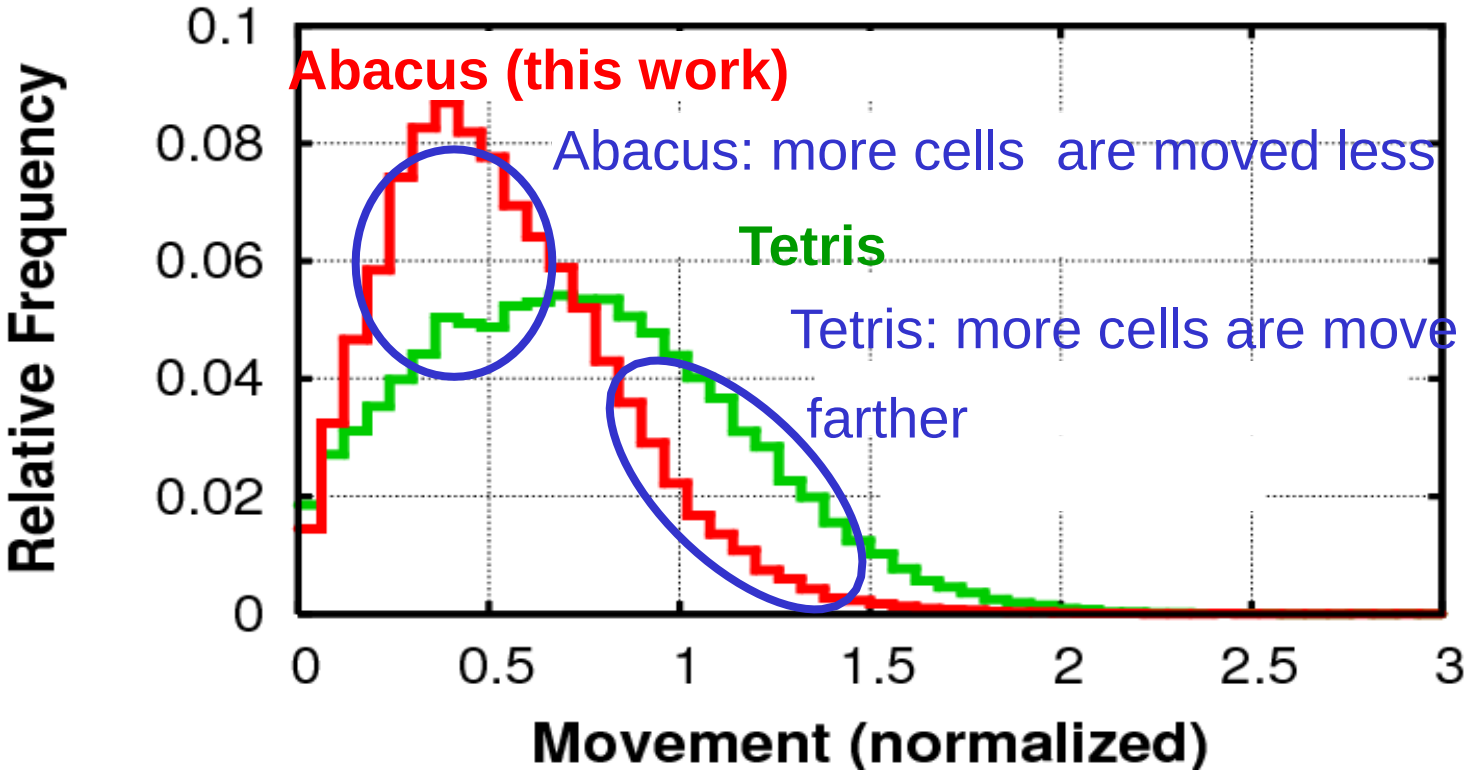
Tetris vs. Abacus

- Abacus can get shorter total movement and shorter maximum movement than Tetris
- Abacus can avoid overflow in each row
- The runtime of Abacus is longer than Tetris, but both can complete in few seconds for large designs



Comparison on Movement

Experimental results of one circuit:



→ Lower movement with Abacus

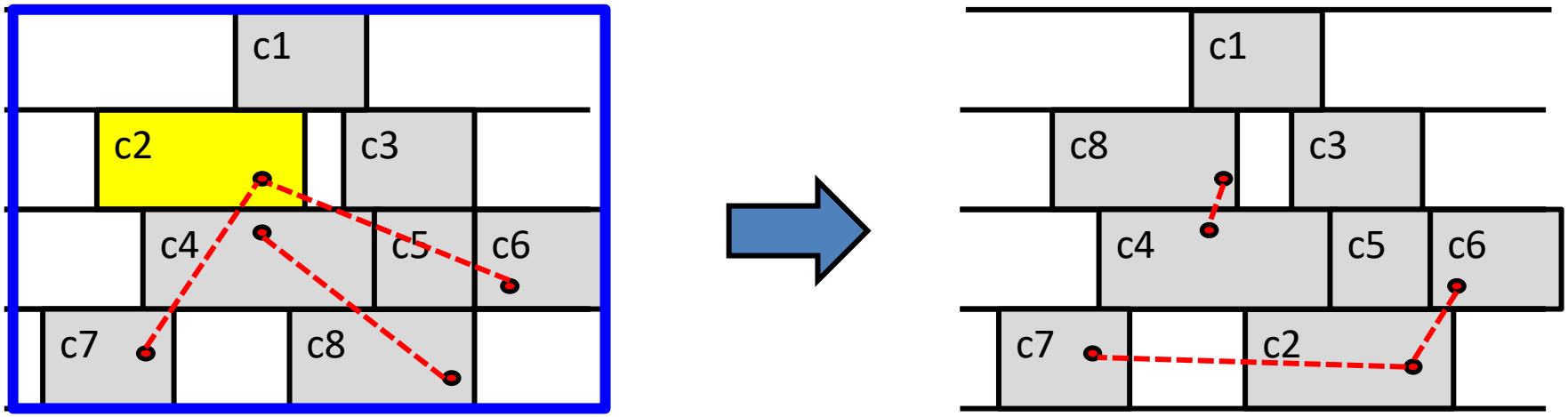
Detailed Placement

- Given a legalized placement solution, detailed placement further improves the wirelength (or other objectives) by locally rearranging cells while maintaining legality.
- Room for wirelength improvement?
  - A global placement algorithm typically uses an inaccurate wirelength model
  - A global placement algorithm might place a cell in a subregion without paying attention to the location of the cell within the subregion
  - During legalization, wirelength is likely to be worsened

Detailed Placement Techniques (1/5)

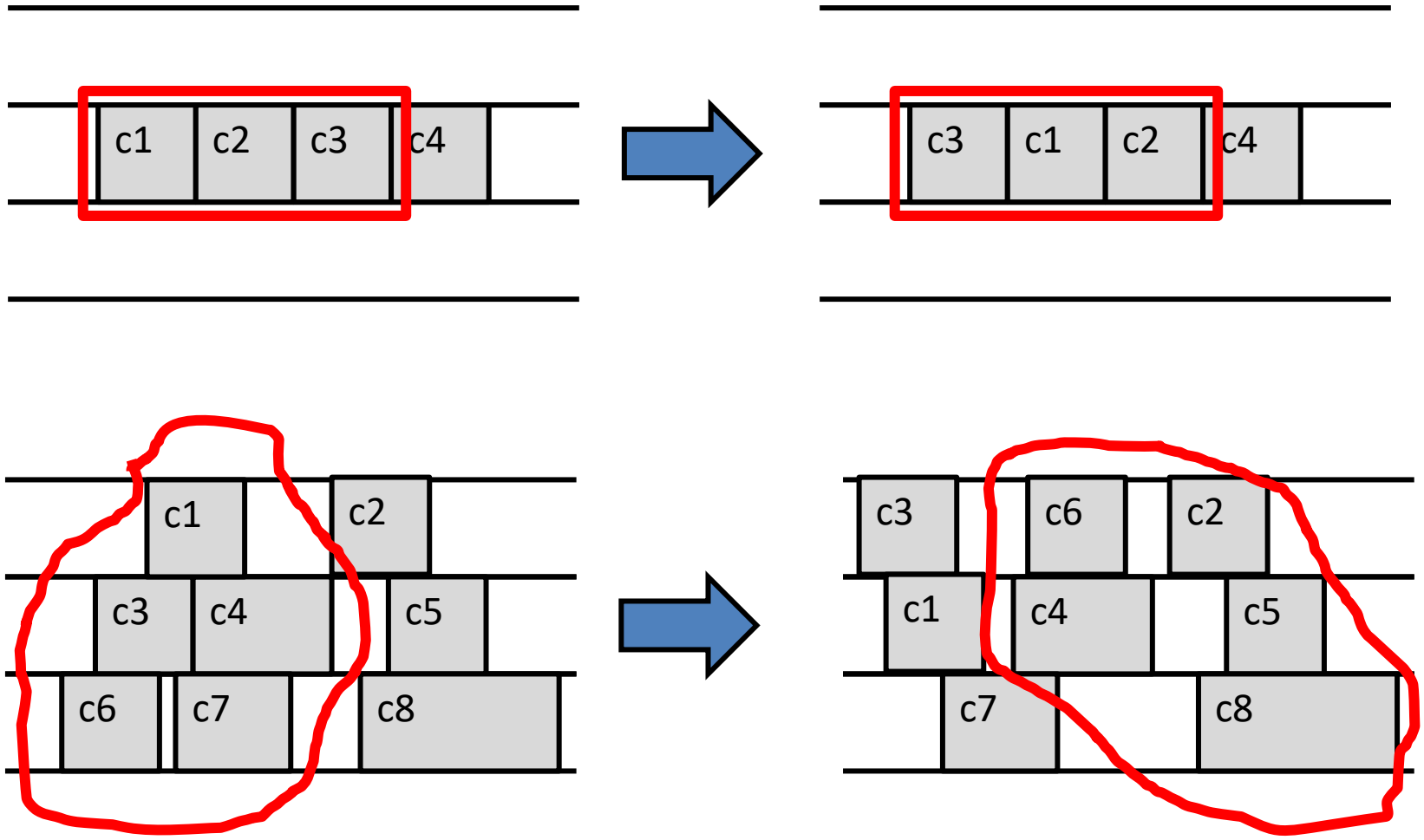
- Moving cells to minimize total HPWL without cell overlapping

Cell Swapping



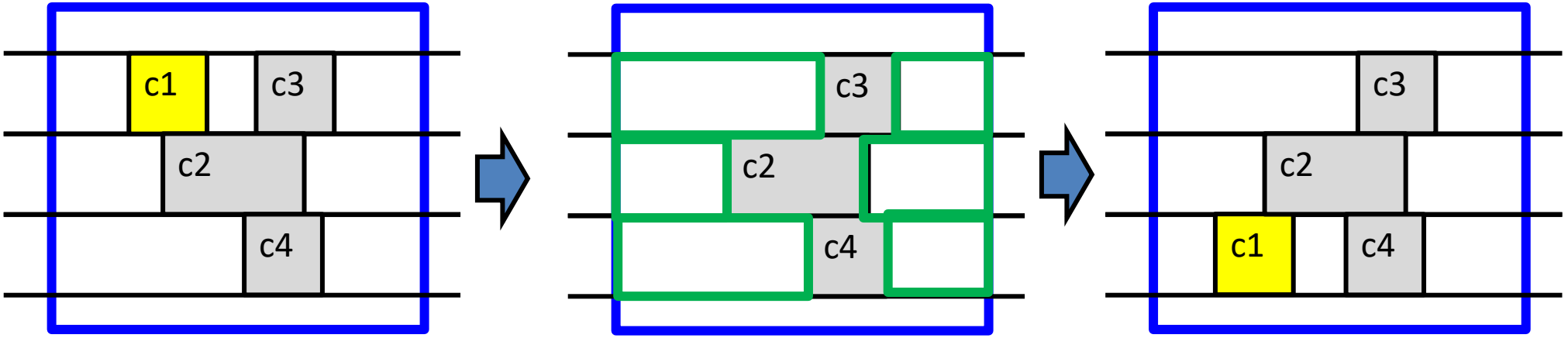
Detailed Placement Techniques (2/5)

Sliding Windows (followed by Branch and Bound or Network Flow)

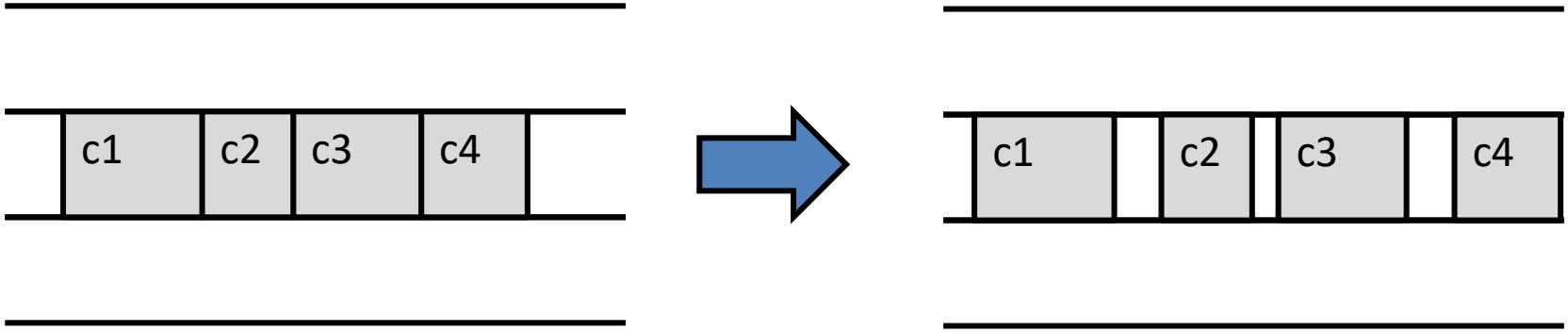


Detailed Placement Techniques (3/5)

Moving Cells to Empty Spots

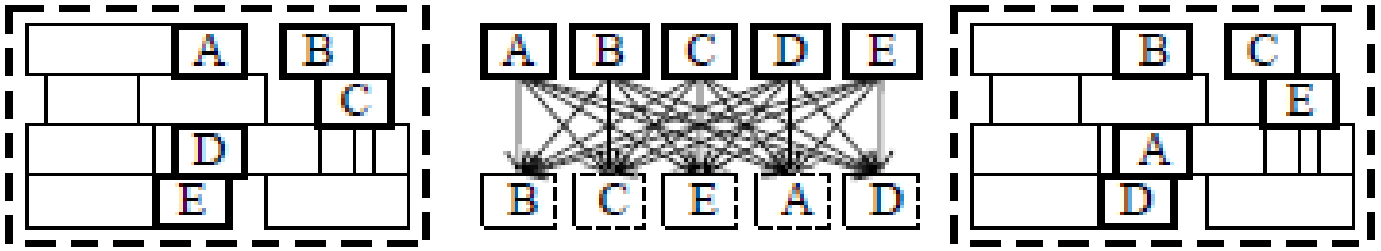


Dynamic-Programming-based Single-Row Placement



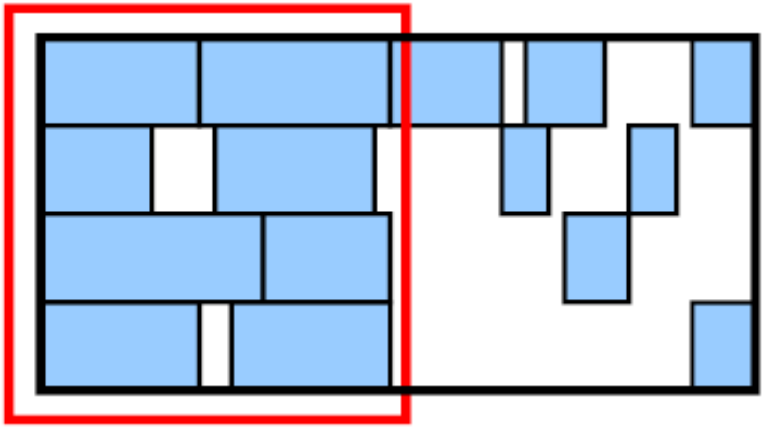
Detailed Placement Techniques (4/5)

- Cell matching
  - Select a window
  - Select **independent** cells (i.e., no common net between any pair of cells) from the window
  - Create a bipartite matching problem (edge weight = wirelength)
  - Find the minimum weighted matching to optimize the wirelengt
  - Update cell positions



Detailed Placement Techniques (5/5)

- Cell sliding: density optimization
- Steps
  - Select an overflow window
  - Calculate the amount of area to shift
  - Move cells left/right to reduce the density

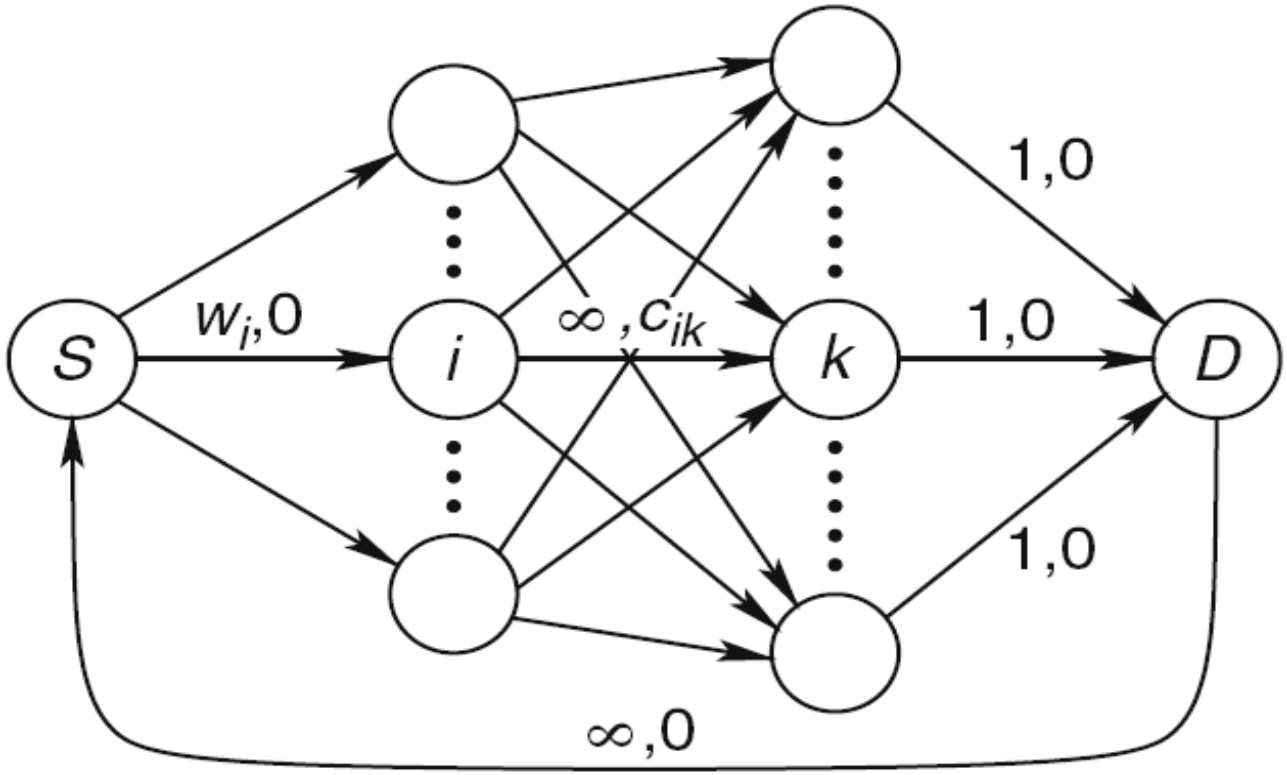


Detailed Placement: Domino [IEEE TCAD-94]

- Use a sliding window approach to iteratively refine a small region
- The problem of cell assignment is formulated as a transportation problem
- Each cell is divided into unit-width subcells
- The problem of transporting the subcells to the placement slots that minimizes WL is transformed into a **minimum cost maximum flow** problem

Min Cost Max Flow Problem

- c_{ik} models the total HPWL of nets connected to cell i if a subcell of i is assigned to location k
  - Estimated by analyzing different cases



Cost Calculation

- e: a net connected to cell i
- e_i: the set of cells connected to net e inside the region.
- Three cases
  - $|e_i| = 1$: The only cell inside the region that net e connects to is cell i. The HPWL of net e can be calculated exactly.
  - $1 < |e_i| < |e|$: Net e connects cells other than cell i both inside and outside the region. The unknown locations of cells in $e_i - \{i\}$ are estimated by their coordinates in the current placement. Then the HPWL of net e can be calculated.
  - $|e_i| = |e|$: Net e connects only to cells inside the region. The locations of cells in $e_i - \{i\}$ are estimated by their coordinates in the current placement. Besides, a virtual cell is introduced at the center of gravity of the cells in e with respect to the current placement. The HPWL of all cells in e together with the virtual cell is calculated and used.

Cell Arrangement

- After solving the network flow problem, subcells are assigned to locations.
- For all subcells of each cell, as they are associated with the same transportation cost and are pulled towards the cheapest location, they tend to lie side by side.
- Each cell is placed in the row holding most of its subcells, and the x-coordinate of the cell is determined by the center of gravity of its subcells.
- The cells in each row of the region are packed according to their x-coordinates to prevent overlap.

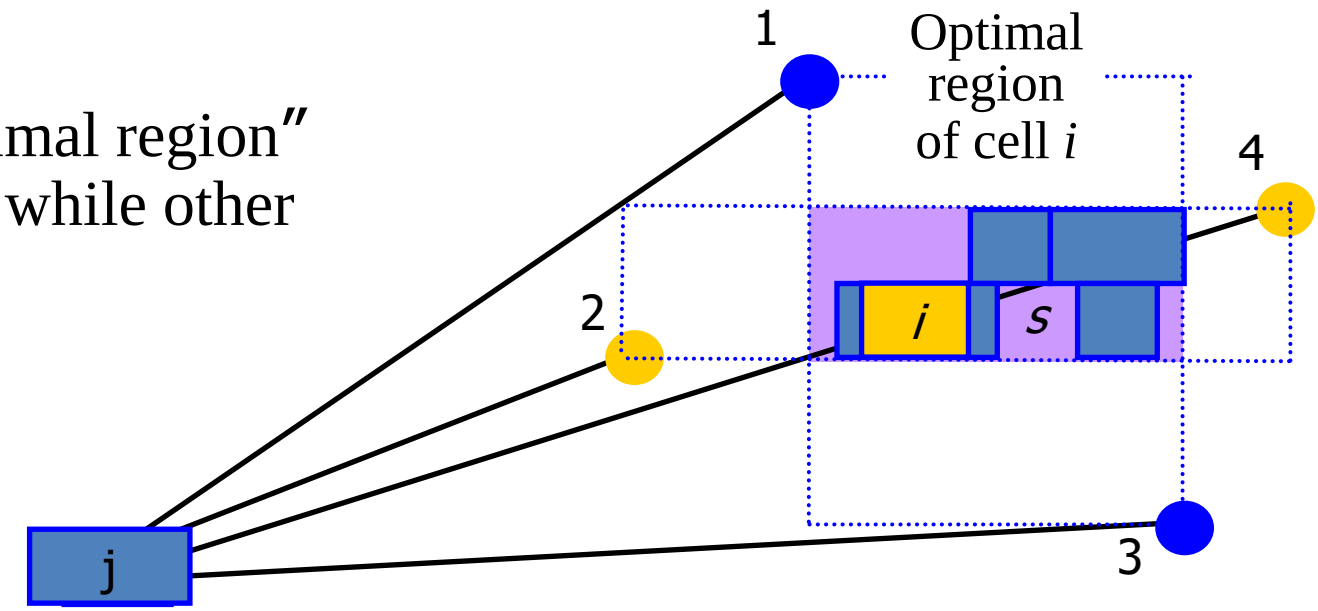
Detailed Placement: FastDP [ICCAD-05]

- A very fast and high-quality greedy heuristic

1. Perform single-segment clustering
2. Repeat
3. Perform global swap
4. Perform vertical swap
5. Perform local reordering
6. Until no significant improvement in wirelength
7. Repeat
8. Perform single-segment clustering
9. Until no significant improvement in wirelength

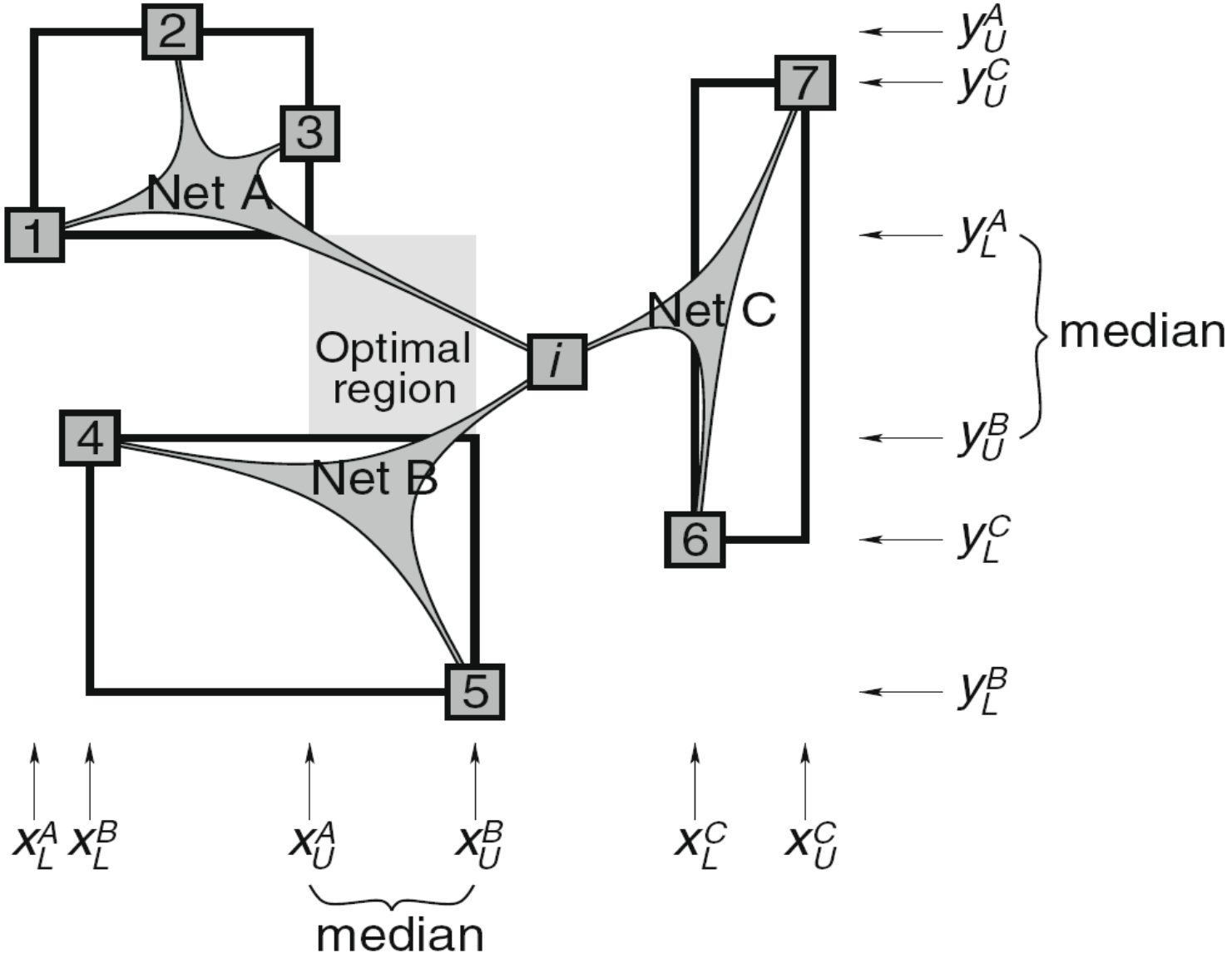
Global Swap

- **Main idea:**
  Move a cell to its "optimal region" (HPWL is minimized) while other cells are fixed



- **Major steps:**
  - For each standard cell *i*, find its optimal region
  - For every candidate cell *j* in the optimal region of cell *i*, *c*ompute the benefit to swap *i* with *j*
  - For every candidate space *s* in the optimal region of cell *i*, *c*ompute the benefit to swap *i* with *s*
  - Pick the cell or space with best positive benefit to perform swap

Optimal Region of a Cell [IEEE CAS-81]



Other Techniques in FastDP

- **Vertical swap:**
  - Move a cell one row up or down toward its optimal region to reduce the wirelength.
- **Local reordering:**
  - To fix local errors by reordering 3 adjacent cells
- **Single segment clustering:**
  - Optimally shift cells in a segment (i.e., a maximal unbroken section of a row) to minimize HPWL in linear time

Placement Benchmarks

- Benchmark suites from placement contests of ISPD 2005, ISPD 2006, ISPD 2011, DAC 2012, ICCAD 2012-2015, ISPD 2014-2015
  - Derived from IBM ASIC designs
  - Mixed-size designs with both fixed and movable modules
  - Up to a few million movable modules
  - A multi-thread global router is released
  - Detailed routability driven placement benchmark with routing information
- IBM-MSwPins
  - Derived from the same circuits as IBM-PLACE 2.0
  - Contain large movable macros and fixed pads
- IBM-PLACE 2.0
  - Derived from smaller IBM circuits released in 1998
  - Has standard cells only and no connection to I/O pads
- Faraday Mixed-sized benchmarks
  - Have sufficient routing information to run industrial routers
- PEKO benchmark suites
  - Synthetic placement benchmarks with known optimal wirelength
  - Many variations

Survey Papers on Placement

- Y.-W. Chang, Z.-W. Jiang, and T.-C. Chen, “Essential Issues in Analytical Placement Algorithms”, IPSJ Trans. on Systems LSI Design Methodology, 2009.

- I. Markov, J. Hu, and M.-C. Kim, “Progress and Challenges in VLSI Placement Research”, Proceedings of the IEEE, 2015.

Summary (1/10)

- Strengths (+) and weaknesses (-) for three types of placers [IPSJ 2009]

Simulated Annealing Placement
(+) Easier to consider multiple objectives simultaneously
(+) Good quality for small designs
(−) Harder to handle modules of very different sizes
(−) Slower and less scalable for large circuits
Min-Cut Placement
(+) More efficient and scalable, even for large circuits
(+) Good at mixed-size circuit legalization
(−) Harder to handle multiple objectives simultaneously
(−) Harder for whitespace management, especially for designs with low utilization rates
Analytical Placement
(+) More efficient and scalable, even for large circuits
(+) Better quality for large-scale designs
(+) Good at whitespace management, regardless of utilization rates
(+) Easier to handle multiple objectives simultaneously
(−) Harder to legalize large macros
(−) Harder to optimize macro orientations

Summary (2/10)

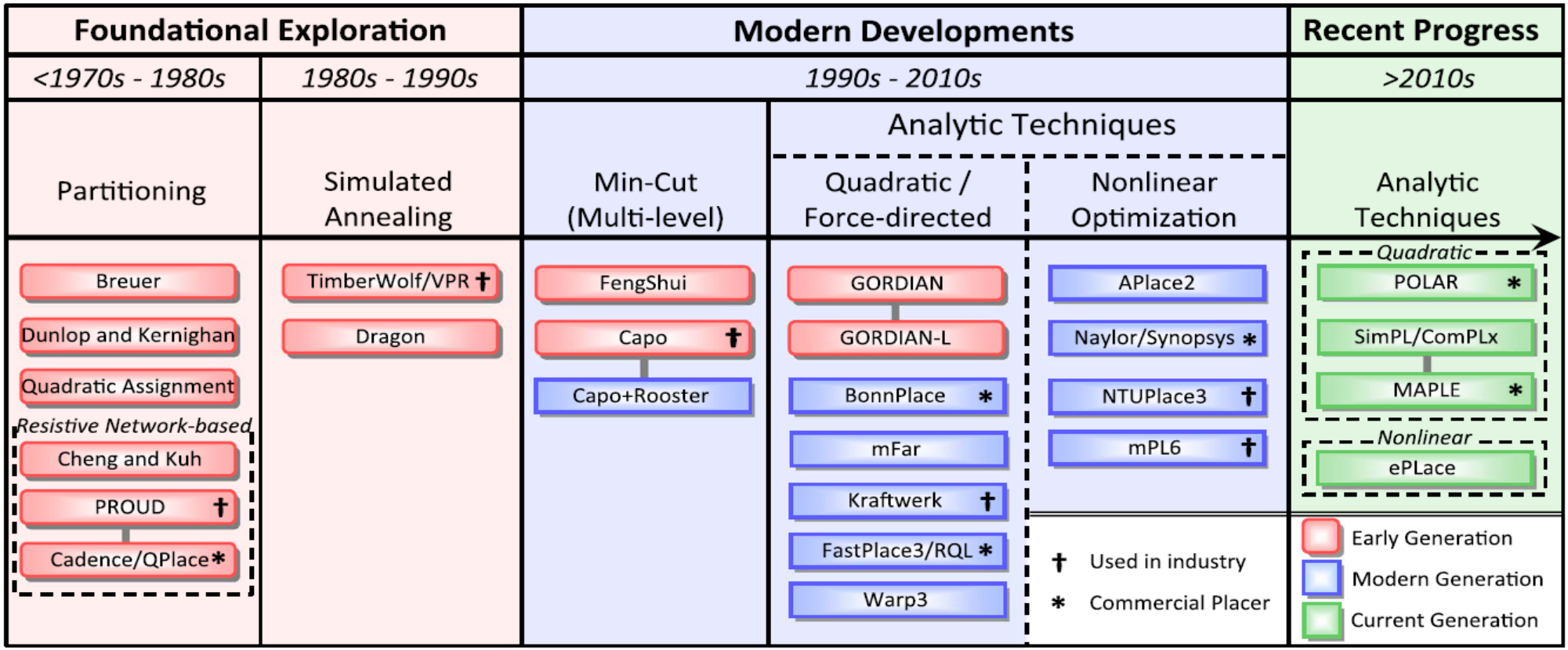
- Comparisons among recent academic analytical placers

Placer	Wirelength Model	Overlap Reduction	Integration	Optimization
APlace	LSE	Bin Density	Penalty Method	Nonlinear
BonnPlace	Quadratic	Partitioning	Region Constraint	Quadratic
DPlace	Quadratic	Diffusion	Fixed Point	Quadratic
ePlace	WA	Density (electrostatics)	Penalty Method	Nonlinear (Nesterov)
FastPlace	Quadratic	Cell Shifting	Fixed Point	Quadratic
FDP	Quadratic	Density	Fixed Point	Quadratic
Gordian	Quadratic	Partitioning	Region Constraint	Quadratic
hATP	Quadratic	Partitioning	Region Constraint	Quadratic
Kraftwerk2	Bound2Bound	Bin Density	Fixed Point	Quadratic
mFAR	Quadratic	Density	Fixed Point	Quadratic
mPL6	LSE	Bin Density	Penalty Method	Nonlinear
NTUplace3/4	WA/LSE	Bin Density	Penalty Method	Nonlinear (gradient)
Ripple	Bound2Bound	Partitioning	Penalty Method	Quadratic
RQL	Quadratic	Cell Shifting	Fixed Point	Quadratic
SimPL	Bound2Bound	Partitioning	Penalty Method	Quadratic
Vassu	LSE	Assignment	Fixed Point	Nonlinear

Source: "Modern Circuit Placement" by Y.-W. Chang at 2015 VLSI Design/CAD Symp.

Summary (3/10)

- Historical development of global placement algorithms and implementations in academia [IEEE 2015]



Summary (4/10)

- Legalization and detailed placement [IEEE 2015]

STAGE	TECHNIQUE
LEGALIZATION	greedy moves to free locations [39], [73], [74], [180], [197]
	ripple cell movement [87]
	diffusion PDE [161]
	dynamic programming [4], [96], [180]
	computational geometry [132]
	network flow [12], [15], [44], [58], [59]
	linear programming [55]
	top-down opt. & clustering [74], [116]
DETAILED PLACEMENT	branch-and-bound [18], [24]
	network flow [58]
	simulated annealing [172]
	mixed ILP [25], [120]
	single-row optimization [14], [98]
	cell-to-slot matching [39]
	cell swapping [55], [149]
	clustering [87], [149]
	dynamic programming [85]
	global-placer integration [166]

Summary (5/10)

- Mixed-size placement [IEEE 2015]

FLOW	TECHNIQUE
SIMULTANEOUS	macro shredding [17], [106], [164]
	macro or cell shifting [39], [107], [197]
	iterative re-legalization [27], [56]
	top-down legalization [27], [55], [164], [167]
	force-directed / non-convex optimization [27], [39], [62], [76], [99], [106], [107], [195], [197]
	floorplacement [45], [164], [167]
SEQUENTIAL	periphery macro packing [38]
	macro shredding [2]
	separate floorplanning & placement steps [2, Flow 1], [35], [38], [216]
	simulated annealing [125], [135]
POST-PROCESS	floorplan repair [139]
	linear programming [17], [55], [181]
	force-directed optimization [2, Flow 2]

Summary (6/10)

- Congestion estimation for placement [IEEE 2015]

APPROACH	TECHNIQUE
STATIC	net bounding box [22], [91]
	Steiner trees [165]
	pin density [13], [223]
	counting nets in regions [203]
PROBABILISTIC	L-shaped routes [71], [72], [78], [179]
	smoothened wire density [189]
	pattern routing [209]
CONSTRUCTIVE	using A*-search [210]
	using a global router: • FastRoute [214] in IPR [49] and POLAR [123] • BFG-R [83] in SimPLR [104], CoPR [84] • NCTUgr [128] in Ripple 2.0 [72]

Summary (7/10)

- Routability-driven placement [IEEE 2015]

PLACEMENT PHASE	TECHNIQUE
GLOBAL PLACEMENT	relocating movable objects: • moving nets [71], [91] • modifying forces [51], [137], [179] • incorporating congestion in objective function [78], [189] • adjusting target density [104]
	cell bloating [13], [71], [72], [75], [84], [104], [127]
	macro porosity [78], [91]
	pin density control [78]
	expanding/shrinking placement regions [154]
INTERMEDIATE	local placement refinement [49]
LEGALIZATION AND DETAILED PLACEMENT	linear placement in small windows [90], [165]
	congestion embedded in objective function [222]
	cell swapping [49], [71], [104]
	cell shifting [57], [78]
POST PLACEMENT	whitespace injection or reallocation [119], [165], [217]
	simulated annealing [40], [79], [200]
	linear programming [122]
	network flows [201], [202]
	shifting modules by expanding gcells [222]
	cell bloating [169]
	cost-based cell relocation [129]

Summary (8/10)

- Timing-driven placement [IEEE 2015]

TECHNIQUE	IMPLEMENTATION
STATIC NET WEIGHTS	slack [20], [28], [61], [111]
	sensitivity [66], [163], [213]
DYNAMIC NET WEIGHTS	incremental timing analysis [20], [160]
	based on previous iterations [62], [160]
NET CONSTRAINTS	in global placement [64], [67], [159], [185], [188]
	in detailed placement [45], [68], [86], [162]
PATH-BASED	partitioning [89]
	simulated annealing [183]
	Lagrangian relaxation [69], [182]
	differential timing analysis [48]
	local movement transforms [33], [136]
	force-directed [138]
COMPOUND	net weights & constraints [130]

Summary (9/10)

- Power-driven placement

POWER TYPE	TECHNIQUE
STATIC	multiple supply voltages [82], [113], [124], [157], [218]
	logic and physical adjacency [158]
DYNAMIC	weights on signal nets [42], [143], [171], [190]
	register clustering [32], [42], [92], [176]
	explicit register relocation [43], [133], [153]
	clock-tree co-synthesis [50], [115], [153], [176], [204]

Summary (10/10)

- Placement-aware physical synthesis [IEEE 2015]

TECHNIQUE	IMPLEMENTATION
LOGIC TRANSFORMATIONS	fanin restructuring [208], [212]
	cloning and decomposition [60]
	simulation-driven restructuring based on don't-cares [155]
INTERCONNECT BUFFERING	fanout restructuring [60]
	delay-optimal [152], [174]
	Steiner tree construction [5]
GATE SIZING	discrete [81], [147], [151]
	continuous [184], [198]
COMPOUND TRANSFORMATIONS	area management [118]
	retiming-based physical synth. [150]
	design flows [151], [153], [186], [198]