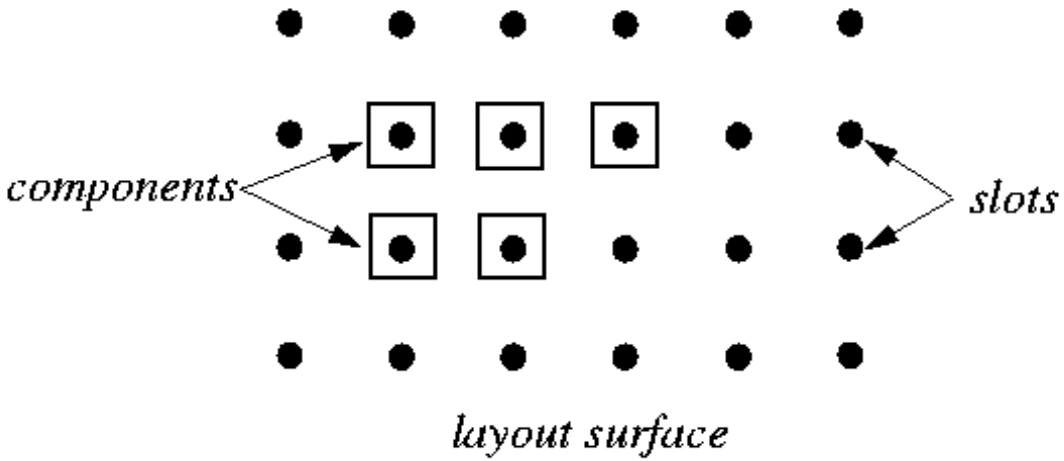
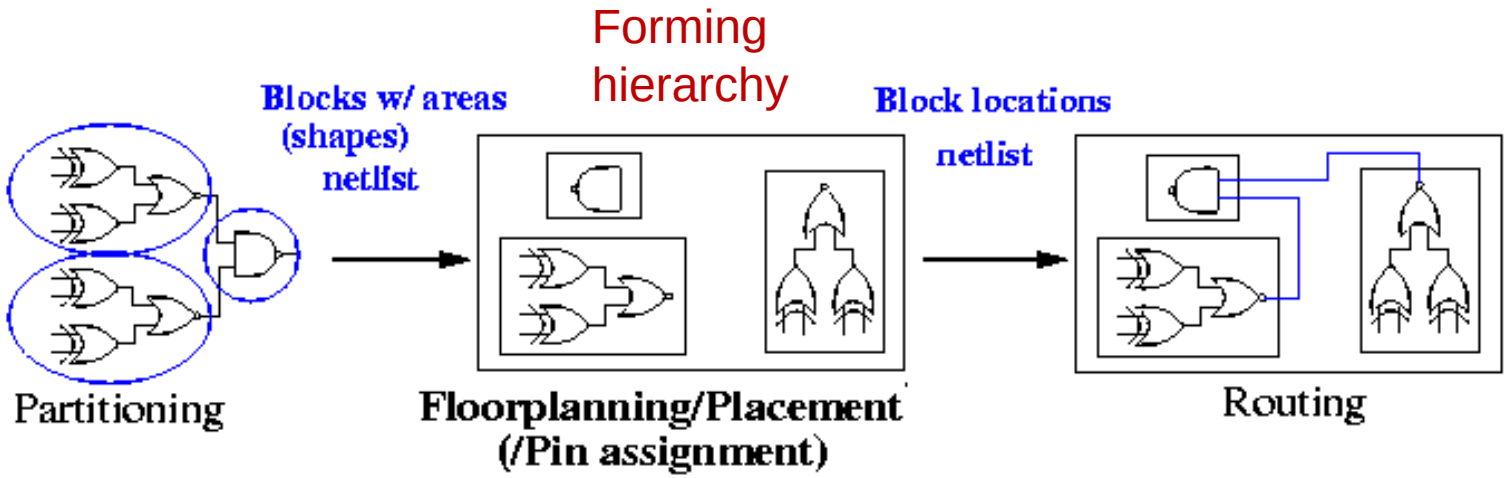
# Unit 5C: Placement

- Course contents:
  - Placement metrics
  - Constructive placement: cluster growth, min cut
  - Iterative placement: force-directed method, simulated annealing, genetic algorithm
- Readings
  - Chapter 7.1--7.4
  - Chapter 5.8 (genetic algorithm)



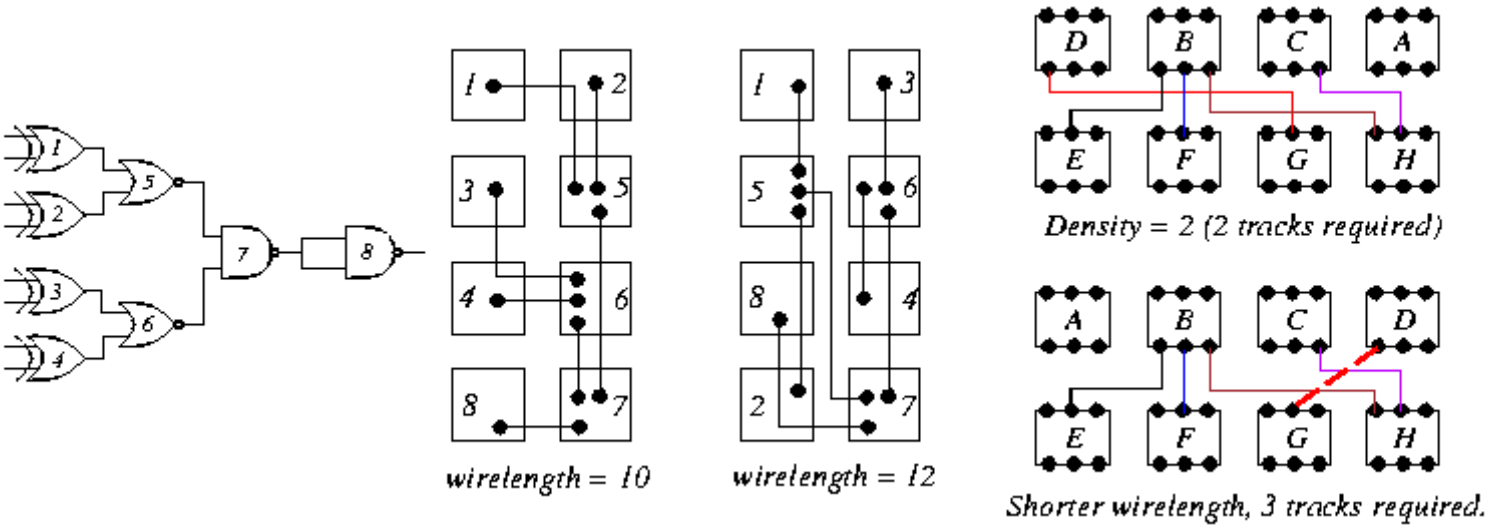
# Placement

- **Placement** is the problem of automatically assigning correct positions on the chip to predesigned cells, such that some cost function is optimized.
- Inputs: A set of **fixed-size** cells/modules, a netlist.
- Goal: Find the best position for each cell/module on the chip according to appropriate cost functions.
  - Considerations: **routability/congestion, channel density**, **wirelength**, cut size, performance, thermal issues, I/O pads.



# Placement Objectives and Constraints

- What does a placement algorithm try to optimize?
  - the total area
  - the total wire length
  - the number of horizontal/vertical wire segments crossing a cut line
- Constraints:
  - The placement should be routable (no cell overlaps; no density overflow)
  - Timing constraints are met (some wires should be shorter than a given length).

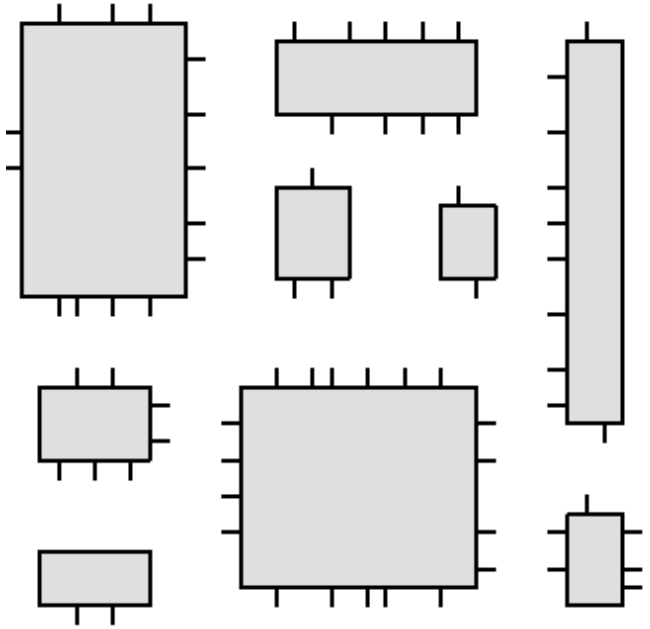


# VLSI Placement: Building Blocks

- Different design styles create different placement problems.
  - E.g., building-block, standard-cell, gate-array placement

- Building block: The cells to be placed have arbitrary shapes

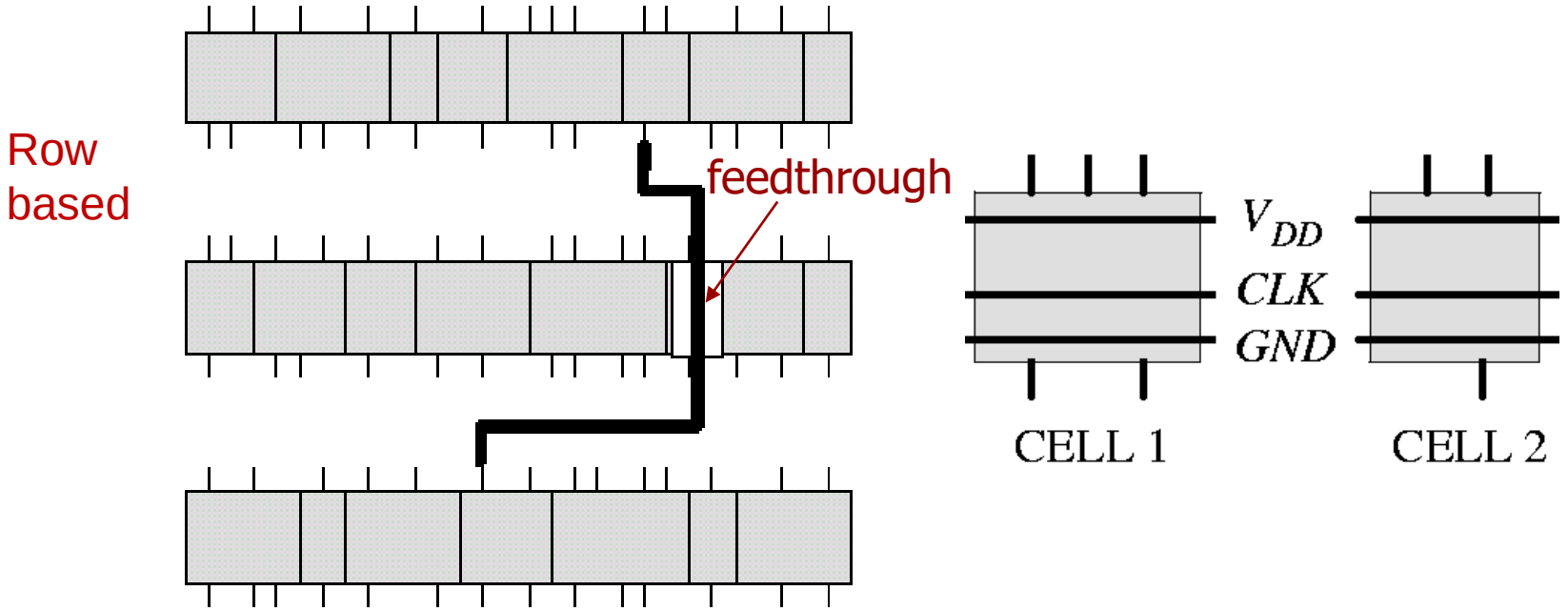
Mixed-size macros (in reality, we have many standard cells and various # of macros)



Building block

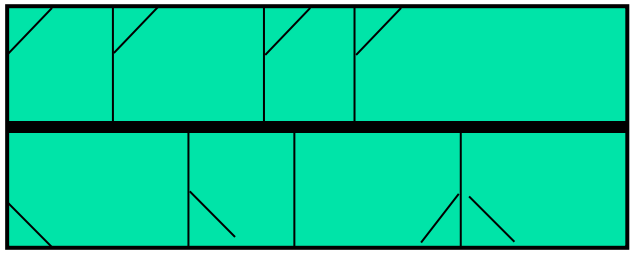
# VLSI Placement: Standard Cells – Row Placer

- Standard cells are designed in such a way that power and clock connections run horizontally through the cell and other I/O leaves the cell from the top or bottom sides.
- The cells are placed in rows.
- In old days feedthrough cells are added to ease wiring.



# More Styles

## Back-to-back

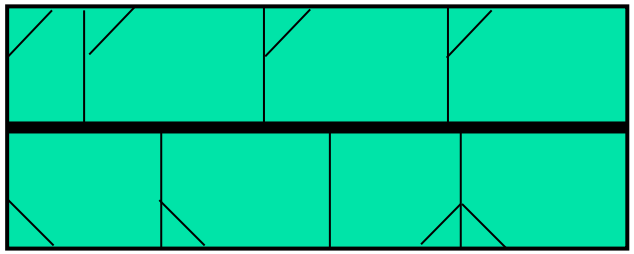


ground

power

ground

## Routing Channel

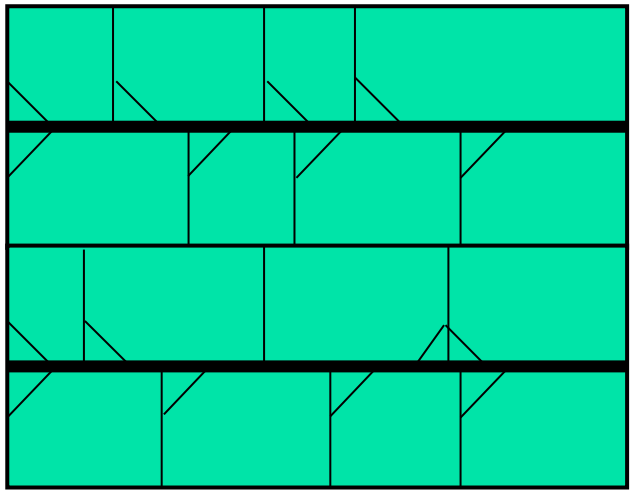


ground

power

ground

## Channel-less



power

ground

power

ground

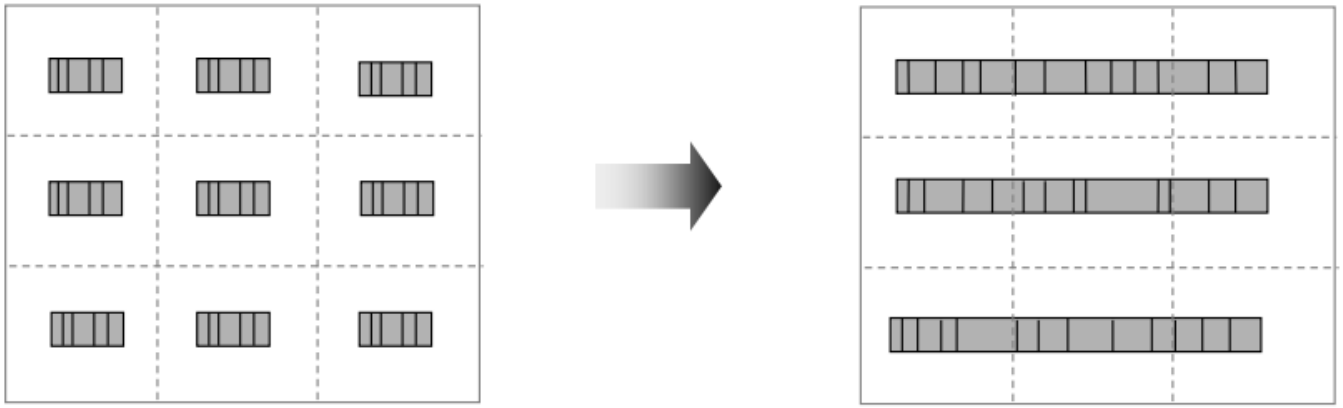
power

# Introduction

Global Placement

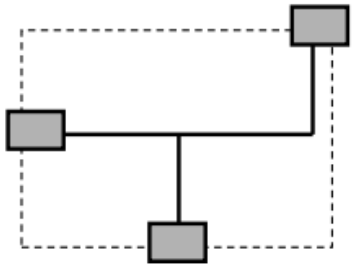
Detailed Placement

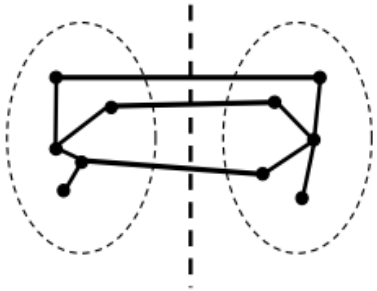
Legalize cell locations

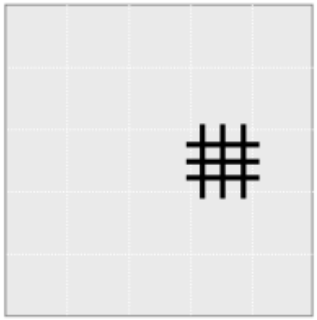


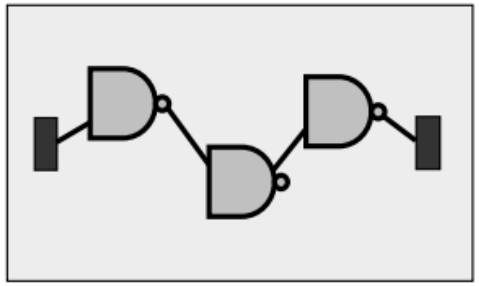
# Optimization Objectives

| Total Wirelength | Number of Cut Nets | Wire Congestion | Signal Delay |
|---|---|---|---|









# Optimization Objectives – Total Wirelength

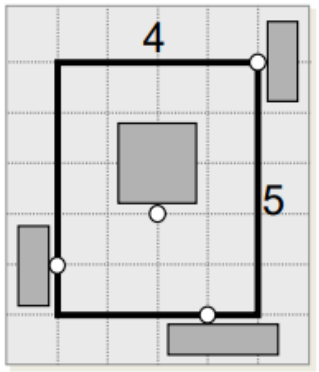
**Wirelength estimation for a given placement**

Half-perimeter wirelength (HPWL)

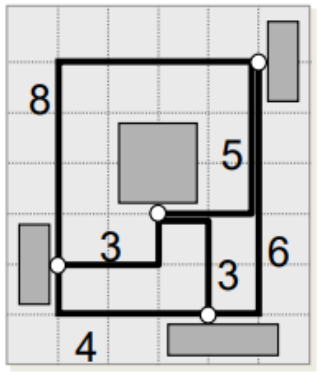
Complete graph (clique)

Monotone chain

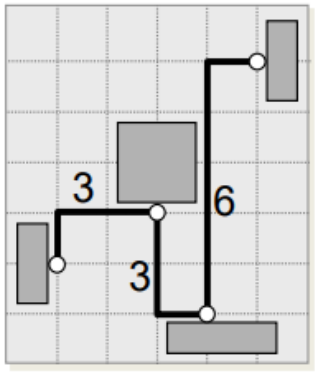
Star model



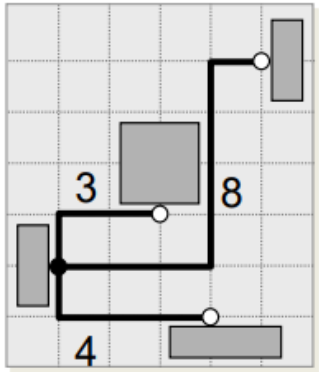

HPWL = 9




Clique Length = $(2/p)\Sigma_{e \in \text{clique}} d_M(e)$ = 14.5




Chain Length = 12




Star Length = 15

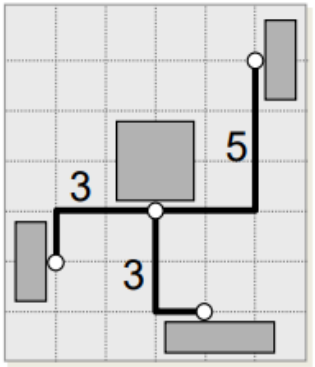
Sait, S. M., Youssef, H.: VLSI Physical Design Automation, World Scientific

# Optimization Objectives – Total Wirelength

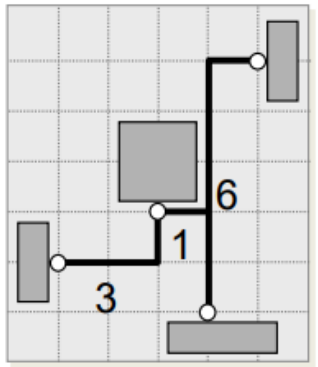
Custom routing like to use trunk style

**Wirelength estimation for a given placement (cont'd.)**

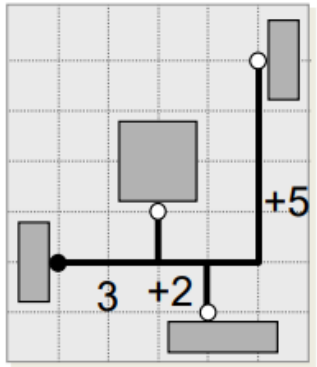
| Rectilinear minimum spanning tree (RMST) | Rectilinear Steiner minimum tree (RSMT) | Rectilinear Steiner arborescence model (RSA) | Single-trunk Steiner tree (STST) |
|---|---|---|---|



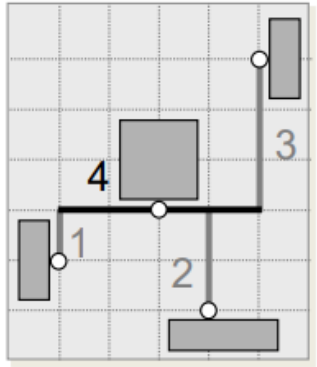

RMST Length = 11




RSMT Length = 10




RSA Length = 10




STST Length = 10

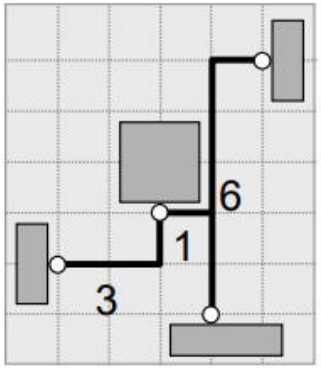
Sait, S. M., Youssef, H.: VLSI Physical Design Automation, World Scientific

# Optimization Objectives – Total Wirelength

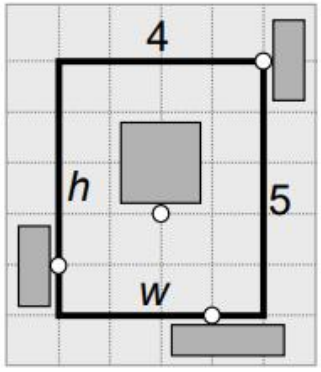
**Wirelength estimation for a given placement (cont'd.)**

Preferred method: Half-perimeter wirelength (HPWL)

- Fast (order of magnitude faster than RSMT)
- Equal to length of RSMT for 2- and 3-pin nets
- Margin of error for real circuits approx. 8% [Chu, ICCAD 04]




RSMT Length = 10

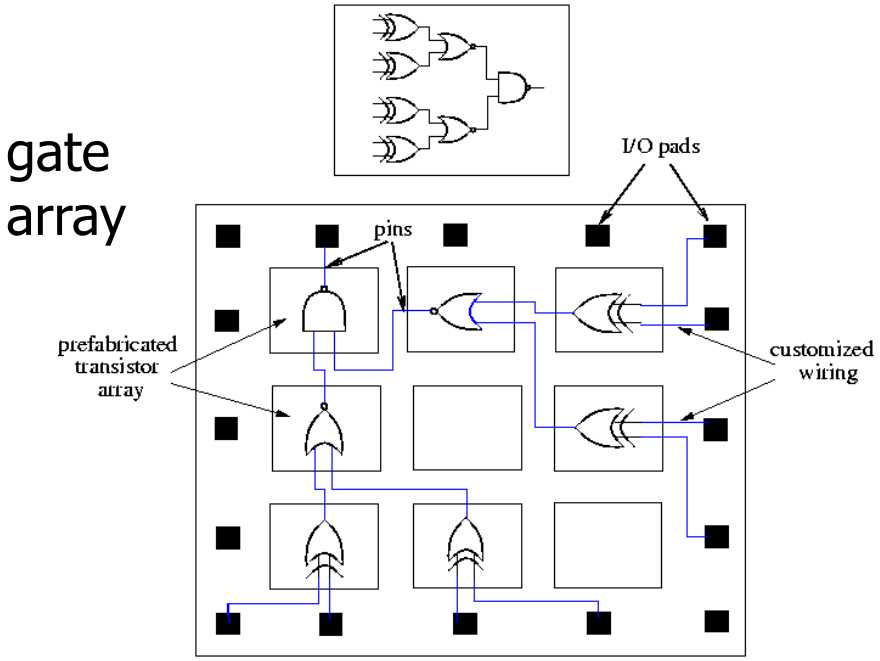



HPWL = 9

$$L_{\text{HPWL}} = w + h$$

# Consequences of Fabrication Method

- Full-custom fabrication (building block – mixed sizes):
  - Free selection of aspect ratio (quotient of height and width).
  - Height of wiring channels can be adapted to necessity.
- Semi-custom fabrication (gate array, standard cell):
  - Placement has to deal with fixed carrier dimensions.
  - Placement should be able to deal with fixed channel capacities.



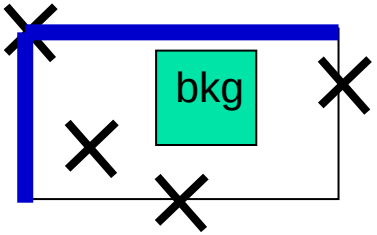
# Relation with Routing

- Ideally, placement and routing should be performed simultaneously as they depend on each other's results. This is, however, too complicated.
  - P&R: placement and routing

- In practice, placement is done prior to routing. The placement algorithm estimates the wire length of nets using some *metrics*.
  - Usually we do global placement first, followed by detailed placement

# Estimation of Wirelength

- **Semi-perimeter method:** Half the perimeter of the bounding rectangle that encloses all the pins of the net to be connected. Most widely used approximation!



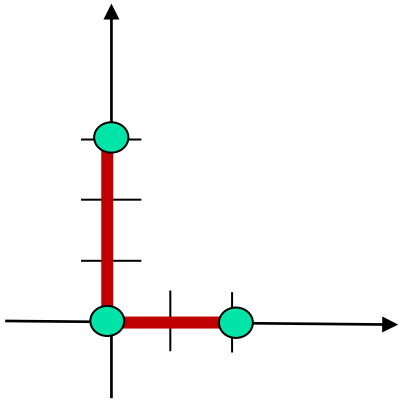

- **Squared Euclidean distance:** Squares of all pairwise terminal distances in a net using a quadratic cost function

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

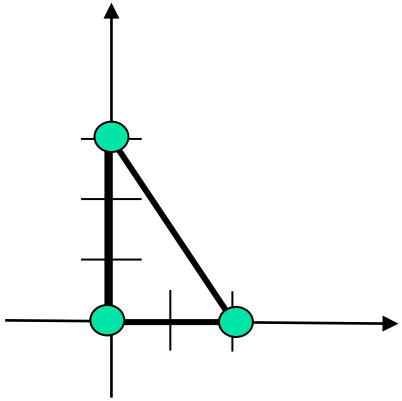
- **Steiner-tree approximation:** Computationally expensive.
- **Minimum spanning tree**: Good approximation to Steiner trees.
- **Complete graph:** Since #edges in a complete graph is $\left(\frac{n(n-1)}{2}\right)$ = $\frac{n}{2}$ × # of tree edges ($n$-1), *wirelength* ≈ $\frac{2}{n}\sum_{(i,\,j)\,\in\,net}$ *dist*($i$, $j$).

# An Example (For Complete Graph)

A net with 3 pins

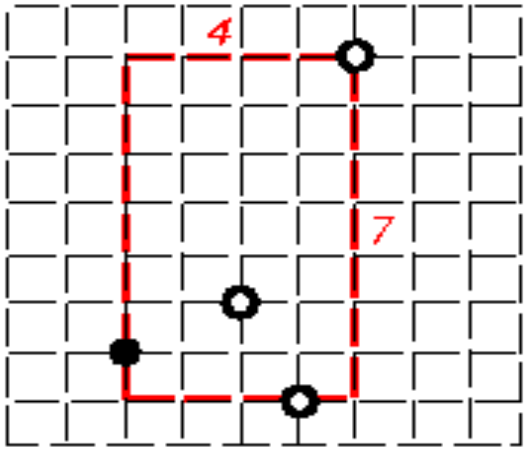


A short connection,
whose total length
is 2+3=5

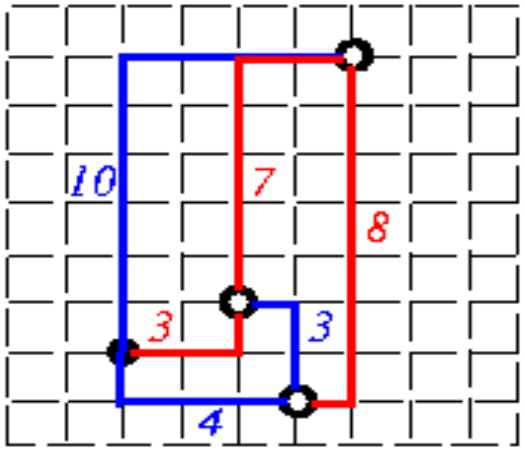


Using clique (complete graph)
to model this 3-pins net:
(2 + 3 + √13) * 2 / 3 ~ 5.7

# Estimation of Wirelength (cont'd)

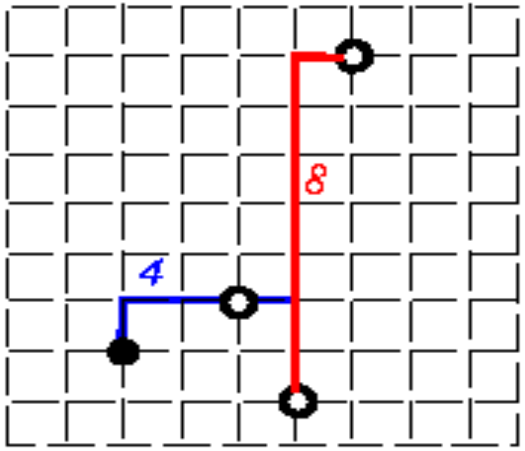



*semi–perimeter len = 11*

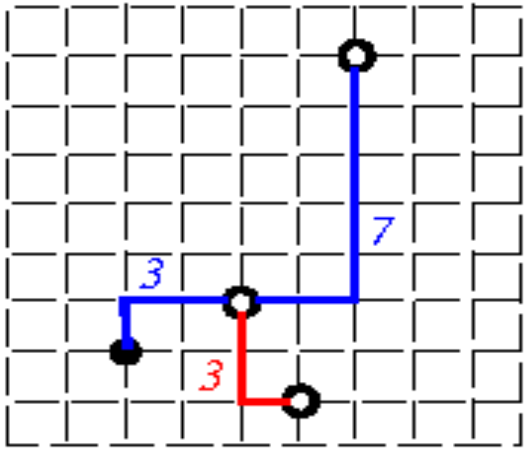



*complete graph len * 2/n = 17.5*

=10+7+8+3+3+4
=35




*Steiner tree len = 12*



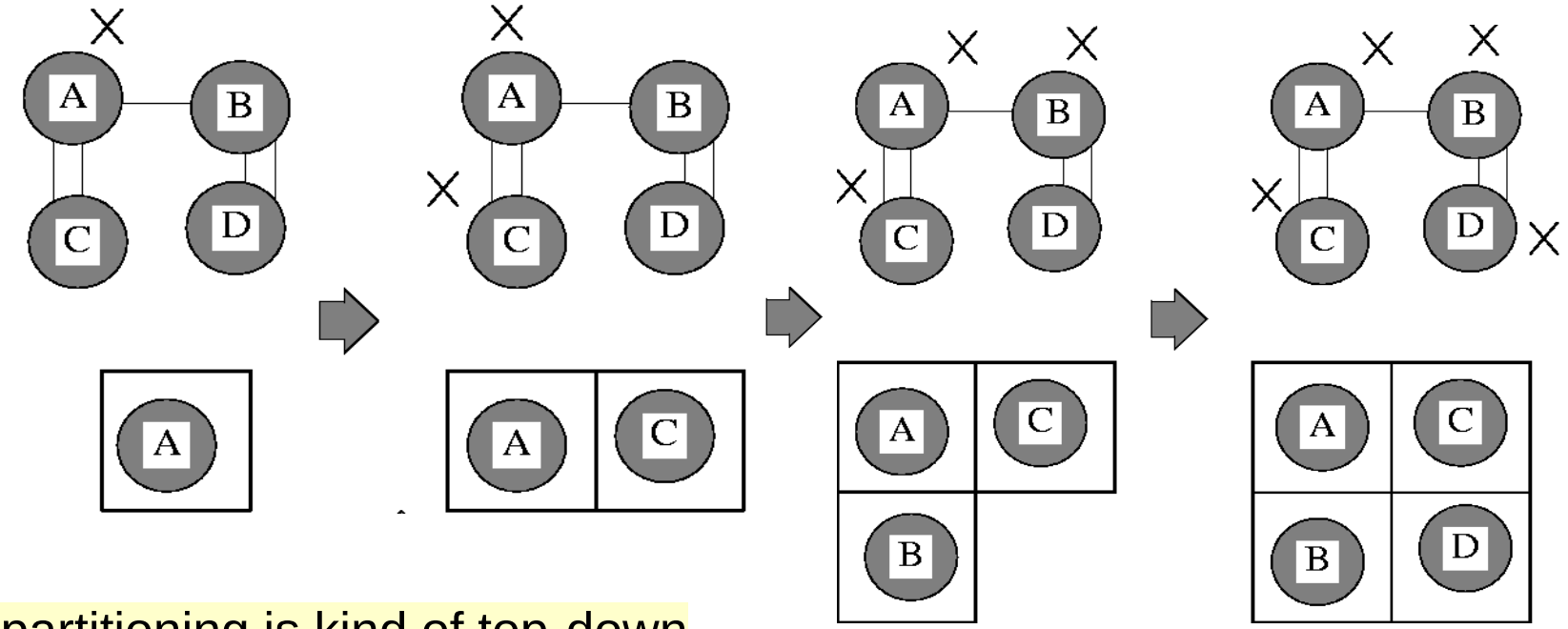

*Spunning tree len = 13*

# Placement Algorithms

- 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, min cut, 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.
  - **Analytical methods**: Gordian, Gordian-L, RePlace, …
- Most approaches combine multiple elements:
  - Constructive algorithms are used to obtain an initial placement.
  - The initial placement is followed by an iterative improvement phase.
  - The results can further be improved by simulated annealing.

# Bottom-Up Placement: Clustering

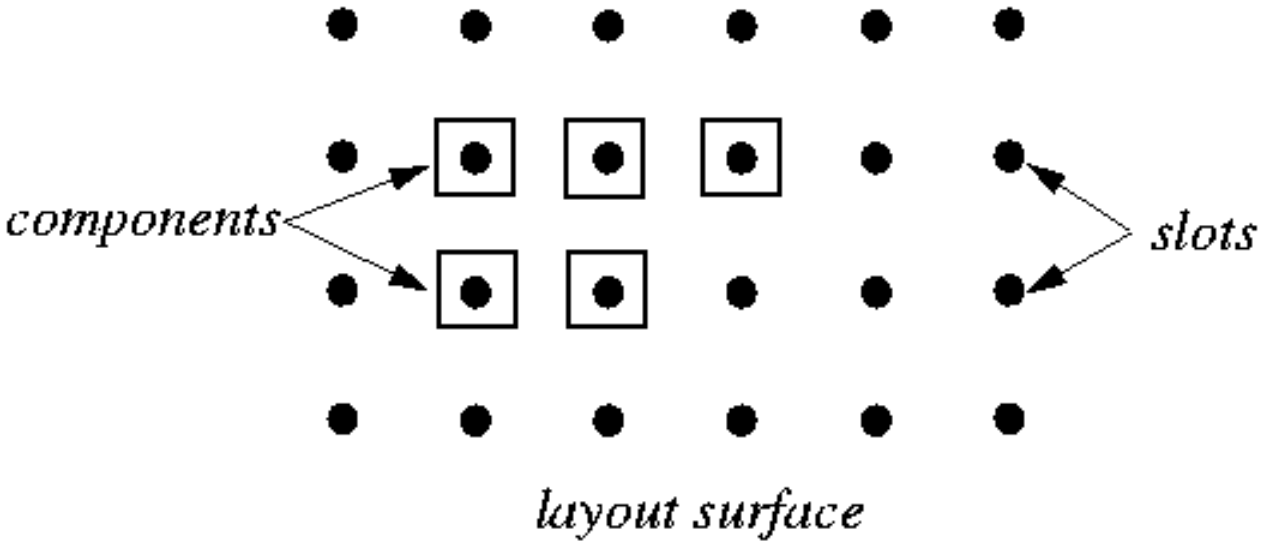
- Starts with a single cell and finds more cells that share nets with it.




K-L partitioning is kind of top-down.
Hence, how about combining the two?
Starting from bottom up to ensure those tightly connected cells are in the same partition

# Placement by Cluster Growth

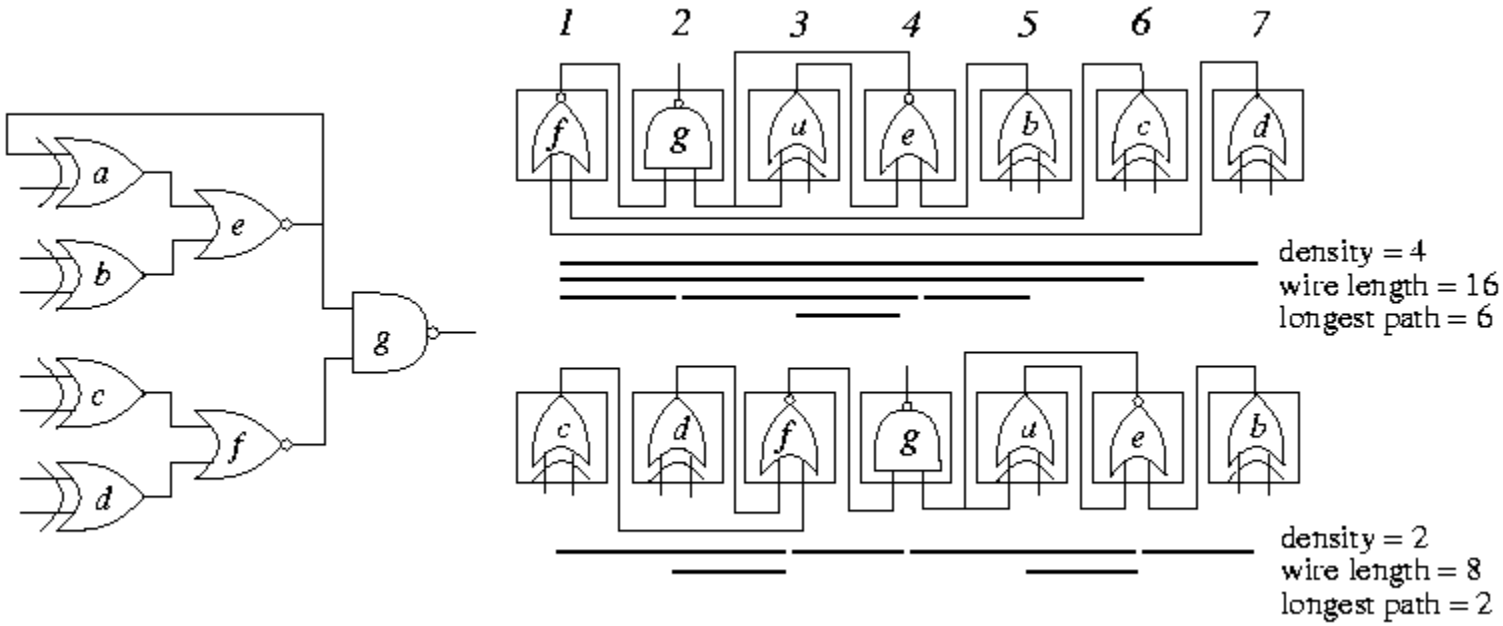
- Greedy method: Selects unplaced components and places them in available slots.
  - SELECT: Choose the unplaced component that is most strongly connected to all of the placed components (or most strongly connected to any single placed component).
  - PLACE: Place the selected component at a slot such that a certain "cost" of the partial placement is minimized.



# Cluster Growth Example – (Into One Row)

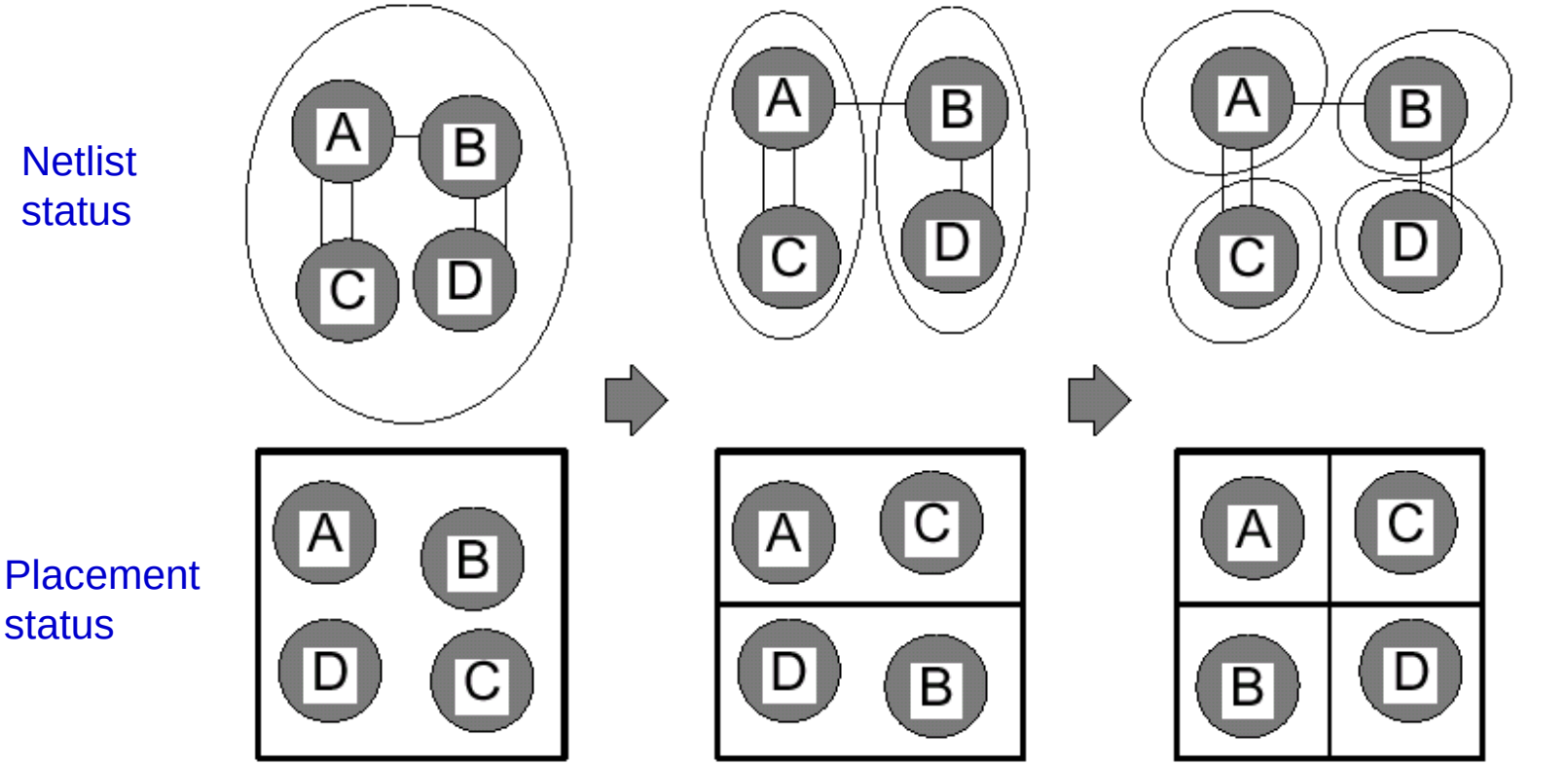
Tentatively ignore primary IOs (for e, it connects to input of a, input of g, output of a, output of b)

- # of other terminals connected: $c_a$=3, $c_b$=1, $c_c$=1, $c_d$=1, $c_e$=4, $c_f$=3, and $c_g$=3 ⇒ e has the most connectivity.
- Place *e* in the center, slot 4. *a, b, g* are connected to *e*, and $\hat{c}_{ae} = 2, \hat{c}_{be} = \hat{c}_{eg} = 1$ ⇒ Place *a* next to *e* (say, slot 3). Continue until all cells are placed.
  - Breaking ties is important, too
- Further improve the placement by swapping the gates.



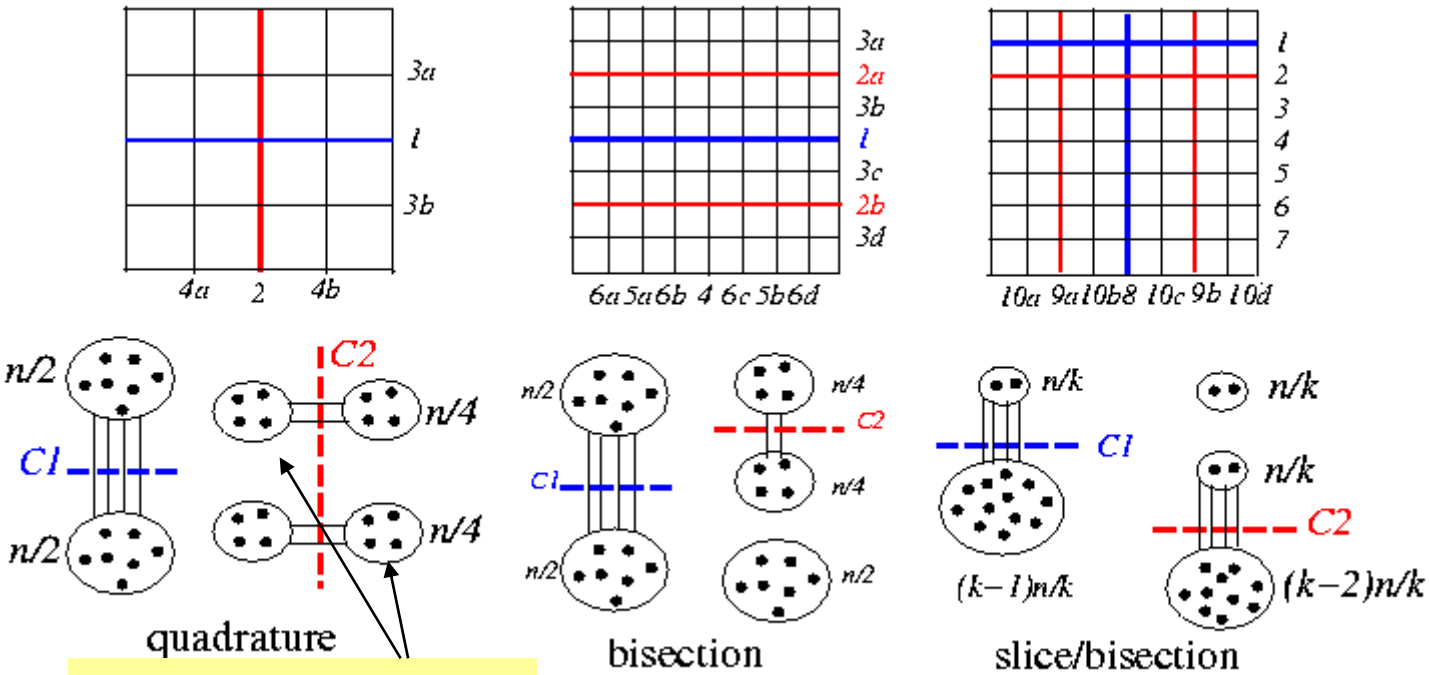
# Top-down Placement: Min Cut

- Starts with the whole circuit and ends with small circuits.
- Recursive bipartitioning of a circuit (e.g., K&L) leads to a min-cut placement.



# 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.




A Little bit misleading: Q2 has connections to Q4

# Algorithm for Min-Cut Placement

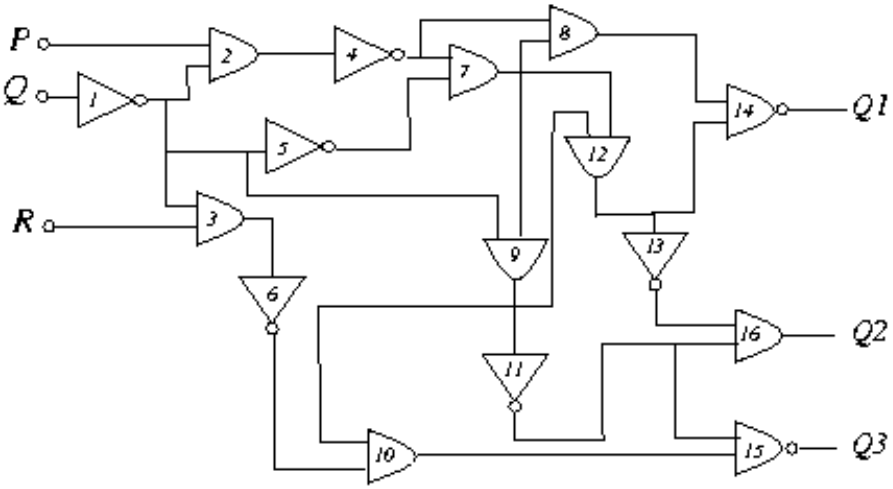
**Algorithm: Min_Cut_Placement(*N*, *n*, *C*)**

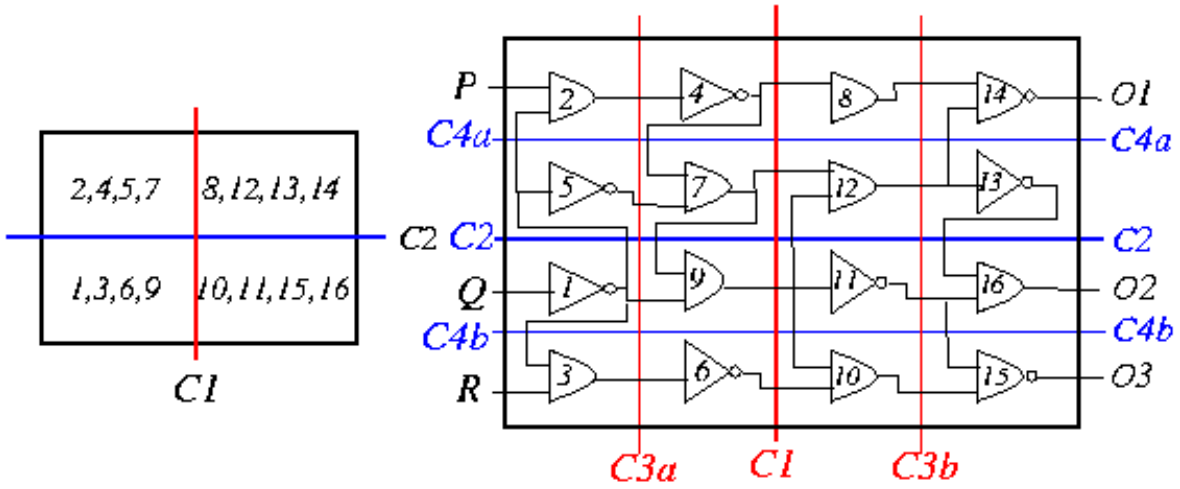
/* *N*: the layout surface */
/* *n* : # of cells to be placed */
/* $n_0$: # of cells in a slot */
/* *C*: the connectivity matrix */

1 **begin**
2 **if** ($n \leq n_0$) **then** PlaceCells(*N*, *n*, *C*)
3 **else**
4 $(N_1, N_2) \leftarrow$ CutSurface(*N*);
5 $(n_1, C_1), (n_2, C_2) \leftarrow$ Partition(*n*, *C*);
6 **Call** Min_Cut_Placement($N_1$, $n_1$, $C_1$);
7 **Call** Min_Cut_Placement($N_2$, $n_2$, $C_2$);
8 **end**

# Quadrature Placement Example

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



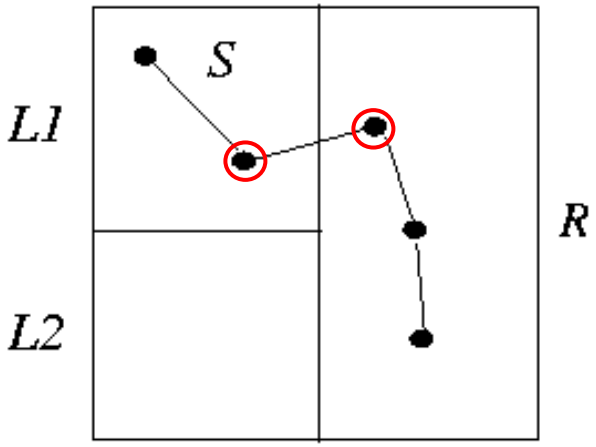



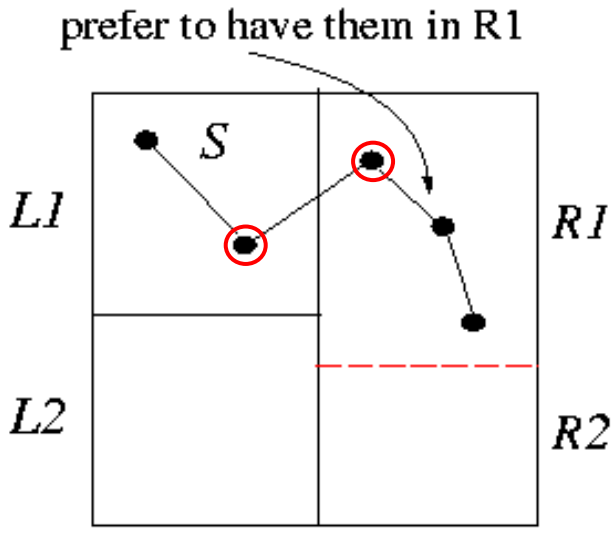
# General Procedure for Iterative Improvement

```
Algorithm: Iterative_Improvement()
1  begin
2  s ← initial_configuration();
3  c ← cost(s);
4  while (not stop()) do
5      s' ← perturb(s);
6      c' ← cost(s');
7      if (accept(c, c'))
8      then   s ← s';
9  end
```

# 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 terminal pins that enter a region.
  - What happens if we swap {1, 3, 6, 9} and {2, 4, 5, 7} in the previous example?



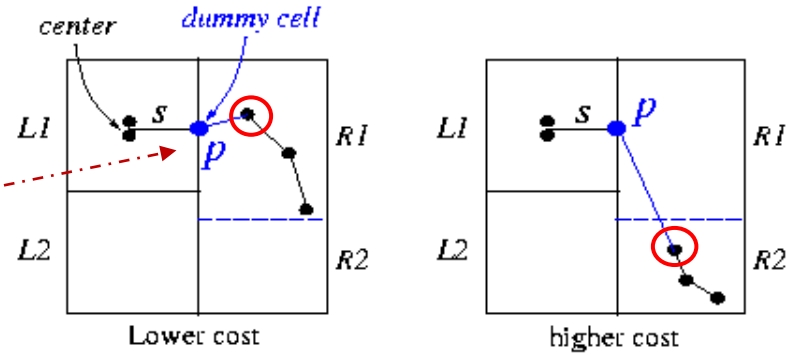




How about sizes?

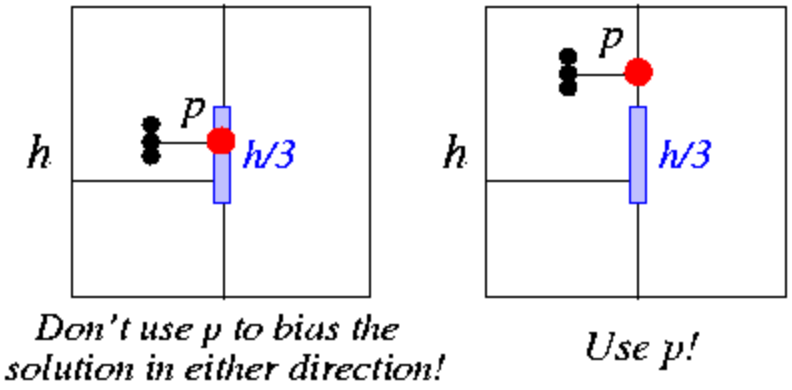
# Terminal Propagation

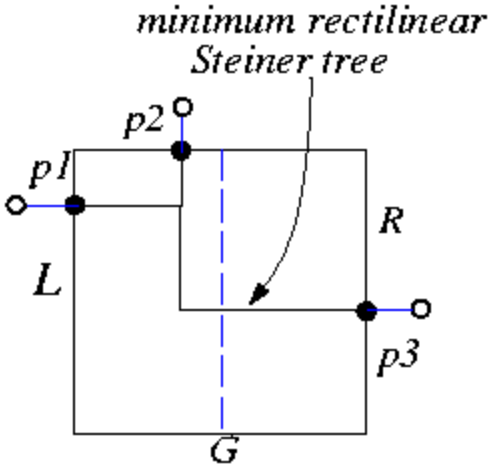
- We should use the fact that *s* is in $L_1$!

Center of pins of net-s in L1 is projected to p



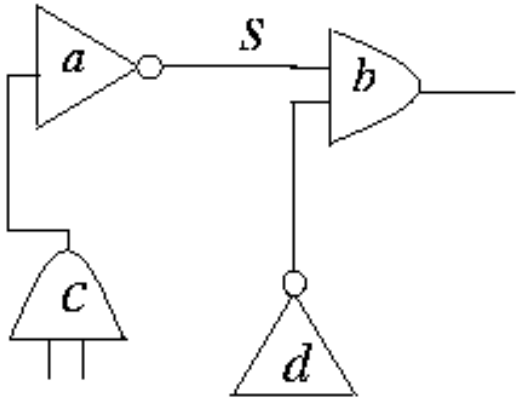

- When not to use *p* to bias partitioning? Net *s* has cells in many groups?

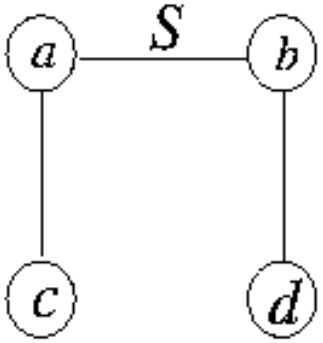


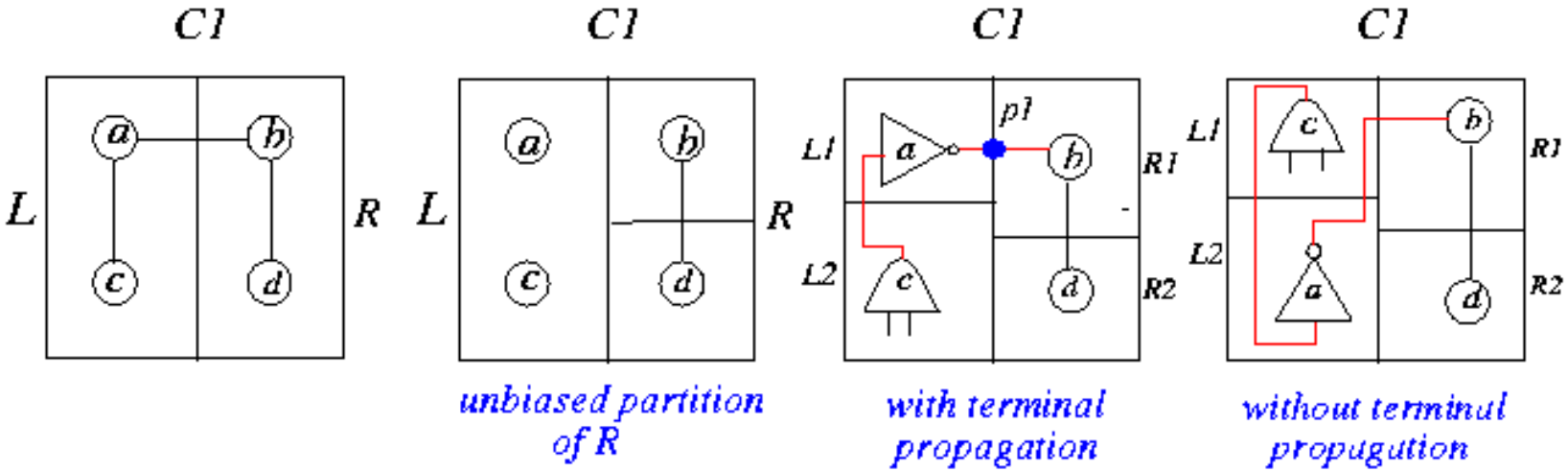



# Terminal Propagation Example

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



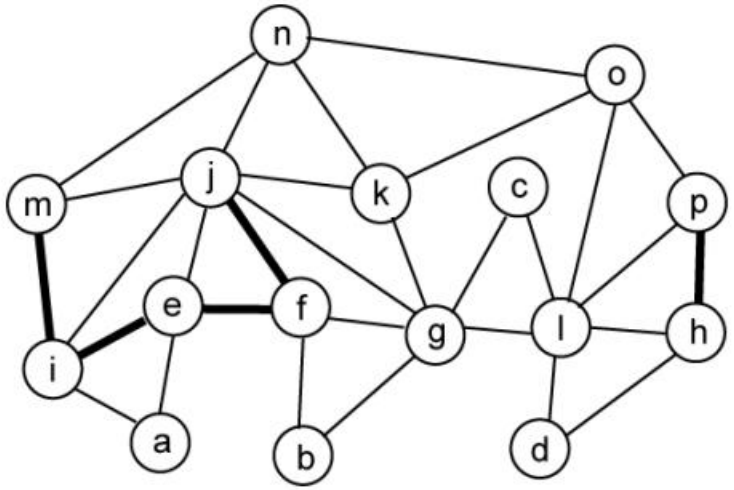
https://link.springer.com/book/10.1007/978-1-4020-6627-6

From Prof. Sung Kyu Lim (Georgia Tech.)

# Mincut Placement

- Perform quadrature mincut onto 4 × 4 grid
  - Start with vertical cut first

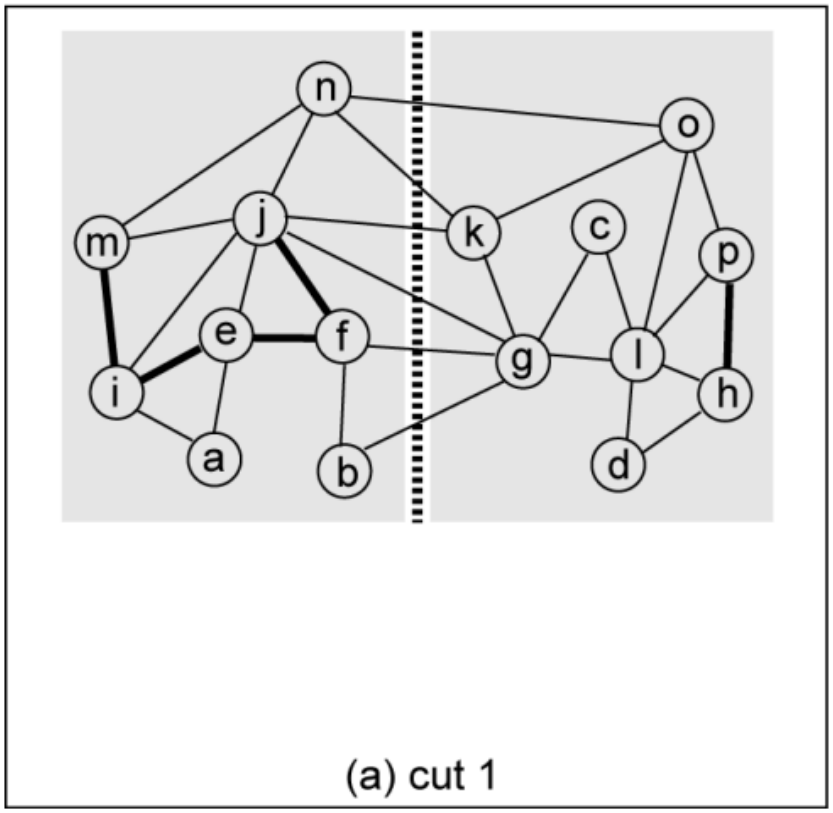
$$
\begin{aligned}
n_1 &= \{e, f\} \\
n_2 &= \{a, e, i\} \\
n_3 &= \{b, f, g\} \\
n_4 &= \{c, g, l\} \\
n_5 &= \{d, l, h\} \\
n_6 &= \{e, i, j\} \\
n_7 &= \{f, j\} \\
n_8 &= \{g, j, k\} \\
n_9 &= \{l, o, p\} \\
n_{10} &= \{h, p\} \\
n_{11} &= \{i, m\} \\
n_{12} &= \{j, m, n\} \\
n_{13} &= \{k, n, o\}
\end{aligned}
$$



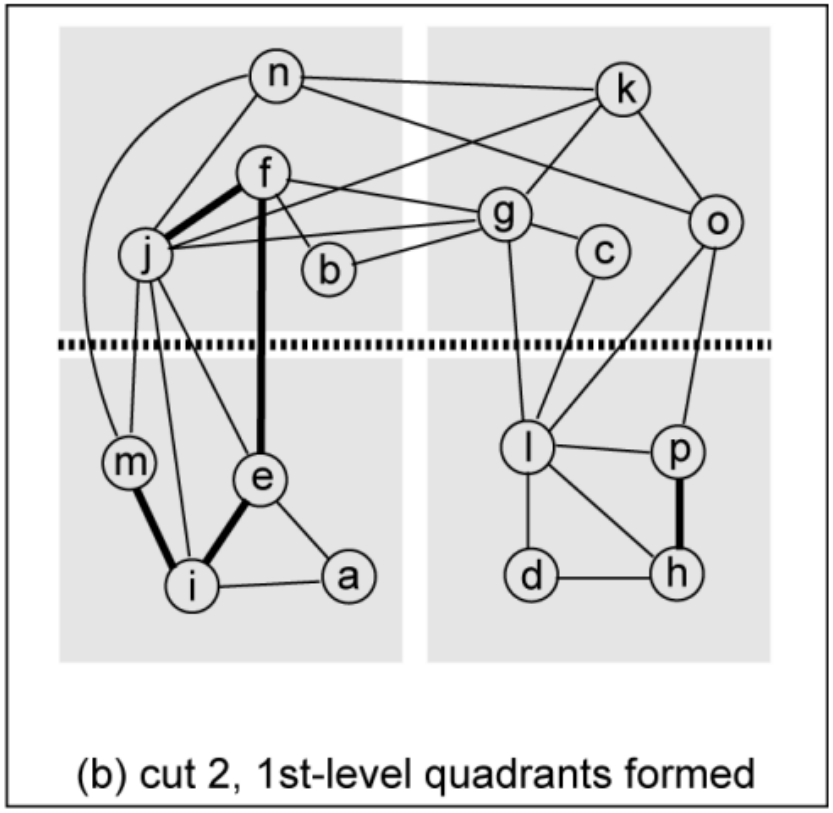

undirected graph model w/ k-clique weighting
thin edges = weight 0.5, thick edges = weight 1

# Cut 1 and 2

- First cut has min-cutsize of 3 (not unique)
  - Both cuts 1 and 2 divide the entire chip




(a) cut 1

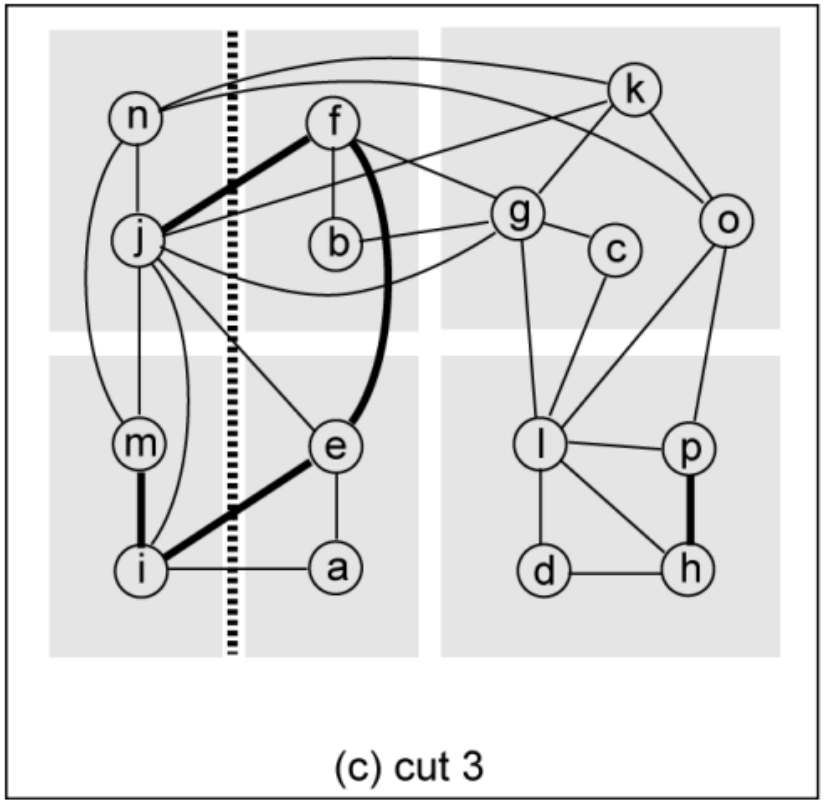



(b) cut 2, 1st-level quadrants formed

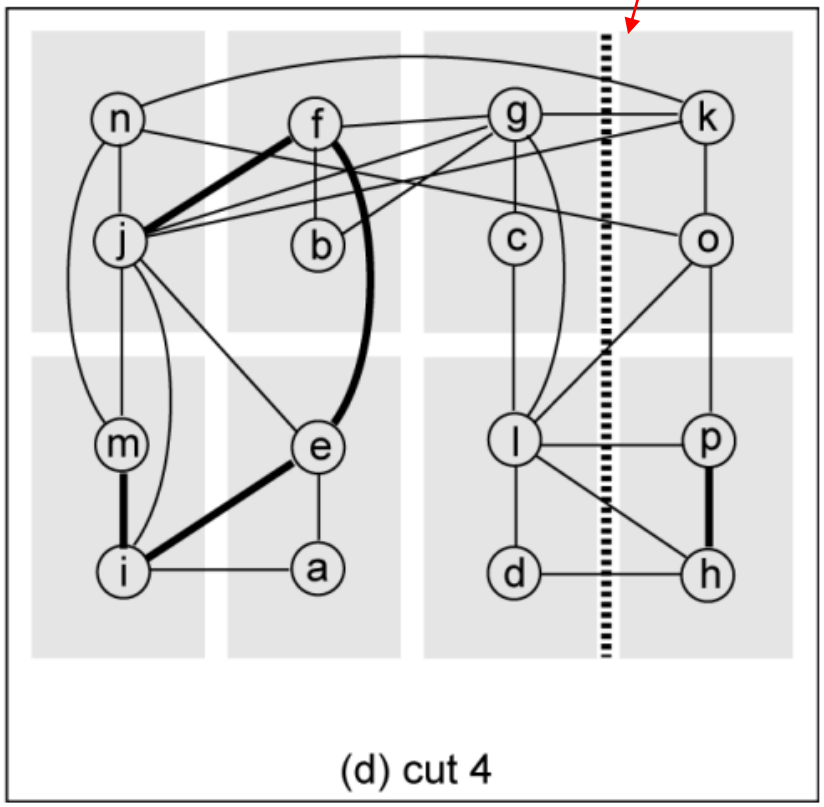
# Cut 3 and 4

- Each cut minimizes cutsize
  - Helps reduce overall wirelength

Describe what cut-4 will consider – at least 2 things:
1) No need to swap (k, d), …
2) If o stays at the right, how about p, k, l?




(c) cut 3

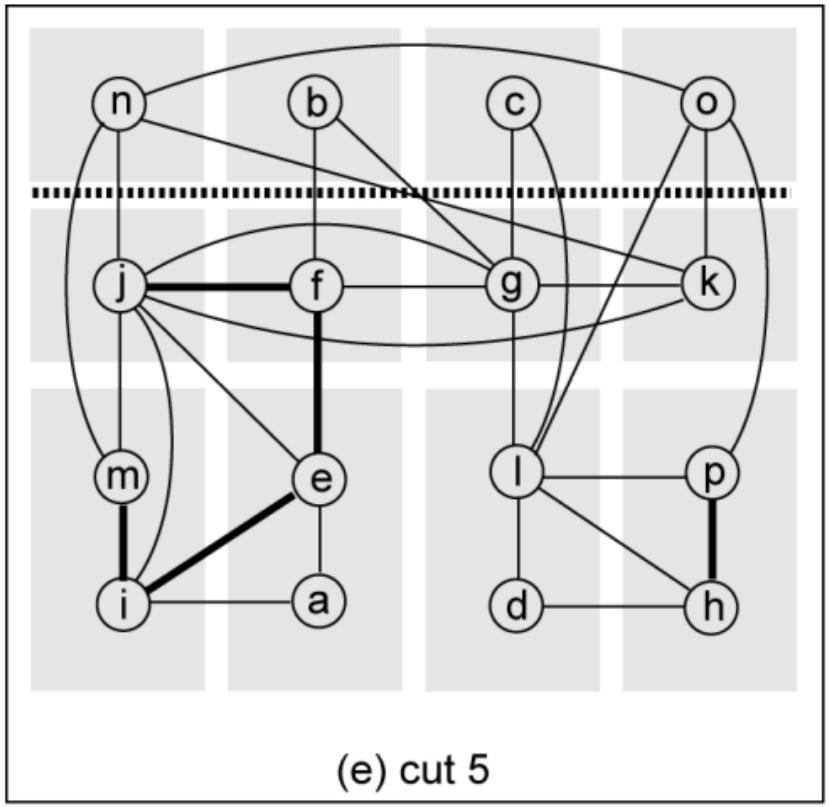



(d) cut 4

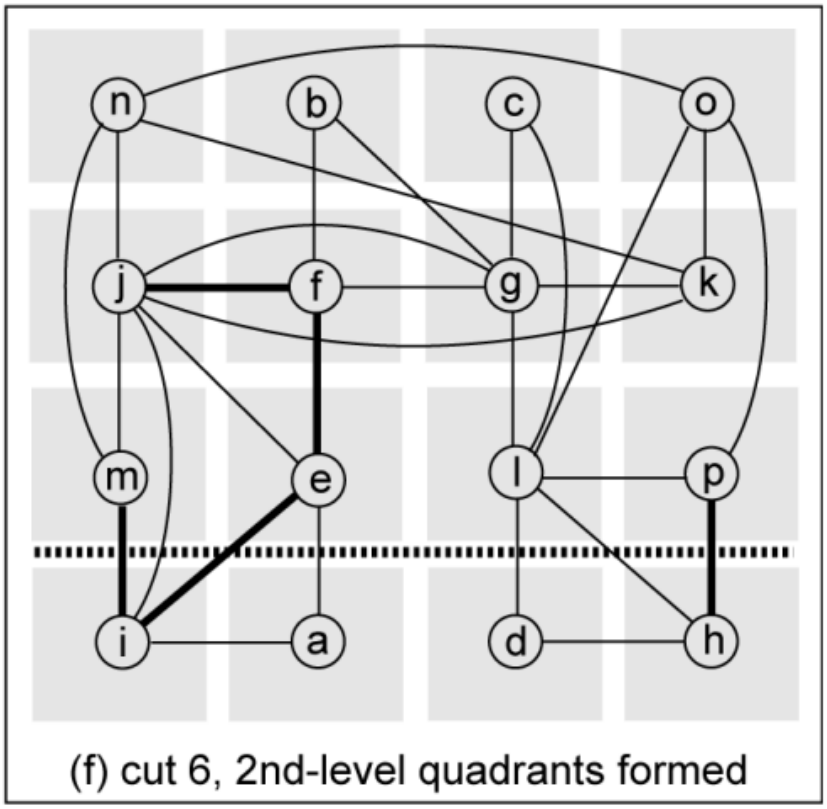
# Cut 5 and 6

- 16 partitions generated by 6 cuts
  - HPBB wirelength = 27

Quadrature partitioning program is not easy to be written



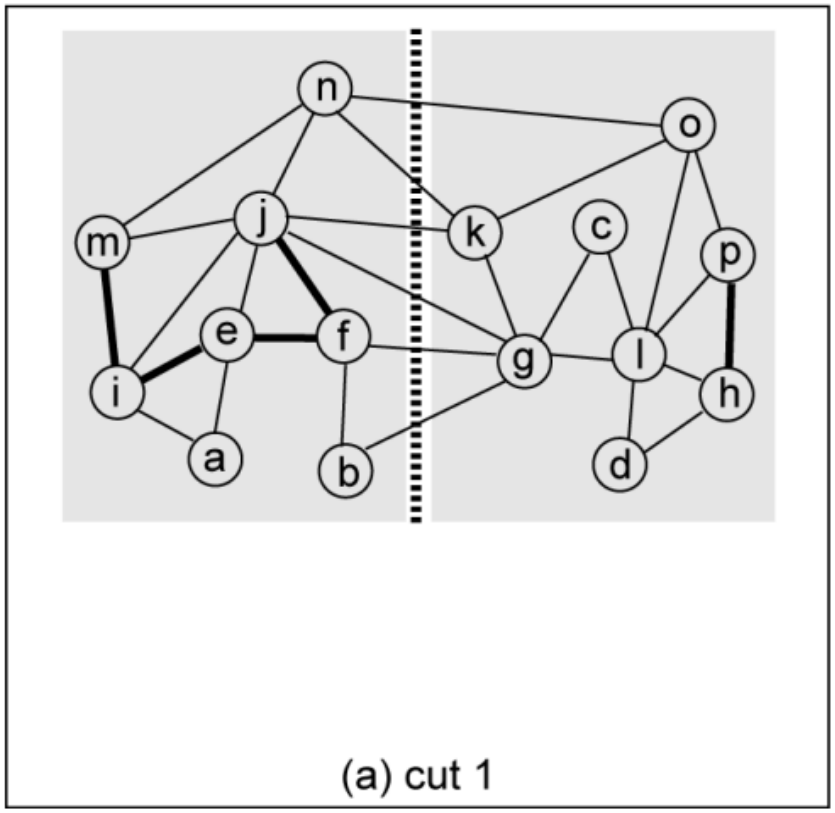

(e) cut 5



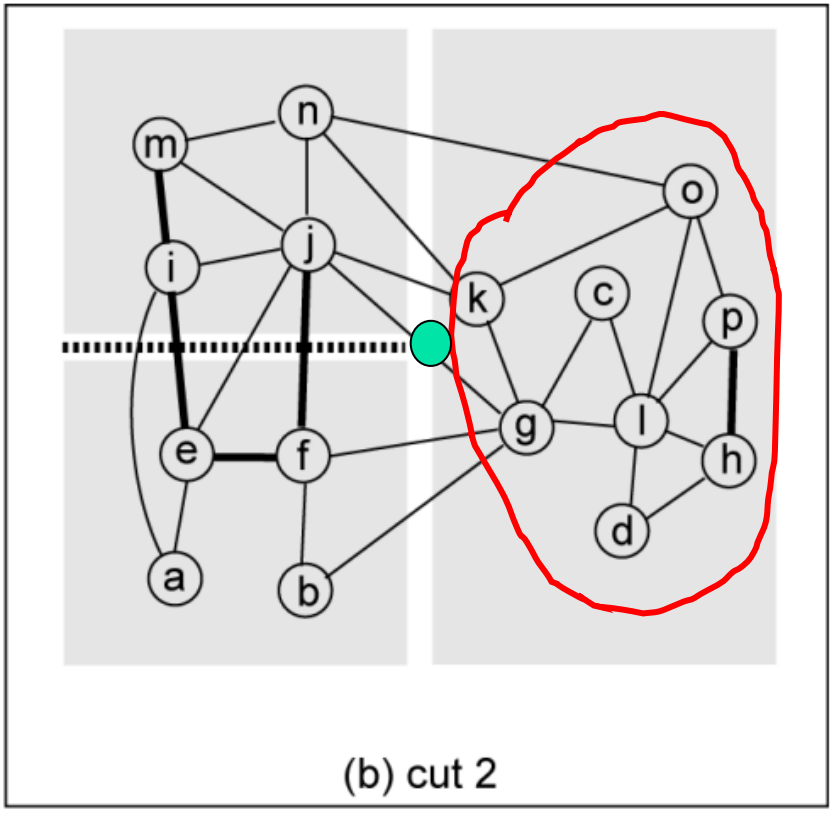

(f) cut 6, 2nd-level quadrants formed

# Recursive Bisection

- Start with vertical cut
  - Perform terminal propagation with middle third window



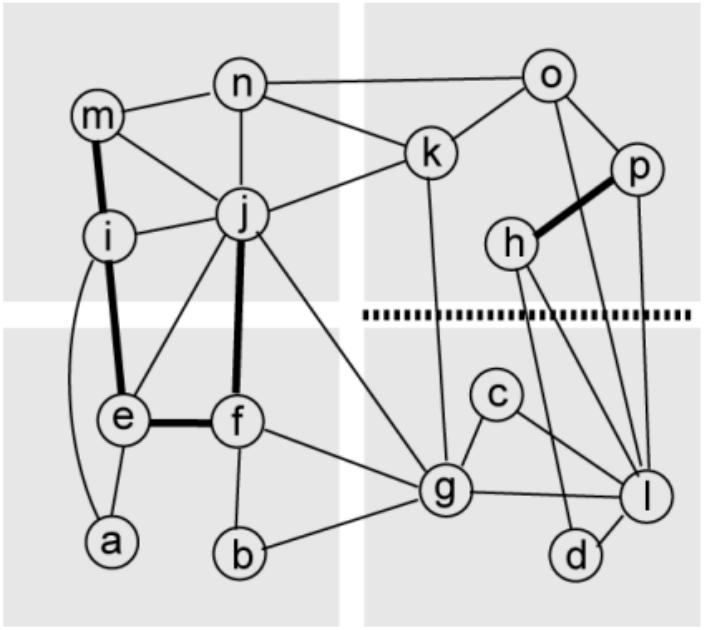

(a) cut 1

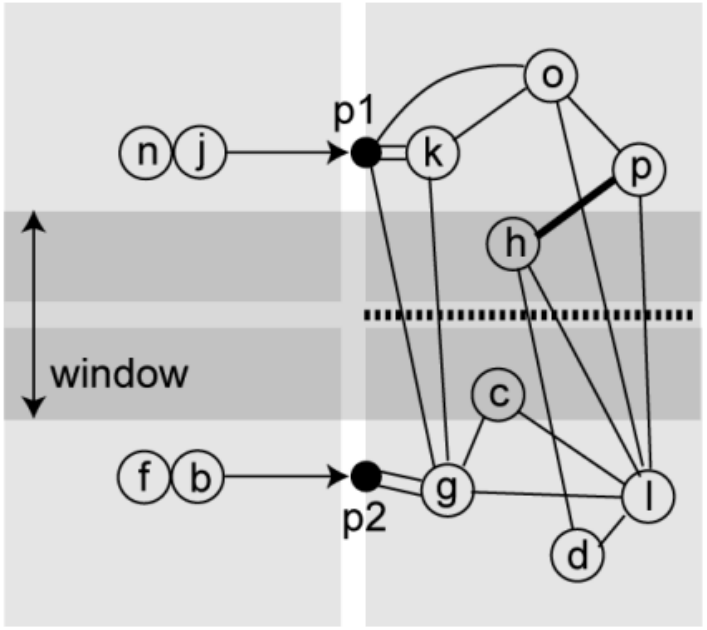



(b) cut 2

# Cut 3: Terminal Propagation

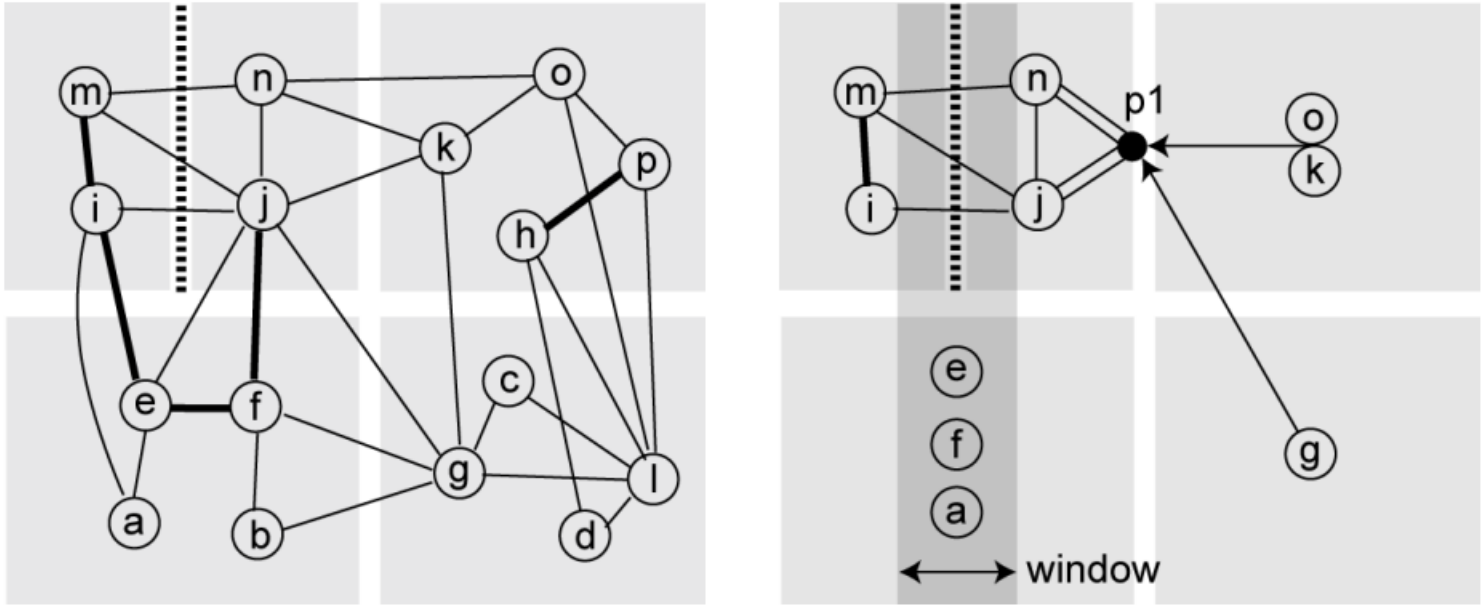
- Two terminals are propagated and are "pulling" nodes
  - Node $k$ and $o$ connect to $n$ and $j$: $p_1$ propagated (outside window)
  - Node $g$ connect to $j$, $f$ and $b$: $p_2$ propagated (outside window)
  - Terminal $p_1$ pulls $k/o/g$ to top partition, and $p_2$ pulls $g$ to bottom



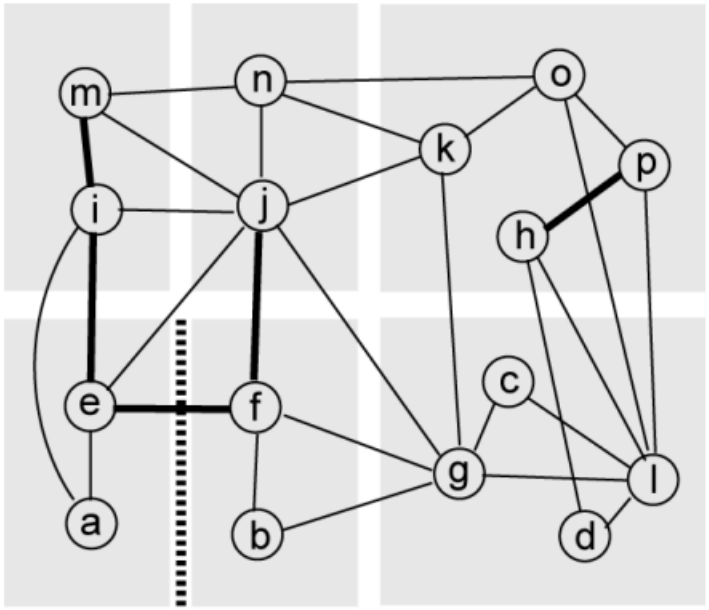
# Cut 4: Terminal Propagation

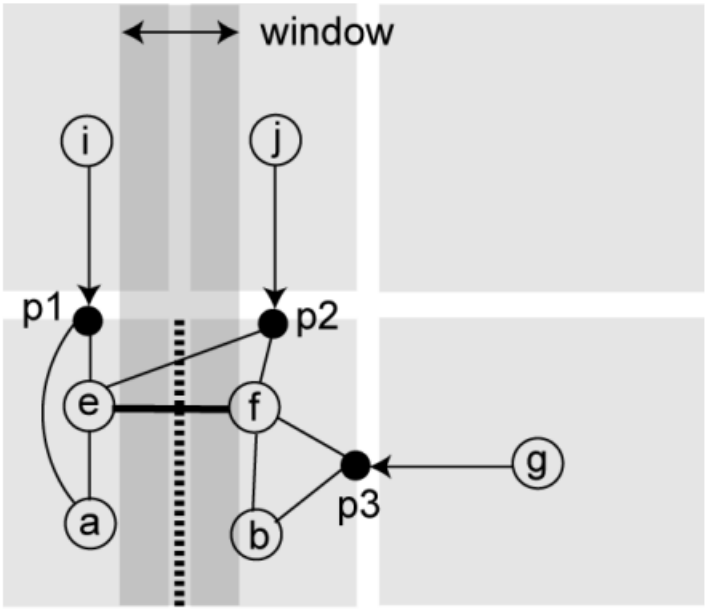
- One terminal propagated
  - Node $n$ and $j$ connect to $o/k/g$: $p_1$ propagated
  - Node $i$ and $j$ connect to $e/f/a$: no propagation (inside window)
  - Terminal $p_1$ pulls $n$ and $j$ to right partition



# Cut 5: Terminal Propagation

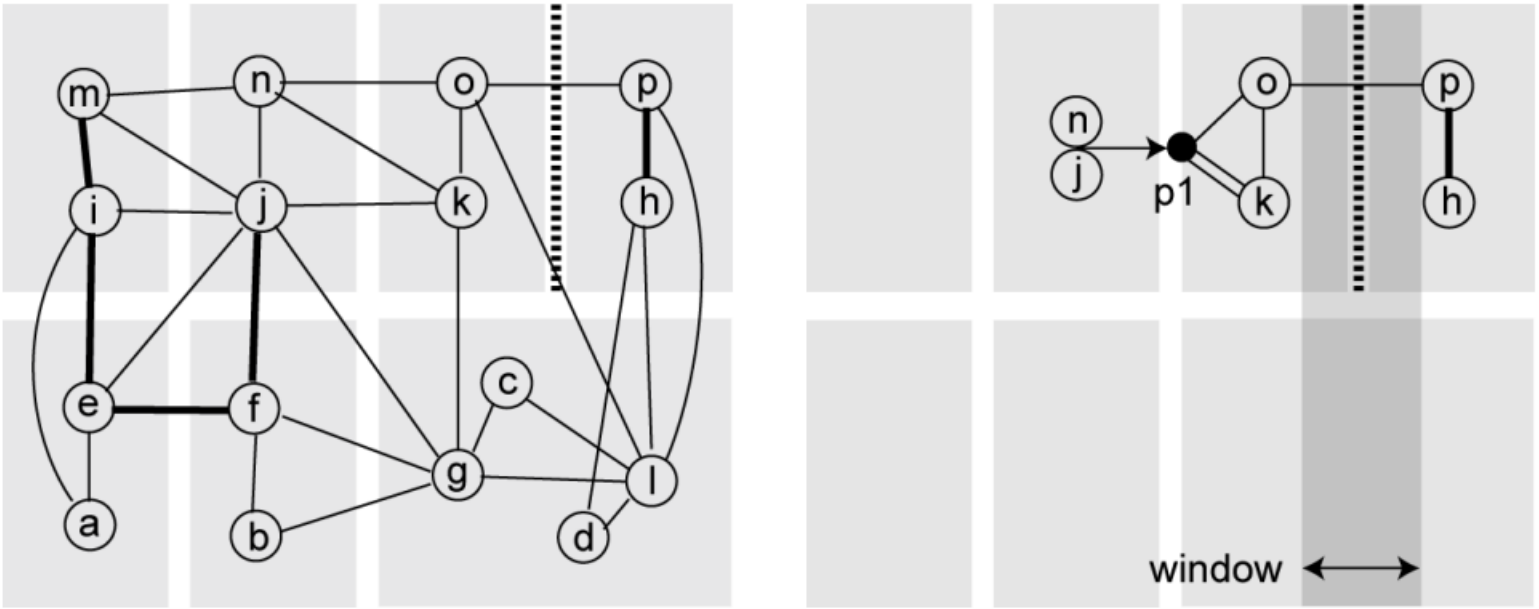
- Three terminals propagated
  - Node $i$ propagated to $p_1$, $j$ to $p_2$, and $g$ to $p_3$
  - Terminal $p_1$ pulls $e$ and $a$ to left partition
  - Terminal $p_2$ and $p_3$ pull $f/b/e$ to right partition



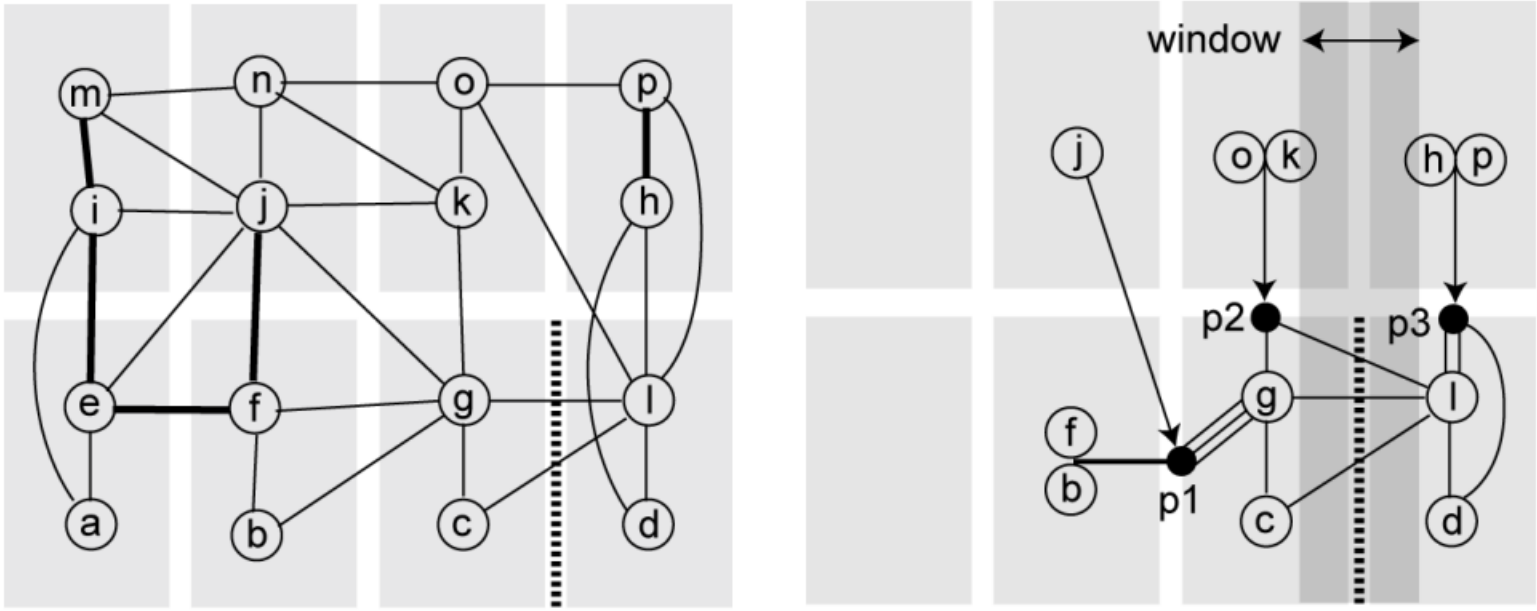
# Cut 6: Terminal Propagation

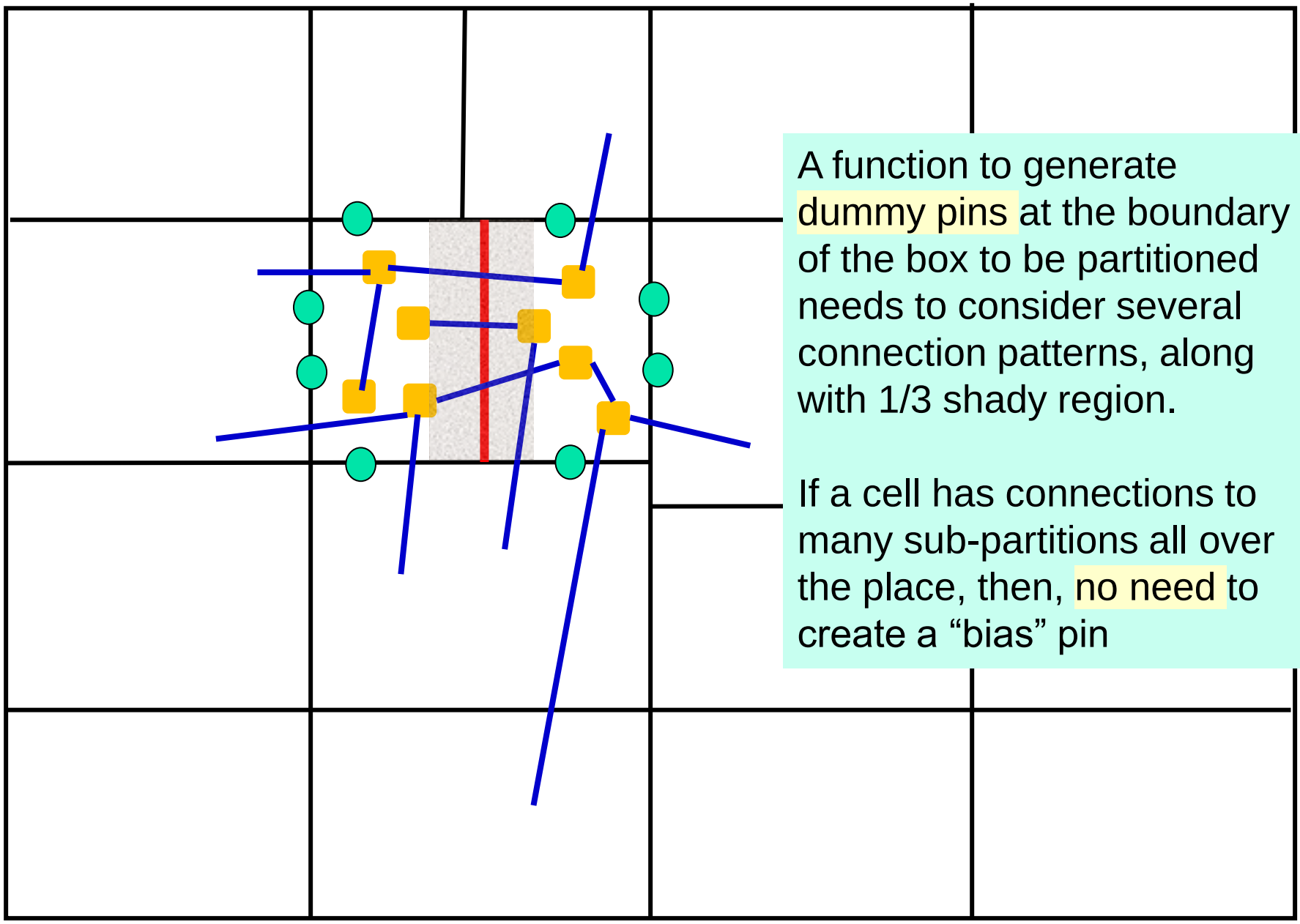
- One terminal propagated
  - Node $n$ and $j$ are propagated to $p_1$
  - Terminal $p_1$ pulls $o$ and $k$ to left partition



# Cut 7: Terminal Propagation

- Three terminals propagated
  - Node *j*/*f*/*b* propagated to $p_1$, *o*/*k* to $p_2$, and *h*/*p* to $p_3$
  - Terminal $p_1$ and $p_2$ pull *g* and *l* to left partition
  - Terminal $p_3$ pull *l* and *d* to right partition



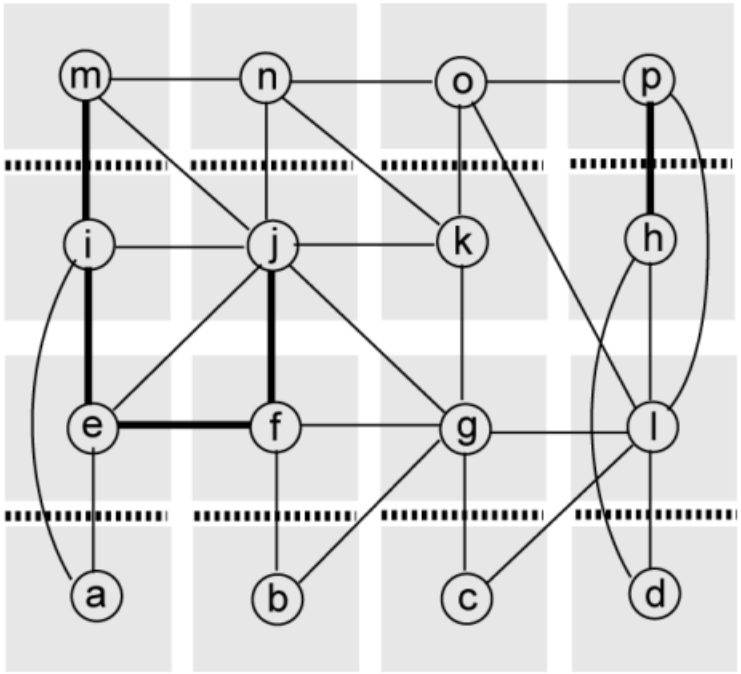


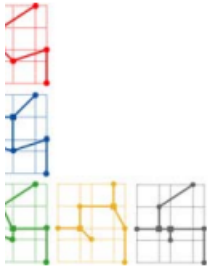
A function to generate dummy pins at the boundary of the box to be partitioned needs to consider several connection patterns, along with 1/3 shady region.

If a cell has connections to many sub-partitions all over the place, then, no need to create a “bias” pin

# Cut 8 to 15

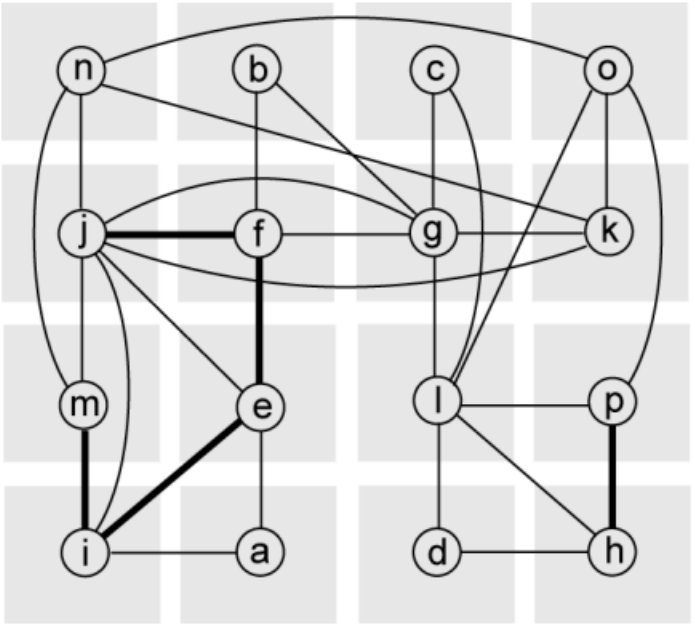
- 16 partitions generated by 15 cuts
  - HPBB wirelength = 23



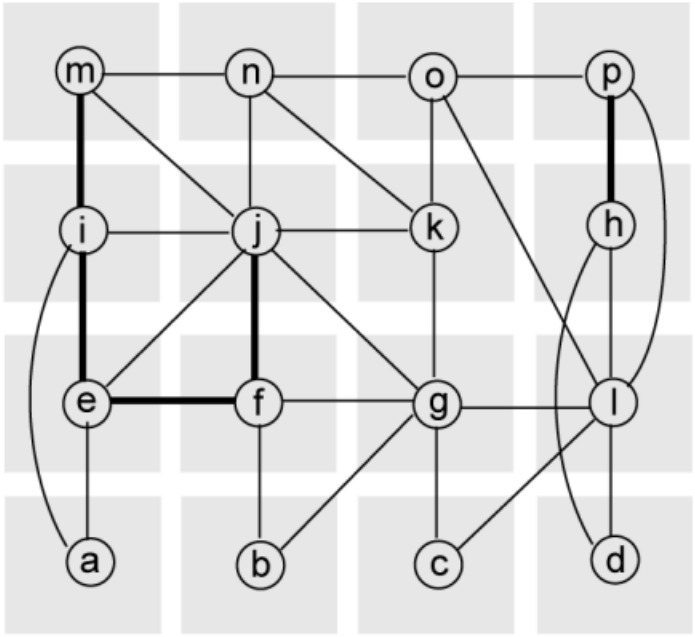



# Comparison

- Quadrature vs recursive bisection + terminal propagation
  - Number of cuts: 6 vs 15
  - Wirelength: 27 vs 23




quadrature

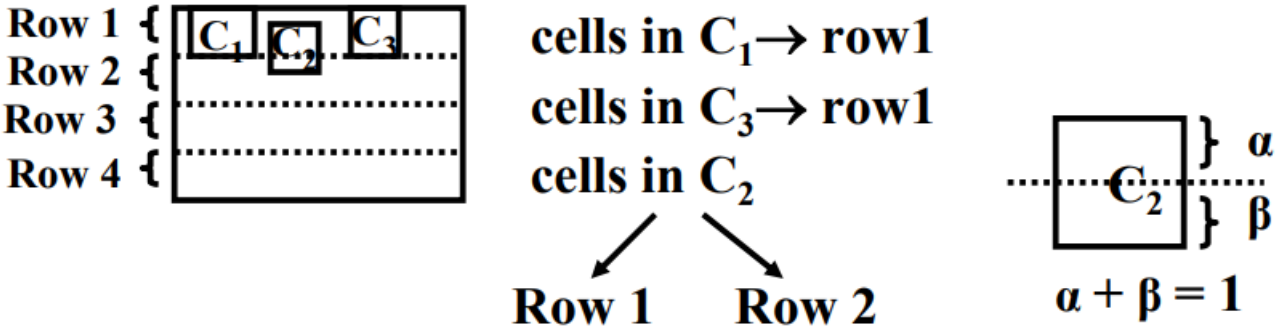



bisection

# Creating Rows

- **Terminal propagation reduce overall area by ~30%**
- **Creating rows**
  - **Choose α and β preferably to balance row to balance row length (during re-arrangement )**

After partitioning, need to legalize the cells' locations

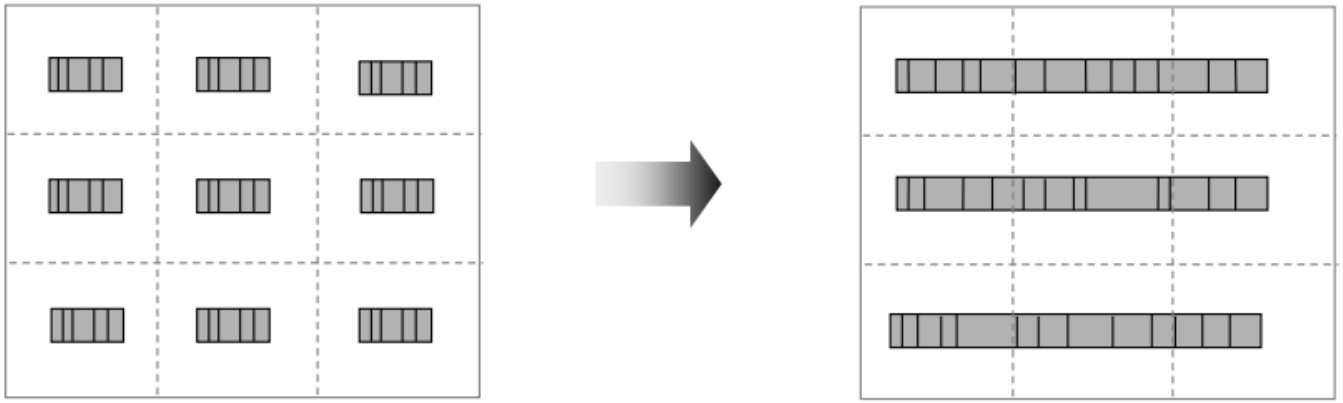


# Introduction

Global Placement

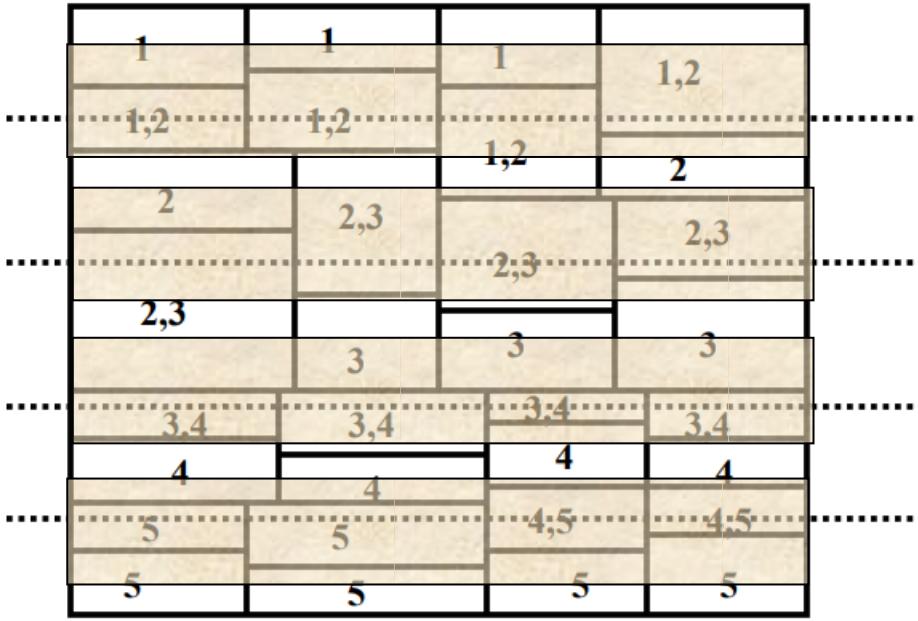
Detailed Placement

Legalize cell locations



# Creating Rows

- **Example**
  - **Partitioning of circuit into 32 groups**
  - **Each group is either assigned to a single row or divided into 2 rows**

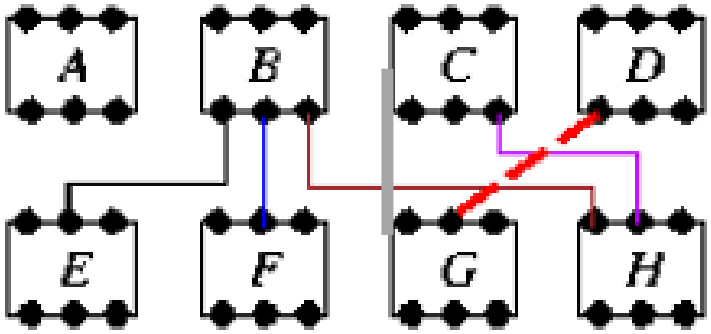



**a four-row standard cell design**

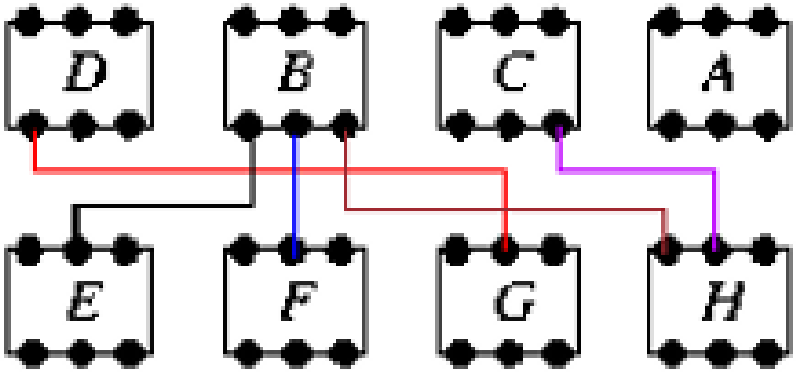
Then, legalize

After putting cells into rows

# Pairwise Swapping




*Shorter wirelength, 3 tracks required.*




*Density = 2 (2 tracks required)*

# Experimental Results

- CMOS Chip with 453 nets and 412 cells
- Manual solution
  - track density=147; feedthroughs=184
- Automated solution
  - without terminal propagation: t.d.=313; f.t.=591
  - (t.d. reduced to 235 by iterative interchanges)
  - with terminal propagation: t.d.=186; f.t.=182
  - (t.d. reduced to 152 by iterative interchanges)
  - Iterative Interchange Refinement is helpful
- The program is in production use as part of an automatic placement system in AT&T Bell Lab.
  - Solutions within 10% of the best hand layout

# Remarks on Min-cut Placement

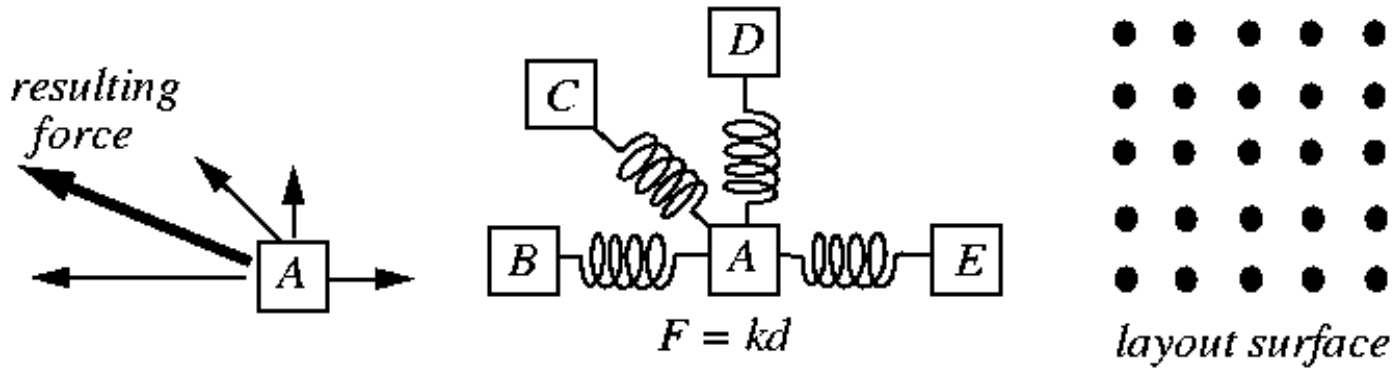
Fiduccia & Mattheyses (1982)

- **Also implemented F-M partitioning method**
  - **Much faster but solutions appeared to be not as good as K-L**
- **Use Simulated Annealing to do partitioning**
  - **Much slower. If restricted to a reasonable CPU time, solutions are of similar quality of those by F-M method. Easy to implement**
- **Seeking an elegant way to force some cells to be in particular positions**
- **Investigate other algorithms for terminal propagation**
  - **Terminal propagation is the bottleneck of CPU time**

# 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.

- Reduce 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

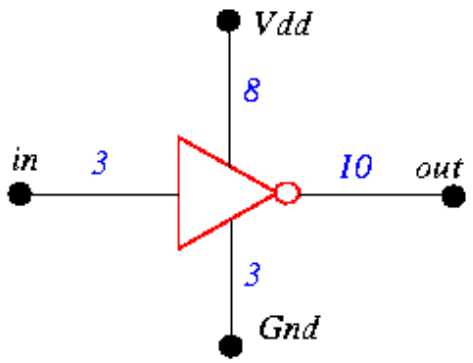
- Cell *i* connects to several cells *j*'s at distances $d_{ij}$'s by wires of weights $w_{ij}$'s. Total force: $F_i = \sum_j w_{ij} d_{ij}$
- The zero-force target location ( $\hat{x}_i$, $\hat{y}_i$ ) can be determined by equating the *x*- and *y*-components of the forces to zero:

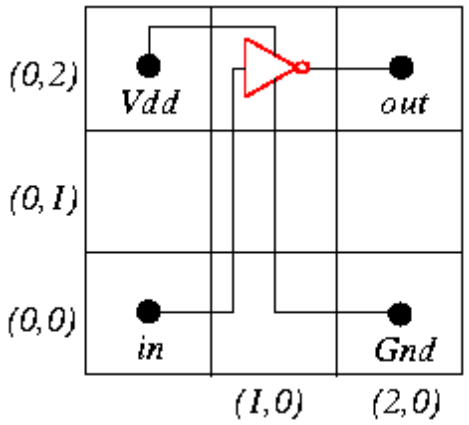
$$\sum_j w_{ij} \cdot (x_j - \hat{x}_i) = 0 \quad \Rightarrow \quad \hat{x}_i = \frac{\sum_j w_{ij} x_j}{\sum_j w_{ij}}$$

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

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

There are fixed objects



# Force-Directed Placement

- 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-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*.

**Algorithm: Force-Directed_Placement**

```
1 begin
2 Compute the connectivity for each cell;
3 Sort the cells in decreasing order of their connectivities into list L;
4 while (IterationCount < IterationLimit) do
5     Seed ← next module from L;
6     Declare the position of the seed vacant;
7         while (EndRipple = FALSE) do
8             Compute target location of the seed;
9             case the target location
10            VACANT:
11                Move seed to the target location and lock;
12                EndRipple ← TRUE; AbortCount ← 0;
13            SAME AS PRESENT LOCATION:
14                EndRipple ← TRUE; AbortCount ← 0;
15            LOCKED:
16                Move selected cell to the nearest vacant location;
17                EndRipple ← TRUE; AbortCount ← AbortCount + 1;
18                if (AbortCount > AbortLimit) then
19                    Unlock all cell locations;
19                    IterationCount ← IterationCount + 1;
20            OCCUPIED AND NOT LOCKED:
21                Select cell as the target location for next move;
22                Move seed cell to target location and lock the target location;
23                EndRipple ← FALSE; AbortCount ← 0;
26 end
```

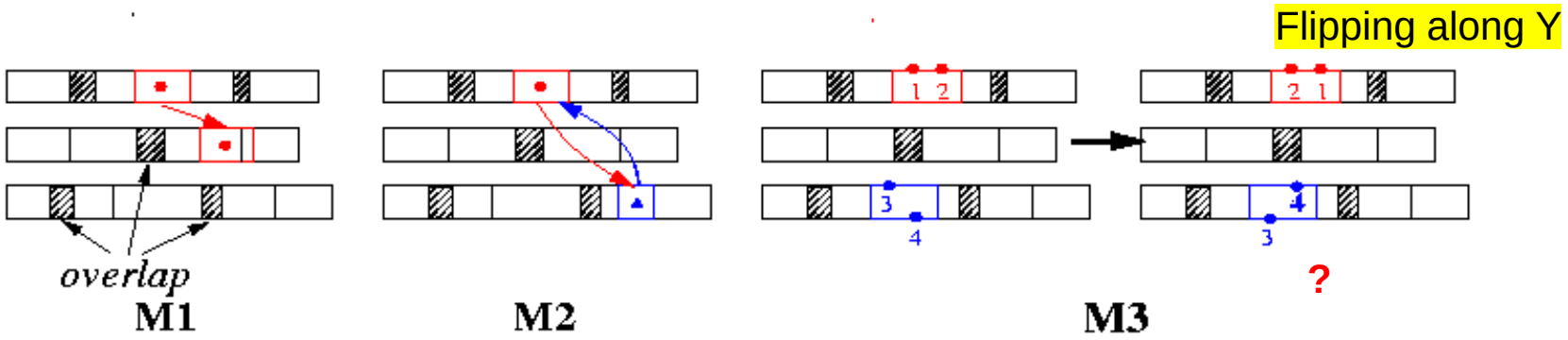
But initially many cells are still floating? Welcome suggestions

# 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.

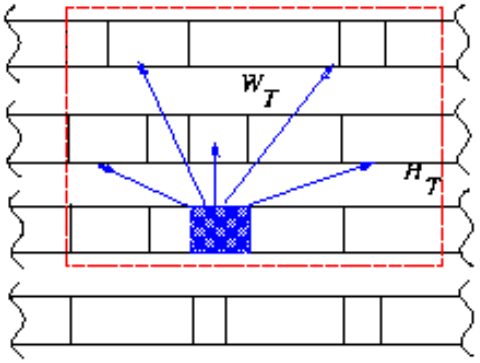
# Solution Space & Neighborhood Structure

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



# Cost Function

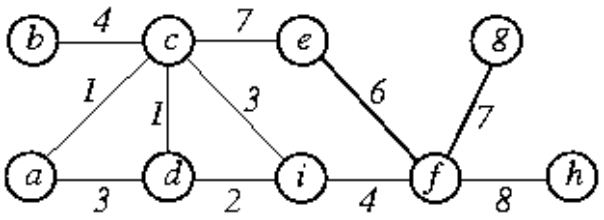
- Cost function: $C = C_1 + C_2 + C_3$.
- $C_1$: total estimated wirelength.
  - $C_1 = \sum_{i \in Nets}(\alpha_i w_i + \beta_i h_i)$
  - $\alpha_i$, $\beta_i$ are horizontal and vertical weights, respectively. ($\alpha_i$=1, $\beta_i$ =1 $\Rightarrow$ half perimeter of the bounding box of Net $i$.)
  - Critical nets: Increase both $\alpha_i$ and $\beta_i$ .
  - If vertical wirings are "cheaper" than horizontal wirings, use smaller vertical weights: $\beta_i < \alpha_i$.
- $C_2$: penalty function for module overlaps.
  - $C_2 = \gamma \sum_{i \neq j} O^2_{ij}$, $\gamma$: 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.
  - $C_2 = \delta \sum_{r \in Rows} |L_r - D_r|$, $\delta$ : 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 Genetic Algorithm

- Cohoon & Paris, “Genetic placement,” ICCAD-86.
- **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 |

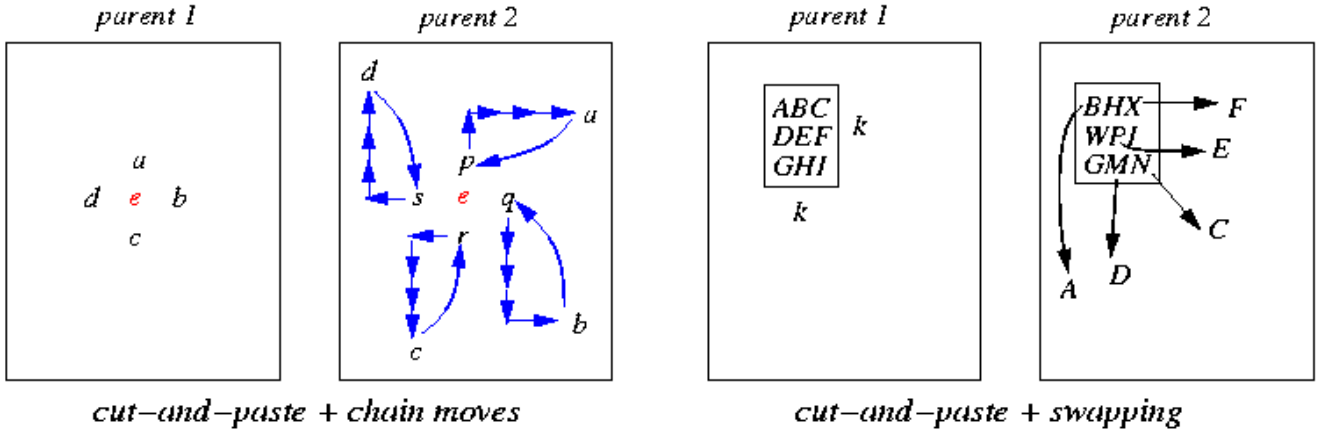
*string: aghcbidef*

# 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.

# Two More Crossover Operations

- Cut-and-paste + Chain moves:
  - Copy a randomly selected cell *e* and its four neighbors from parent 1 to parent 2.
  - The cells that earlier occupied the neighboring locations in parent 2 are shifted outwards.
- Cut-and-paste + Swapping
  - Copy $k \times k$ square modules from parent 1 to parent 2 ($k$: random # from a normal distribution with mean 3 and variance 1).
  - Swap cells not in both square modules.



# Genetic Operators: Mutation & 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 $P_{\mu}$ and $P_{i}$ respectively.

**Algorithm: Genetic_Placement**($N_p$, $N_g$, $N_o$, $P_i$, $P\mu$ )

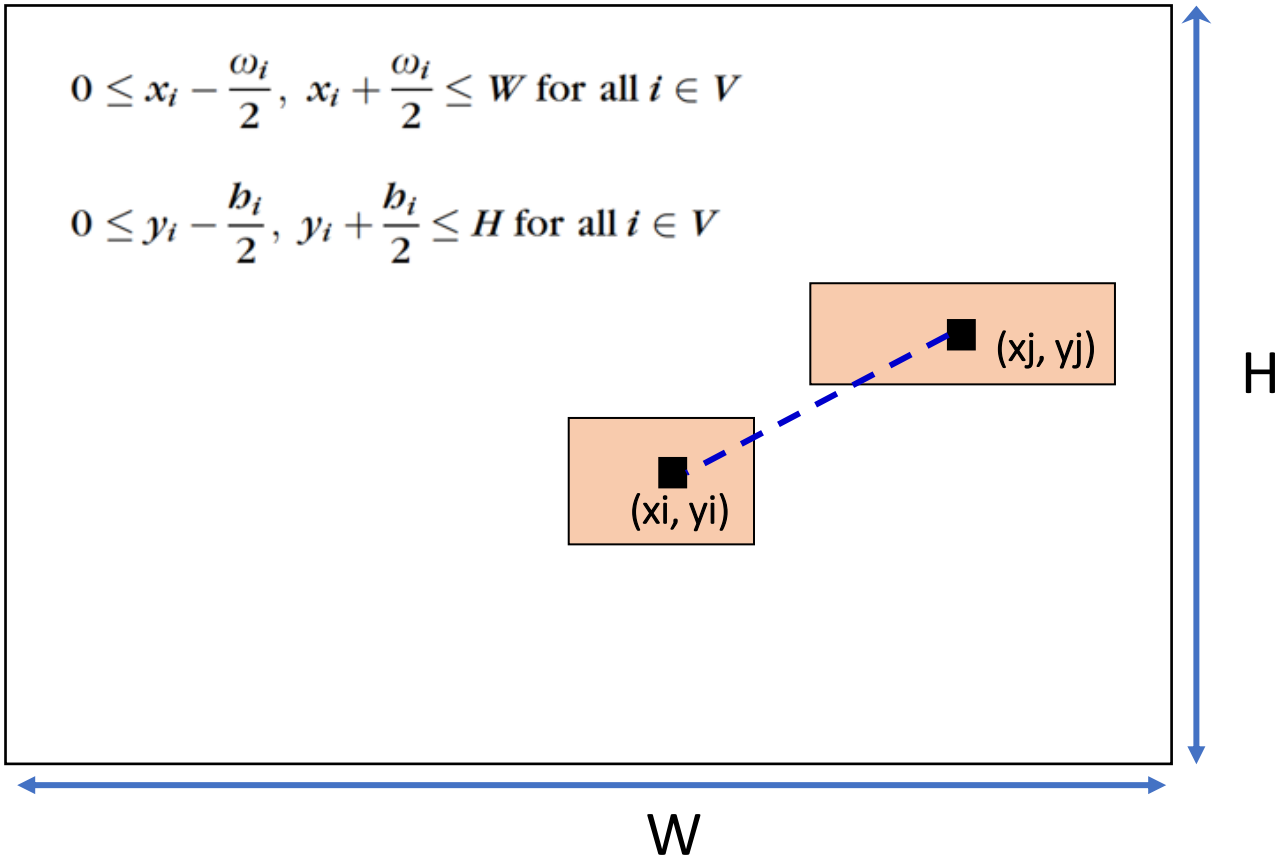
/* $N_p$: population size; */
/* $N_g$: # of generation; */
/* $N_o$: \# of offspring; */
/* $Pi$ : inversion probability; */
/* $P\mu$ : mutation probability; */

```
1 begin
2 ConstructPopulation(N_g);  /* randomly generate the initial population */
3 for j ← 1 to N_p
4     Evaluate Fitness(population(N_p));
5 for i ← 1 to N_g
6    for j ← 1 to N_o
7       (x, y) ← ChooseParents; /* choose parents with probability ∝ fitness value */
8       offspring(j) ← GenerateOffspring(x, y); /* perform crossover to generate offspring */
9      for h ← 1 to N_p
10        With probability Pμ, apply Mutation(population(h));
11     for h ← 1 to N_p
12        With probability P_i, apply Inversion(population(h));
13     Evaluate Fitness(offspring(j));
14    population ← Select(population, offspring, N_p);
15 return the highest scoring configuration in population;
16 end
```

# Genetic Placement Experiment: GINIE

- 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.
  - Needs more memory.

**Place all the cells within this bounding box (W, H)**




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

But HPWL objective function is not easy to be optimized

# Analytical Approach

http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD_Source/EDA_floorplanning.pdf

Analytical approach is a mathematical programming formulation that includes an objective function and a set of constraints. For the floorplanning problem, we need to consider two sets of basic constraints:

(1) the module nonoverlapping constraint and

(2) the dimension constraint

Our objective is to minimize the floorplan area, xy

# Analytical Approaches

- Quadratic vs non-quadratic techniques
- Chapter-11.5 (page 20) by Prof. Chris Chu

The placement problem can be written as the following mathematical program:

$$\begin{aligned}
\text{Minimize} \quad & \textstyle\sum_{e \in E} c_e \times \text{HPWL}_e(x_1, \ldots, x_n, y_1, \ldots, y_n) \\
\text{Subject to} \quad & \text{Overlap}_{ij}(x_i, y_i, x_j, y_j) = 0 \text{ for all } i,j \in V \text{ s.t. } i \neq j \\
& 0 \leq x_i - \frac{\omega_i}{2}, \; x_i + \frac{\omega_i}{2} \leq W \text{ for all } i \in V \\
& 0 \leq y_i - \frac{h_i}{2}, \; y_i + \frac{h_i}{2} \leq H \text{ for all } i \in V
\end{aligned}$$

let wi and hi be cell's width and height, and let xi and yi be the x- and y-coordinates of its center

This mathematical program is extremely difficult to handle.

$$L_{\{i,j\}} = |x_i - x_j| + |y_i - y_j|$$

This is usually referred to as linear wirelength. However, this function is not differentiable

# Quadratic Programming

Professors CK Cheng, RenSong Tsay also did something similar

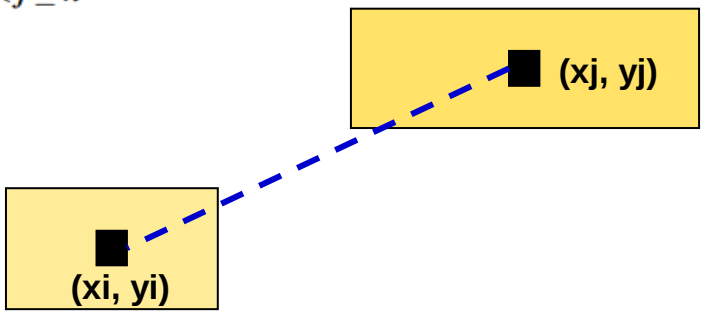
- So a common idea is to consider the squared Euclidean distance between the modules instead.

Quadratic wire length

$$\tilde{L}_{\{i,\,j\}} = (x_i - x_j)^2 + (y_i - y_j)^2$$

Quadratic placement

$$\tilde{L} = \frac{1}{2} \sum_{1 \le i < j \le n} c_{\{i,j\}} \times \left( (x_i - x_j)^2 + (y_i - y_j)^2 \right)$$




$$\left[ \tilde{L} = \frac{1}{2} \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{d}_x^{\,T} \boldsymbol{x} + \frac{1}{2} \boldsymbol{y}^T \boldsymbol{Q} \boldsymbol{y} + \boldsymbol{d}_y^{\,T} \boldsymbol{y} + \text{constant terms} \right]$$

Page 22

$$\tilde{L} = \frac{1}{2}(c_{12}((x_1 - x_2)^2 + (y_1 - y_2)^2)$$

$$+(c_{13}((x_1 - x_3)^2 + (y_1 - y_3)^2)$$
$$+(c_{14}((x_1 - x_4)^2 + (y_1 - y_4)^2)$$
$$+(c_{24}((x_2 - x_4)^2 + (y_2 - y_4)^2)$$
$$+(c_{25}((x_2 - x_5)^2 + (y_2 - y_5)^2)$$
$$+(c_{36}((x_3 - x_6)^2 + (y_3 - y_6)^2)$$

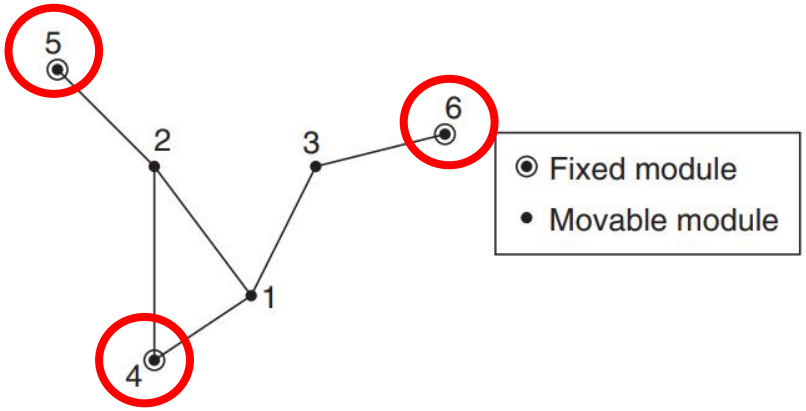



**FIGURE 11.11**

Connections of a circuit.

$$= \frac{1}{2}x^T Qx + d_x^T x + \frac{1}{2}y^T Qy + d_y^T y$$

$$+\frac{1}{2}((c_{14} + c_{24})x_4^2 + c_{25}x_5^2 + c_{36}x_6^2 + (c_{14} + c_{24})y_4^2 + c_{25}y_5^2 + c_{36}y_6^2)$$

where

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \; y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix},$$

$$Q = \begin{pmatrix} c_{12} + c_{13} + c_{14} & 0 & 0 \\ 0 & c_{21} + c_{24} + c_{25} & 0 \\ 0 & 0 & c_{31} + c_{36} \end{pmatrix} - \begin{pmatrix} 0 & c_{12} & c_{13} \\ c_{21} & 0 & c_{23} \\ c_{31} & c_{32} & 0 \end{pmatrix}$$

$$d_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix} \text{ and } d_y = \begin{pmatrix} -c_{14}y_4 \\ -c_{24}y_4 - c_{25}y_5 \\ -c_{36}y_6 \end{pmatrix}$$

$$= \frac{1}{2}\Big( c_{12}\big((x_1^2 - 2x_1x_2 + x_2^2) + (y_1^2 - 2y_1y_2 + y_2^2)\big)$$

$$+ c_{13}\big((x_1^2 - 2x_1x_3 + x_3^2) + (y_1^2 - 2y_1y_3 + y_3^2)\big)$$

$$+ c_{14}\big((x_1^2 - 2x_1x_4 + x_4^2) + (y_1^2 - 2y_1y_4 + y_4^2)\big)$$

$$+ c_{24}\big((x_2^2 - 2x_2x_4 + x_4^2) + (y_2^2 - 2y_2y_4 + y_4^2)\big)$$

$$+ c_{25}\big((x_2^2 - 2x_2x_5 + x_5^2) + (y_2^2 - 2y_2y_5 + y_5^2)\big)$$

$$+ c_{36}\big((x_3^2 - 2x_3x_6 + x_6^2) + (y_3^2 - 2y_3y_6 + y_6^2)\big)\Big)$$

$$= \frac{1}{2}\left(C_{12}\left(x_1^2 + x_2^2 + y_1^2 + y_2^2\right)\right) - C_{12}\left(x_1x_2 + y_1y_2\right)$$

$$+ \frac{1}{2}\left(C_{13}\left(x_1^2 + x_3^2 + y_1^2 + y_3^2\right)\right) - C_{13}\left(x_1x_3 + y_1y_3\right)$$

$$+ \frac{1}{2}\,C_{14}\left(x_1^2 + x_4^2 + y_1^2 + y_4^2\right) - C_{14}\left(x_1x_4 + y_1y_4\right)$$

$$+ \frac{1}{2}\,C_{24}\left(x_2^2 + x_4^2 + y_2^2 + y_4^2\right) - C_{24}\left(x_2x_4 + y_2y_4\right)$$

$$+ \frac{1}{2}\,C_{25}\left(x_2^2 + x_5^2 + y_2^2 + y_5^2\right) - C_{25}\left(x_2x_5 + y_2y_5\right)$$

$$+ \frac{1}{2}\,C_{36}\left(x_3^2 + x_6^2 + y_3^2 + y_6^2\right) - C_{36}\left(x_3x_6 + y_3y_6\right)$$

$$= (x_1\ x_2\ x_3)\begin{bmatrix} c_{12}+c_{13}+c_{14} & 0 & 0 \\ 0 & c_{21}+c_{24}+c_{25} & 0 \\ 0 & 0 & c_{31}+c_{36} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= \begin{pmatrix} (c_{12}+c_{13}+c_{14})x_1 & (c_{21}+c_{24}+c_{25})x_2 & (c_{31}+c_{36}) \\ & & x_3 \end{pmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= (c_{12}+c_{13}+c_{14})x_1^2 + (c_{21}+c_{24}+c_{25})x_2^2 + (c_{31}+c_{36})x_3^2$$

$$\tilde{L} = \frac{1}{2}(c_{12}((x_1 - x_2)^2 + (y_1 - y_2)^2)$$

$$+(c_{13}((x_1 - x_3)^2 + (y_1 - y_3)^2)$$
$$+(c_{14}((x_1 - x_4)^2 + (y_1 - y_4)^2)$$
$$+(c_{24}((x_2 - x_4)^2 + (y_2 - y_4)^2)$$
$$+(c_{25}((x_2 - x_5)^2 + (y_2 - y_5)^2)$$
$$+(c_{36}((x_3 - x_6)^2 + (y_3 - y_6)^2)$$

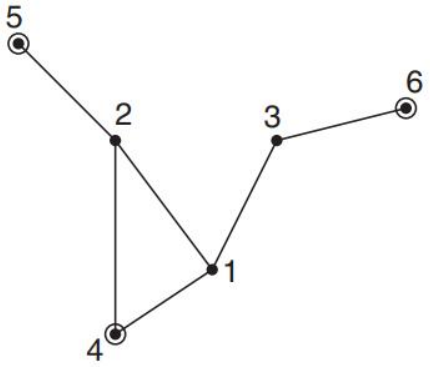



**FIGURE 11.11**

Connections of a circuit.

Welcome to add another movable node, then, get matrices, compare

$$= \frac{1}{2}x^T Qx + d_x^T x + \frac{1}{2}y^T Qy + d_y^T y$$

$$+\frac{1}{2}((c_{14} + c_{24})x_4^2 + c_{25}x_5^2 + c_{36}x_6^2 + (c_{14} + c_{24})y_4^2 + c_{25}y_5^2 + c_{36}y_6^2)$$

where

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \; y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix},$$

$$Q = \begin{pmatrix} c_{12} + c_{13} + c_{14} & 0 & 0 \\ 0 & c_{21} + c_{24} + c_{25} & 0 \\ 0 & 0 & c_{31} + c_{36} \end{pmatrix} - \begin{pmatrix} 0 & c_{12} & c_{13} \\ c_{21} & 0 & c_{23} \\ c_{31} & c_{32} & 0 \end{pmatrix}$$

$$d_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix} \text{ and } d_y = \begin{pmatrix} -c_{14}y_4 \\ -c_{24}y_4 - c_{25}y_5 \\ -c_{36}y_6 \end{pmatrix}$$

It is clear that $\tilde{L}_{\{i,j\}}$ for all $\{i, j\} \in E$ are convex and continuously differentiable functions. Hence, $\tilde{L}$, which is a weighted sum of $\tilde{L}_{\{i,j\}}$'s, is also convex and differentiable. So $\tilde{L}$ should be easy to minimize. In particular,

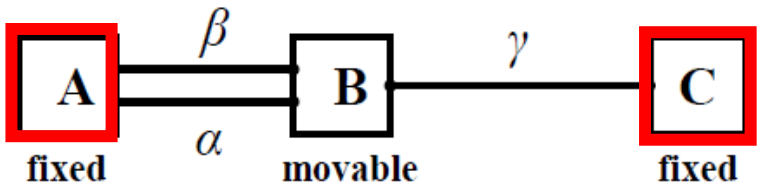
$$\frac{\partial \tilde{L}}{\partial x} = \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x \text{ and } \frac{\partial \tilde{L}}{\partial y} = \boldsymbol{Q}\boldsymbol{y} + \boldsymbol{d}_y$$

Therefore, the placement with minimum wirelength is given by

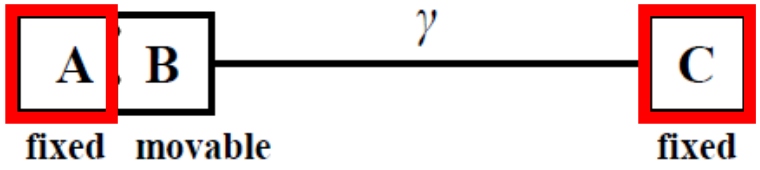
$$\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x = 0 \text{ and } \boldsymbol{Q}\boldsymbol{y} + \boldsymbol{d}_y = 0 \qquad (11.1)$$

In other words, if **nonoverlapping constraints are ignored**, the quadratic placement problem is equivalent to solving **a system of linear equations**. If all movable modules are connected to fixed modules either directly or indirectly, **Q is positive definite** and thus invertible

# Linear vs. Quadratic Objective




Quadratic objective function




Linear objective function

**Quadratic:**

$$\varphi_q = l_\alpha^2 + l_\beta^2 + l_\gamma^2 = 2(l - l_\gamma)^2 + l_\gamma^2$$

$$\varphi_q' = -4(l - l_\gamma) + 2l_\gamma = 0, \text{So the optimal } l_\gamma = \frac{2}{3}l$$

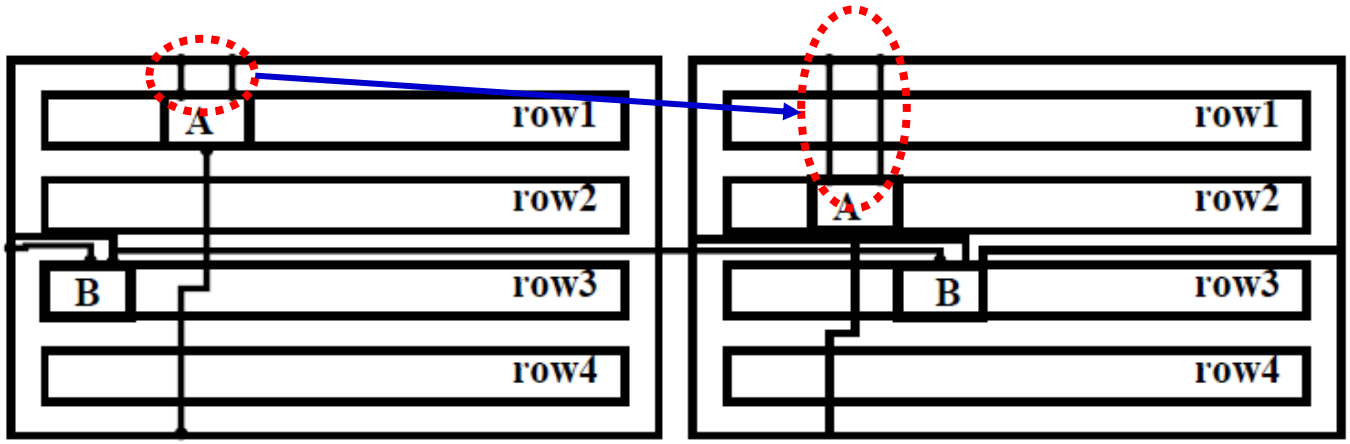
Long wire tends to become shorter

**Linear:**

$$\varphi_l = l_\alpha + l_\beta + l_\gamma, \text{So the optimal } l_\gamma = l$$

# Linear vs. Quadratic Objective

- **Quadratic objective function**
  - tends to make very long net shorter than linear objective function
  - lets short nets become slightly longer




Linear objective function  Quadratic objective function

placement_Sung_Kyu_Lim.pdf
(from page 32)

# Analytical Placement

- **Gordian package:**
  - **GORDIAN: Gordian: VLSI Placement by Quadratic Programming and slicing Optimization: J. M. Kleinhans, G.Sigl, F.M. Johannes, K.J. Antreich, IEEE TCAD, 1991**
  - **GORDIAN-L: Analytical Placement: A Linear or a Quadratic Objective Function?: G. Sigl, K. Doll, F.M. Johannes, DAC91**
- **Gordian: A Quadratic Placement Approach**
  - **Global optimization: solves a sequence of quadratic programming problems**
  - **Partitioning: enforces the non-overlap constraints**

# Quadratic Programming (QP)

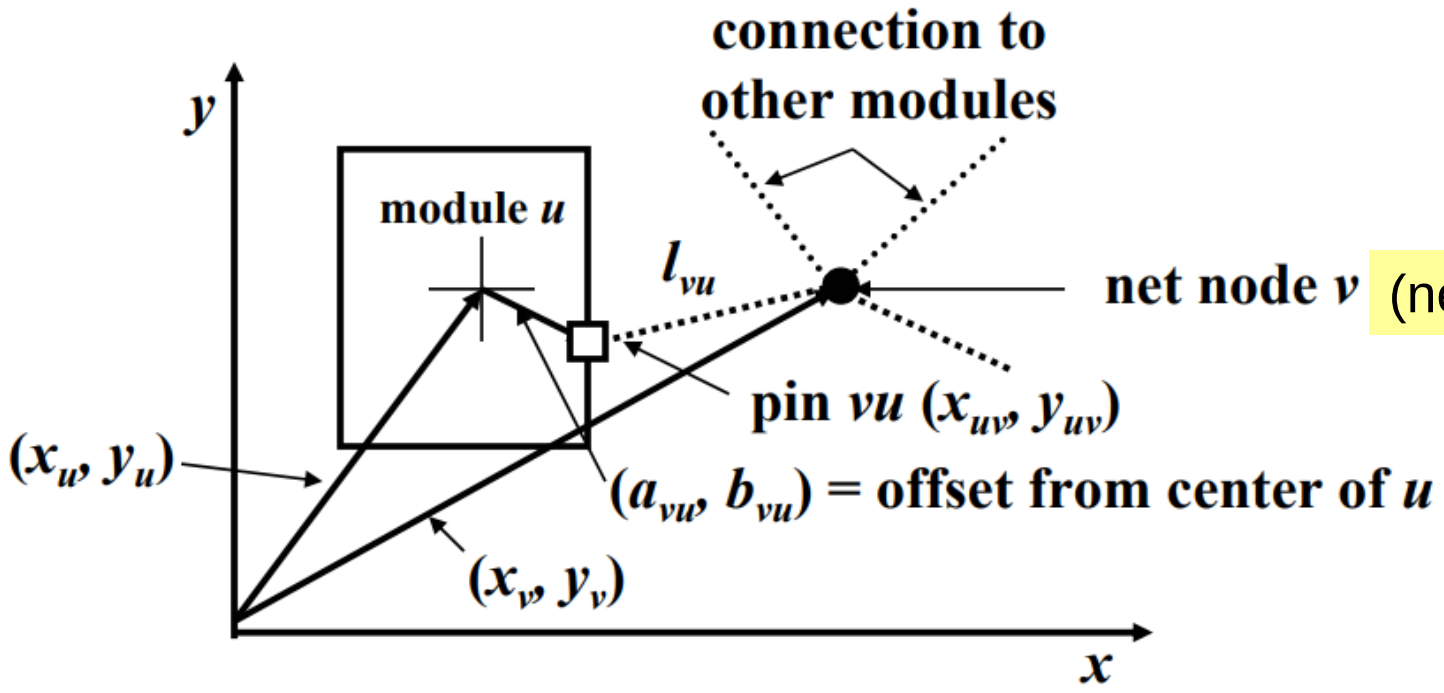
- **Definition**
  - Process of solving optimization problems involving quadratic functions
  - One seeks to optimize (minimize or maximize) a multivariate quadratic function subject to linear constraints on the variables

- **QP with n variables and m constraints**

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^{\mathrm{T}} Q \mathbf{x} + \mathbf{c}^{\mathrm{T}} \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \preceq \mathbf{b}, \end{aligned}$$

  - n-dimensional vector c
  - n × n-dimensional real symmetric matrix Q
  - m × n-dimensional real matrix A
  - m-dimensional real vector b

# Problem Definition




**Squared wire length of net *v***

$$L_v = \sum_{u \in M_v} [(x_{uv} - x_v)^2 + (y_{uv} - y_v)^2]$$

$$x_{uv} = x_u + a_{vu}, \; y_{uv} = y_u + b_{vu}$$

# Cost Function

- **Minimize the following:**

$$\phi = \frac{1}{2}\sum_{v \in N} L_v w_v$$

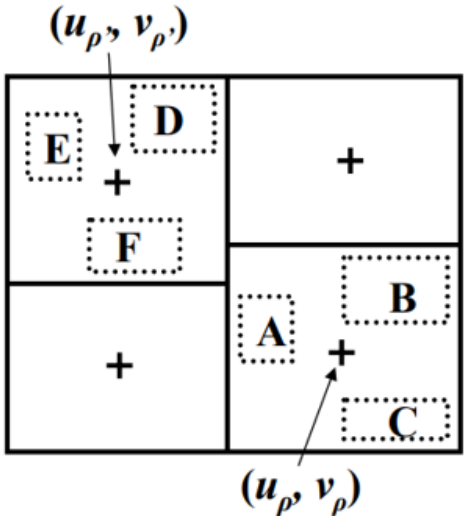
$$\phi(x, y) = X^T CX + d_x^T X + Y^T CY + d_y^T Y$$

$$\phi(x) = X^T CX + d^T X$$

# Constraints

- **The center of gravity constraints**
  - At level $l$, chip is divided into $q$ ($\leq 2^l$) regions
  - For region $p$, the center coordinates: $(u_p, v_p)$
  - $M_p$: set of modules in region $p$
  - Matrix from for all regions

$$\sum_{m \in M_p} F_m \cdot x_m = u_p \times \sum_{m \in M_p} F_m$$

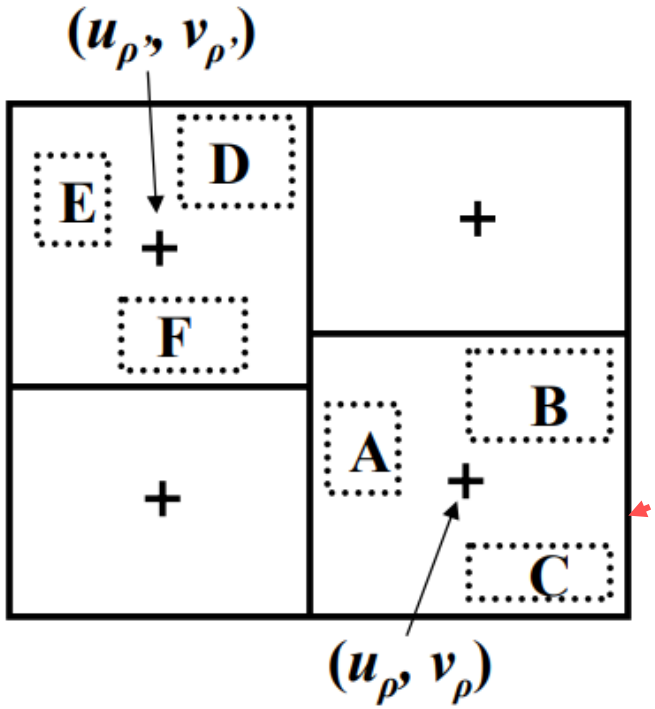



  - Lastly, we have

$$A^l X = u^l, \text{where } a_{pm} = \begin{cases} F_m / \sum_{m \in M_p} F_m, & \text{if } m \in M_p \\ 0 & \text{otherwise} \end{cases}$$

# Problem Formulation

# Row = # partitions
# Column = # of cells




$$A^{(l)} = \begin{array}{c} \\ \vdots \\ \rho \\ \rho' \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & * & * & * & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \end{array}$$

**Linearly constrained Quadratic Programming problem**

$$\text{LQP}: \min_{x \in R^m} \{\Phi(x) = X^T C X + d^T X \text{ such that } A^l X = u^l\}$$

The placement solution for a circuit with fixed I/O pins at boundary when quadratic wirelength is minimized and nonoverlapping constraints are ignored

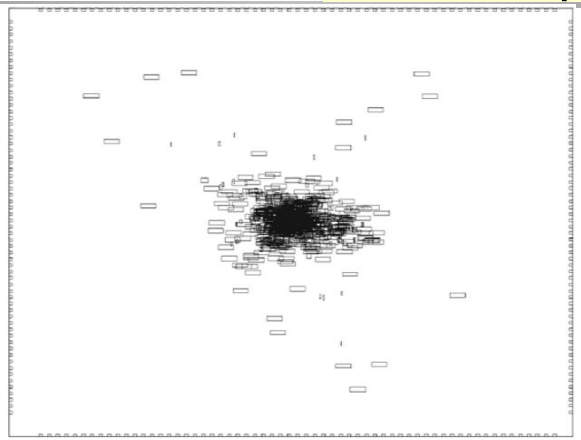


FIGURE 11.17

**GORDIAN: VLSI placement by quadratic programming and slicing optimization**

**Center of mass constraints are added.**

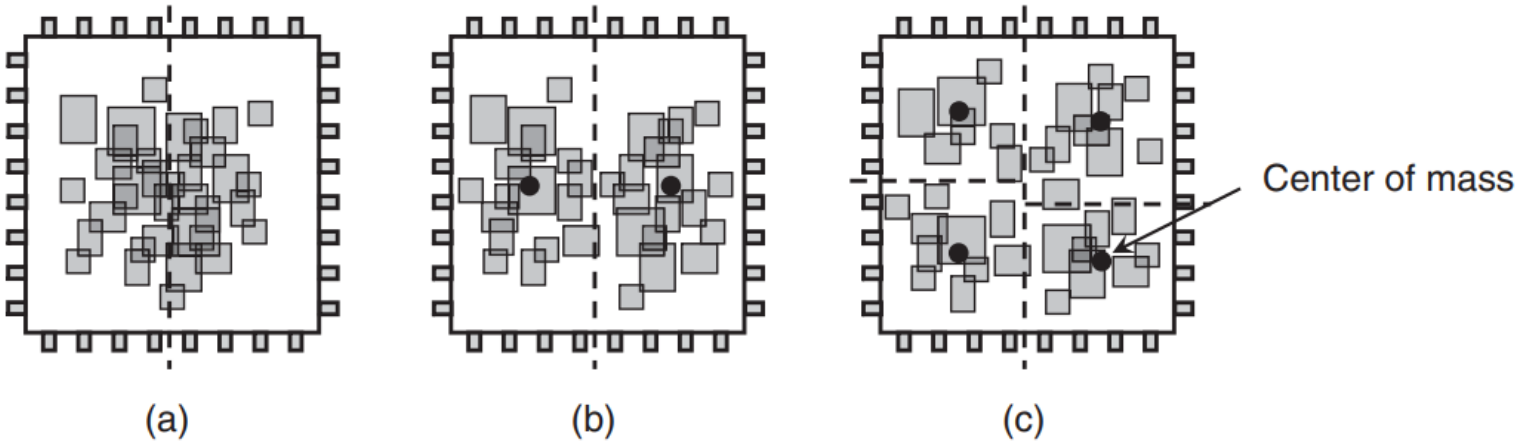



**FIGURE 11.18**

Module spreading by center-of-mass constraints in GORDIAN.

# Adaptec1 Stats

- **Circuit stats**
  - **# cells/nets/pins** — **210,863/219,687/19,205**
  - **chip size** — **6000um × 6000um**
  - **bin size** — **50um × 50um**
  - **# placement bins** — **120 × 120**
  - **Average bin occupancy** — **210K/120$^2$ =14.6 gates/bin**
- **Wirelength result (HPBB)**
  - **iteration 0** — **34,069,060**
  - **iteration 29** — **46,352,680**
  - **iteration 58** — **80,783,336**
  - **iteration 87** — **98,111,904**

# Experimental Results

-

**Comparison of Results for Standard Cell Blocks**

| | Area After Routing/mm$^2$ | | |
|---|---|---|---|
| Circuit | GORDIAN | Min-Cut | Annealing |
| scb1 | 2.7 | 3.1 | 2.6 |
| scb2 | 5.8 | 5.3 | 5.0 |
| scb3 | 15.7 | 25.6 | 9.1 |
| scb4 | 14.0 | 16.9 | 13.2 |
| scb5 | 10.6 | 11.3 | 10.9 |
| scb6 | 11.3 | 12.7 | 12.8 |
| scb7 | 16.4 | 20.2 | 19.8 |
| scb8 | 51.7 | 89.2 | 59.5 |
| scb9 | 54.0 | 98.6 | 80.0 |
| CPU-time scb8 | 120s | 366s | 39851s |
| CPU-time scb9 | 135s | 440s | 34709s |
| ratio | 1 | :3 | :300 |

https://dl.acm.org/doi/pdf/10.1145/266021.266362

Quadratic Placement Revisited (DAC 1997)

GitHub - KangliC/Gordian_Placement: Partition and Placement via Gordian Algorithm
KangliC/Gordian_Placement

https://www.netlib.org/lapack/

LAPACK — Linear Algebra PACKage

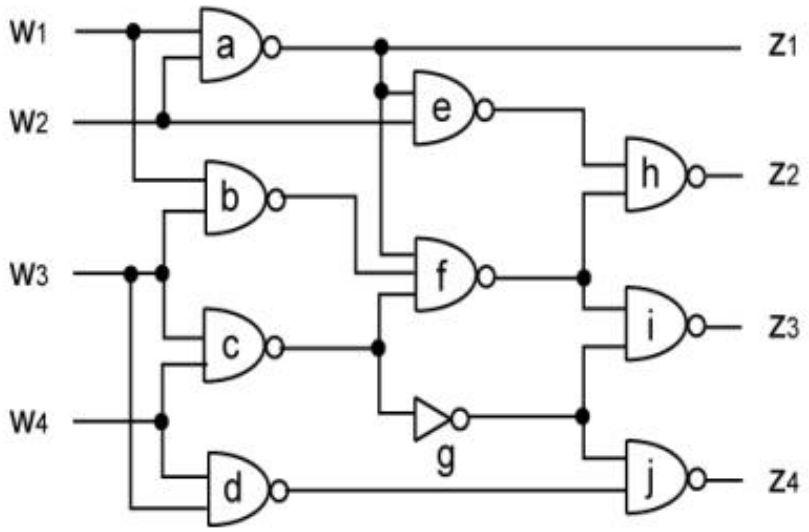
https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

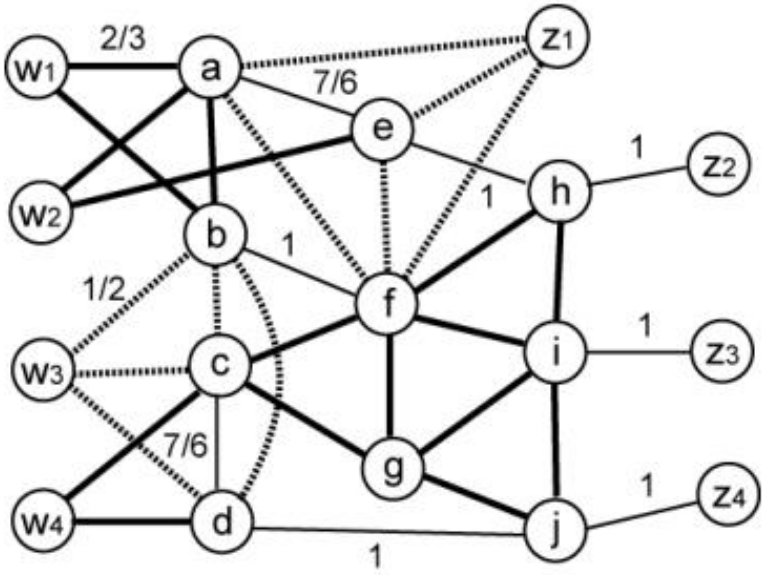
Basic Linear Algebra Subprograms

https://www.netlib.org/blas/

# GORDIAN Placement

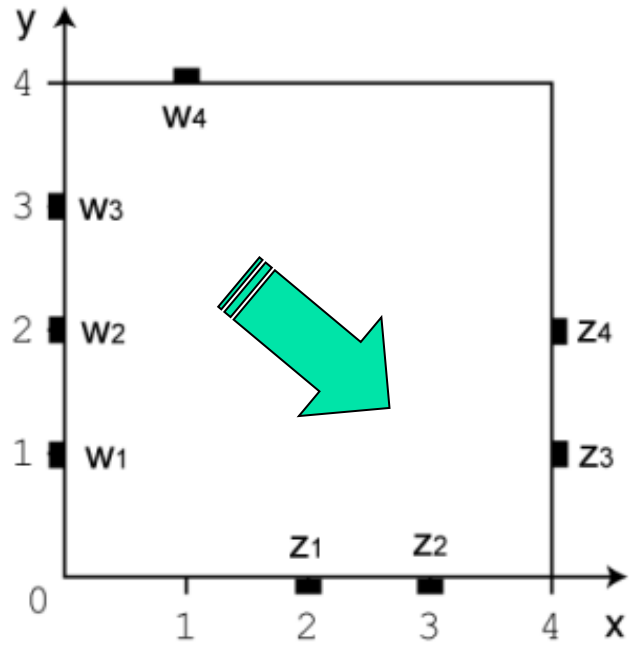
- Perform GORDIAN placement
  - Uniform area and net weight, area balance factor = 0.5
  - Undirected graph model: each edge in $k$-clique gets weight $2/k$



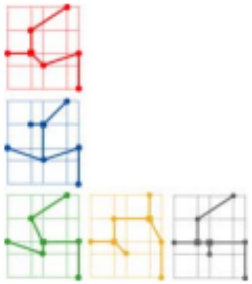



# IO Placement

- Necessary for GORDIAN to work




In some sense there is a dataflow concept



# Adjacency Matrix

- Shows connections among movable nodes
  - Among nodes *a* to *j*

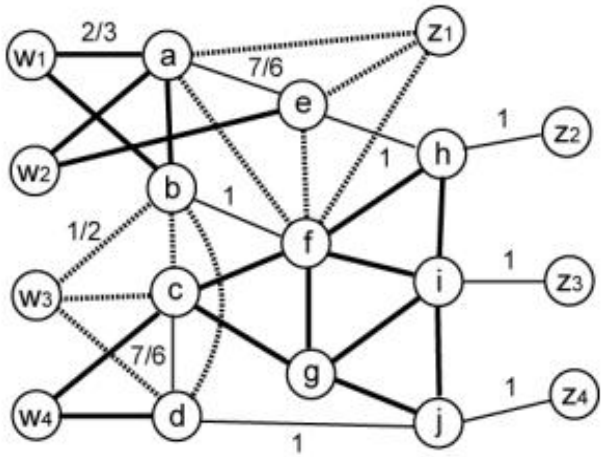
10x10 (**10 movable** modules)

b e f

$$\begin{pmatrix}
0 & \frac{2}{3} & 0 & 0 & \frac{7}{6} & \frac{1}{2} & 0 & 0 & 0 & 0 \\
\frac{2}{3} & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2} & 0 & \frac{7}{6} & 0 & \frac{2}{3} & \frac{2}{3} & 0 & 0 & 0 \\
0 & \frac{1}{2} & \frac{7}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\frac{7}{6} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 1 & 0 & 0 \\
\frac{1}{2} & 1 & \frac{2}{3} & 0 & \frac{1}{2} & 0 & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & 0 \\
0 & 0 & \frac{2}{3} & 0 & 0 & \frac{2}{3} & 0 & 0 & \frac{2}{3} & \frac{2}{3} \\
0 & 0 & 0 & 0 & 1 & \frac{2}{3} & 0 & 0 & \frac{2}{3} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{2}{3} & \frac{2}{3} & \frac{2}{3} & 0 & \frac{2}{3} \\
0 & 0 & 0 & 1 & 0 & 0 & \frac{2}{3} & 0 & \frac{2}{3} & 0
\end{pmatrix}$$

14/6

7/3



$$Q = \begin{pmatrix} c_{12}+c_{13}+c_{14} & 0 & 0 \\ 0 & c_{21}+c_{24}+c_{25} & 0 \\ 0 & 0 & c_{31}+c_{36} \end{pmatrix} - \begin{pmatrix} 0 & c_{12} & c_{13} \\ c_{21} & 0 & c_{23} \\ c_{31} & c_{32} & 0 \end{pmatrix}$$

$$d_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix} \text{ and } d_y = \begin{pmatrix} -c_{14}y_4 \\ -c_{24}y_4 - c_{25}y_5 \\ -c_{36}y_6 \end{pmatrix}$$

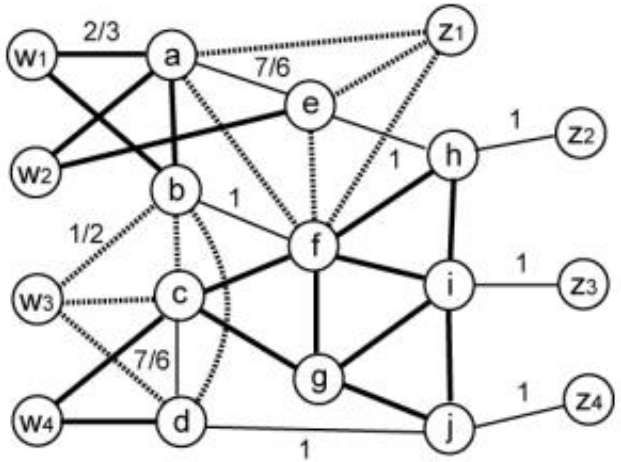
# Pin Connection Matrix

- Shows connections between movable nodes and IO
  - Rows = movable nodes, columns = IO (fixed) **10x8**

w1 w2 z1

$$\begin{pmatrix} \frac{2}{3} & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

11/6



# Degree Matrix

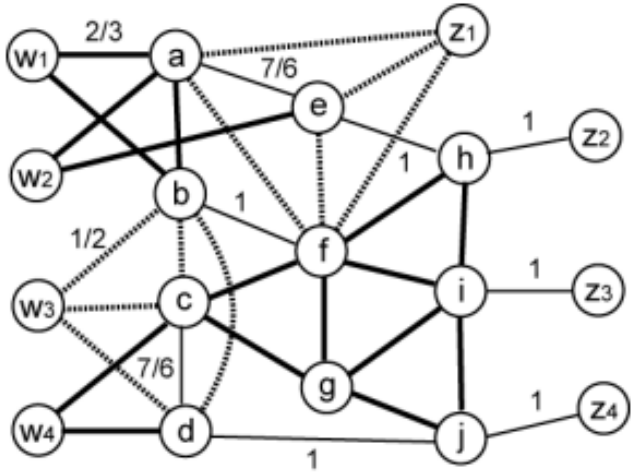
$$Q = \begin{pmatrix} c_{12}+c_{13}+c_{14} & 0 & 0 \\ 0 & c_{21}+c_{24}+c_{25} & 0 \\ 0 & 0 & c_{31}+c_{36} \end{pmatrix} - \begin{pmatrix} 0 & c_{12} & c_{13} \\ c_{21} & 0 & c_{23} \\ c_{31} & c_{32} & 0 \end{pmatrix}$$

$$d_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix} \text{ and } d_y = \begin{pmatrix} -c_{14}y_4 \\ -c_{24}y_4 - c_{25}y_5 \\ -c_{36}y_6 \end{pmatrix}$$

- Based on both adjacency and pin connection matrices
  - Sum of entries in the same row (= node degree) **10x10**

14/6 11/6

$$\begin{pmatrix} \frac{25}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{23}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{25}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{23}{6} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{23}{6} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{31}{6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{8}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{10}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{11}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{10}{3} \end{pmatrix}$$




7/3 1

$$Q = \begin{pmatrix} c_{12}+c_{13}+c_{14} & 0 & 0 \\ 0 & c_{21}+c_{24}+c_{25} & 0 \\ 0 & 0 & c_{31}+c_{36} \end{pmatrix} - \begin{pmatrix} 0 & c_{12} & c_{13} \\ c_{21} & 0 & c_{23} \\ c_{31} & c_{32} & 0 \end{pmatrix}$$

# Laplacian Matrix

$$d_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix} \text{ and } d_y = \begin{pmatrix} -c_{14}y_4 \\ -c_{24}y_4 - c_{25}y_5 \\ -c_{36}y_6 \end{pmatrix}$$

- Degree matrix minus adjacency matrix **10x10**

b e f

$$\begin{pmatrix} \frac{25}{6} & -\frac{2}{3} & 0 & 0 & -\frac{7}{6} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ -\frac{2}{3} & \frac{23}{6} & -\frac{1}{2} & -\frac{1}{2} & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & \frac{25}{6} & -\frac{7}{6} & 0 & -\frac{2}{3} & -\frac{2}{3} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & -\frac{7}{6} & \frac{23}{6} & 0 & 0 & 0 & 0 & 0 & -1 \\ -\frac{7}{6} & 0 & 0 & 0 & \frac{23}{6} & -\frac{1}{2} & 0 & -1 & 0 & 0 \\ -\frac{1}{2} & -1 & -\frac{2}{3} & 0 & -\frac{1}{2} & \frac{31}{6} & -\frac{2}{3} & -\frac{2}{3} & -\frac{2}{3} & 0 \\ 0 & 0 & -\frac{2}{3} & 0 & 0 & -\frac{2}{3} & \frac{8}{3} & 0 & -\frac{2}{3} & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -1 & -\frac{2}{3} & 0 & \frac{10}{3} & -\frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{2}{3} & -\frac{2}{3} & -\frac{2}{3} & \frac{11}{3} & -\frac{2}{3} \\ 0 & 0 & 0 & -1 & 0 & 0 & -\frac{2}{3} & 0 & -\frac{2}{3} & \frac{10}{3} \end{pmatrix}$$

# Fixed Pin Vectors

$$d_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix}$$

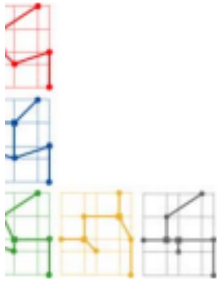
- Based on pin connection matrix and IO location

  Each entry $i$ in $d_x$, denoted $d_{x,i}$, is computed as follows:

$$d_{x,i} = -\sum_j p_{ij} \cdot x(p_j)$$

  where $p_{ij}$ denotes the entry of the pin connection matrix, and $x(p_j)$ is the $x$-coordinate of the corresponding IO pin $j$.

  - Y-direction is defined similarly



# Fixed Pin Vectors (cont)

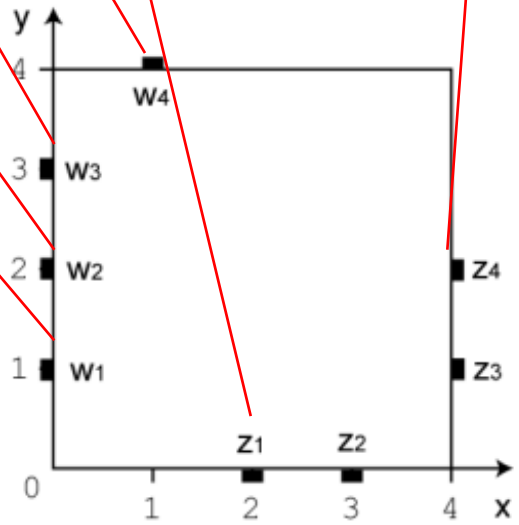
$$d_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix}$$

$$d_{x,1} = -(\frac{2}{3}\cdot 0 + \frac{2}{3}\cdot 0 + 0\cdot 0 + 0\cdot 1 + \frac{1}{2}\cdot 2 + 0\cdot 3 + 0\cdot 4 + 0\cdot 4) = -1$$

By examining the remaining 9 movable cells, we get

$$d_x^T = \begin{pmatrix} -1 & 0 & -\frac{2}{3} & -\frac{2}{3} & -1 & -1 & 0 & -3 & -4 & -4 \end{pmatrix}$$

$$\begin{pmatrix} \frac{2}{3} & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$




**10x8**

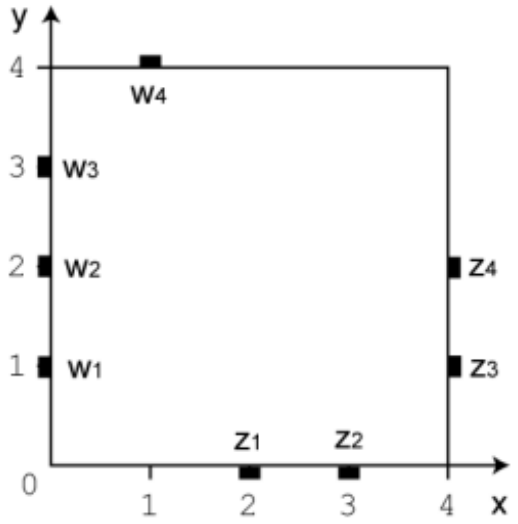
# Fixed Pin Vectors (cont)

$$d_{y,1} = -(\frac{2}{3} \cdot 1 + \frac{2}{3} \cdot 2 + 0 \cdot 3 + 0 \cdot 4 + \frac{1}{2} \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 2) = -2$$

By examining the remaining 9 movable cells, we get

$$d_y^T = \begin{pmatrix} -2 & -\frac{13}{6} & -\frac{25}{6} & -\frac{25}{6} & -\frac{4}{3} & 0 & 0 & 0 & -1 & -2 \end{pmatrix}$$

$$\begin{pmatrix} \frac{2}{3} & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{2}{3} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



# Level 0 QP Formulation

- No constraint necessary

Minimize

$$\phi(x) = \frac{1}{2}x^T Cx + {d_x}^T x$$

and

$$\phi(y) = \frac{1}{2}y^T Cy + {d_y}^T y$$

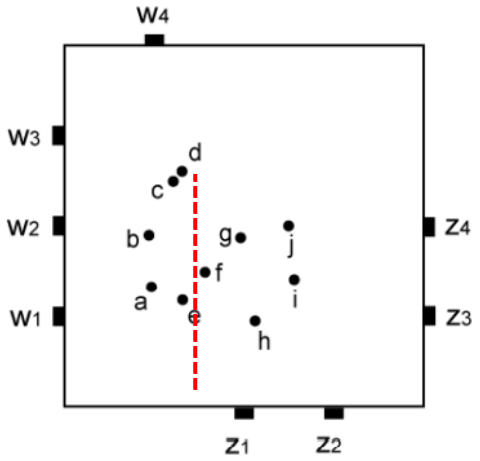
We use MOSEK and obtain the following solution:

$$x^T = (0.95 \quad 0.92 \quad 1.21 \quad 1.32 \quad 1.32 \quad 1.61 \quad 1.98 \quad 2.13 \quad 2.59 \quad 2.51)$$

$$y^T = (1.27 \quad 1.83 \quad 2.48 \quad 2.61 \quad 1.16 \quad 1.45 \quad 1.84 \quad 0.92 \quad 1.41 \quad 2.03)$$

## Level 0 Placement

- Cells with real dimension will overlap



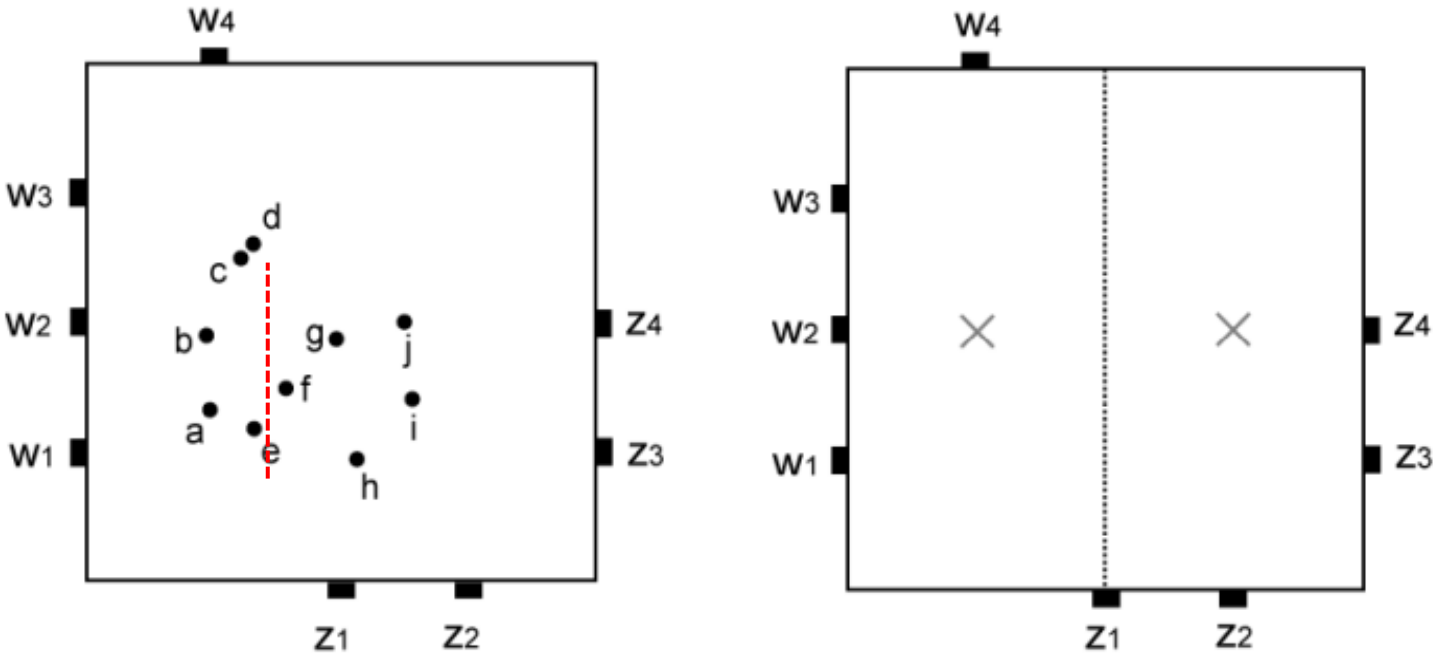
# MOSEK

Our solver is the solution for all your LPs, QPs, SOCPs, SDPs and MIPs. Includes interfaces to C, C++, Java, MATLAB, .NET, Python and R.

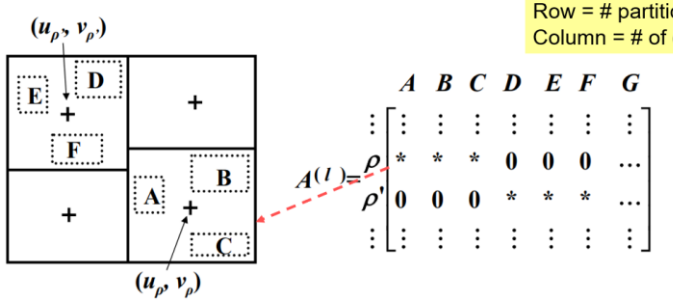
$$C := \{x \in \mathrm{R}^3 : x_1 \geq \sqrt{x_2^2 + x_3^2}\}$$

# Level 1 Partitioning

- Perform level 1 partitioning
  - Obtain center locations for center-of-gravity constraints



# Level 1 Constraint



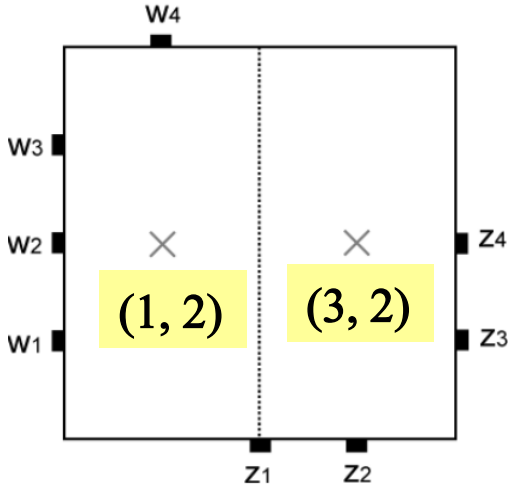

We first sort the nodes based on their $x$-coordinates.

$$\{b, a, c, e, d, f, g, h, j, i\}$$

We perform partitioning under $\alpha = 0.5$:

Not K-L min cut(?)

$$S_{\rho'} = \{b, a, c, e, d\},\ S_{\rho''} = \{f, g, h, j, i\}$$




The center location vectors are:

$$u_x^{(1)} = \begin{pmatrix} 1 \\ 3 \end{pmatrix},\ u_y^{(1)} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

We build the matrix $A^{(1)}$ for the center-of-gravity constraint at level $l = 1$:

$$A^{(1)} = \begin{pmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{pmatrix}$$

# Level 1 LQP Formulation

We now solve the following Linearly constrained QP (LQP) to obtain the new placement for the movable nodes:

$$\text{Minimize } \phi(x) = \frac{1}{2}x^T Cx + {d_x}^T x, \text{ subject to } A^{(1)} \cdot x = u_x^{(1)}$$

$$\text{Minimize } \phi(y) = \frac{1}{2}y^T Cy + {d_y}^T y, \text{ subject to } A^{(1)} \cdot y = u_y^{(1)}$$

The solutions are as follows:

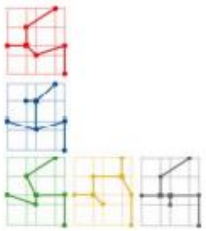
f g

$$x^T = \begin{pmatrix} 0.70 & 0.71 & 1.17 & 1.21 & 1.22 & 2.17 & 3.10 & 2.84 & 3.56 & 3.33 \end{pmatrix}$$

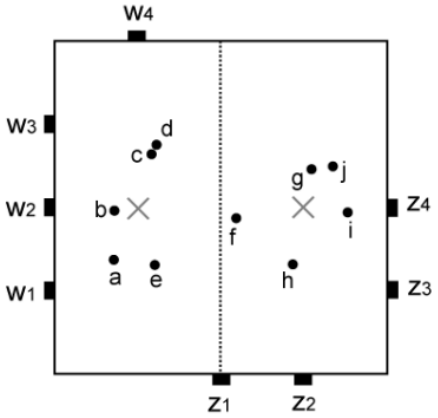
$$y^T = \begin{pmatrix} 1.34 & 1.94 & 2.66 & 2.76 & 1.30 & 1.83 & 2.45 & 1.32 & 1.91 & 2.49 \end{pmatrix}$$

We can see **e, f** are more apart

## Level 1 Placement







# Verification

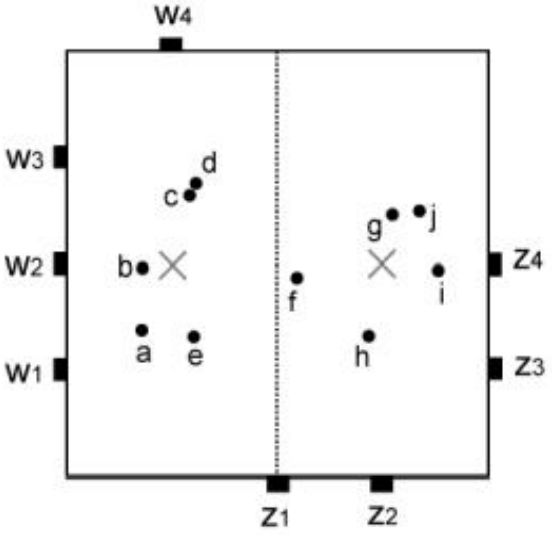
- Verify that the constraints are satisfied in the left partition

The following cells are partitioned to the left: $a(0.70, 1.34)$, $b(0.71, 1.94)$, $c(1.17, 2.66)$, $d(1.21, 2.76)$, and $e(1.22, 1.30)$. Thus, the center of gravity is located at:

$$\frac{0.70 + 0.71 + 1.17 + 1.21 + 1.22}{5} = 1.00$$

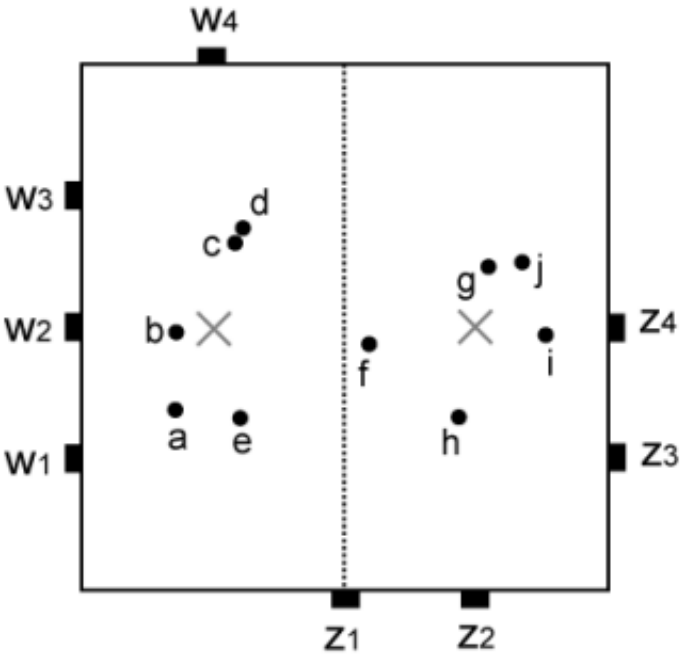
$$\frac{1.34 + 1.94 + 2.66 + 2.76 + 1.30}{5} = 2.00$$

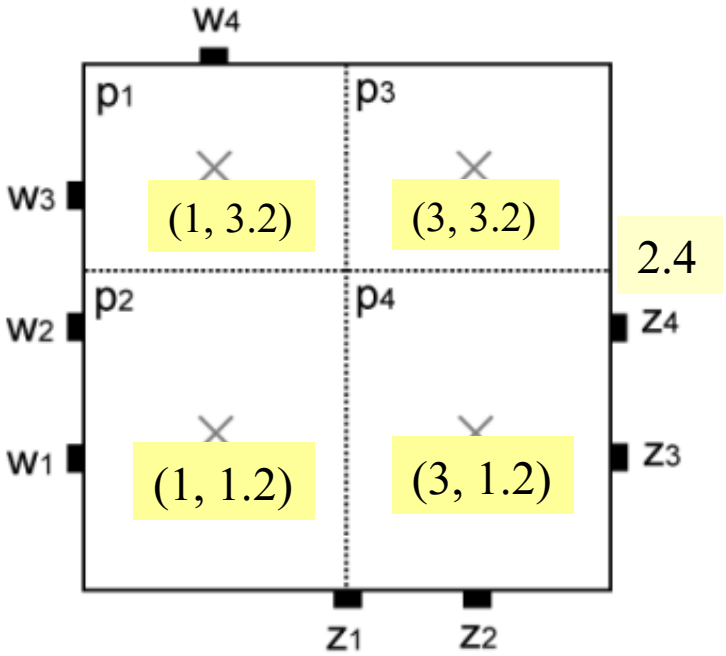
This agrees with the center location $(1, 2)$.



# Level 2 Partitioning

- Add two more cut-lines
  - This results in $p_1=\{c,d\}$, $p_2=\{a,b,e\}$, $p_3=\{g,j\}$, $p_4=\{f,h,i\}$



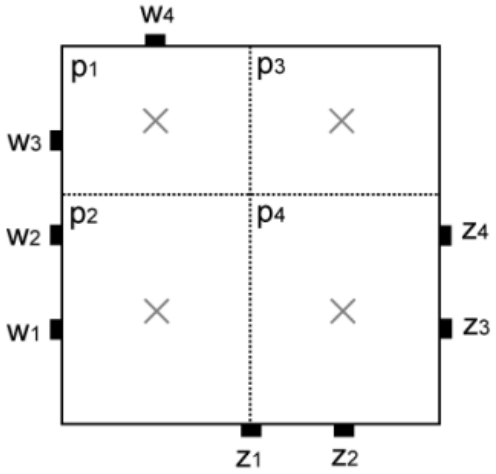

4*3/5/2=1.2

chip height is 4
we split 5 cells into 2:3 ratio

# Level 2 Constraint

The center location vectors are:

$$u_x^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 3 \end{pmatrix}, \quad u_y^{(2)} = \begin{pmatrix} 3.2 \\ 1.2 \\ 3.2 \\ 1.2 \end{pmatrix}$$




Next, we build the matrix $A^{(2)}$ for the center-of-gravity constraint at level $l = 2$. Recall that $p_1 = \{c, d\}$, $p_2 = \{a, b, e\}$, $p_3 = \{g, j\}$, $p_4 = \{f, h, i\}$. Thus,

$$A^{(2)} = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix}$$

where the rows denote the partitions $p_1$ through $p_4$, and the columns denote the cells $a$ through $j$.

# Level 2 LQP Formulation

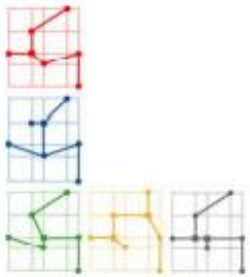
We now solve the following LQP to obtain the placement of the movable nodes:

$$\text{Minimize } \phi(x) = \frac{1}{2}x^T Cx + {d_x}^T x, \text{ subject to } A^{(2)} \cdot x = u_x^{(2)}$$

$$\text{Minimize } \phi(y) = \frac{1}{2}y^T Cy + {d_y}^T y, \text{ subject to } A^{(2)} \cdot y = u_y^{(2)}$$

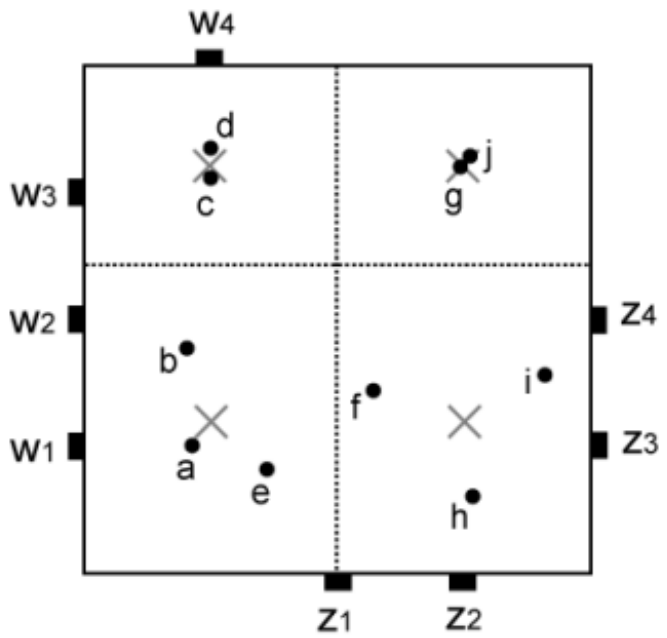
The solutions are as follows:

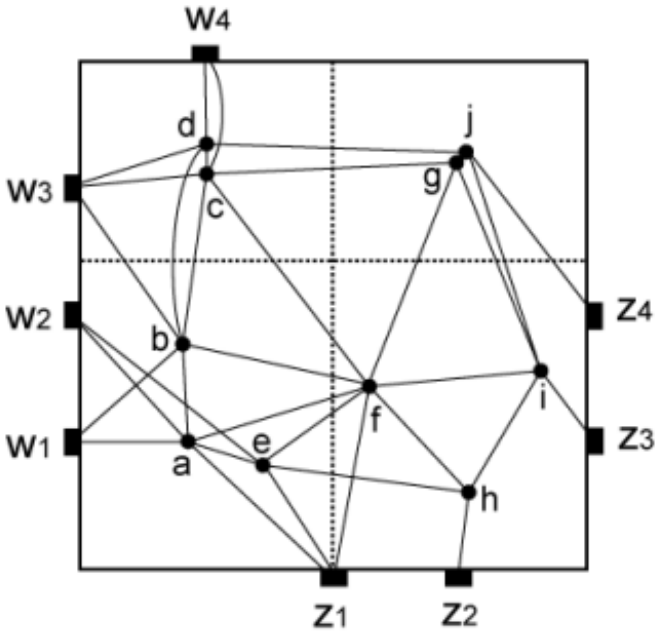
$$x^T = \begin{pmatrix} 0.83 & 0.78 & 1.00 & 1.00 & 1.39 & 2.28 & 2.89 & 3.06 & 3.66 & 3.11 \end{pmatrix}$$
$$y^T = \begin{pmatrix} 1.01 & 1.78 & 3.08 & 3.32 & 0.82 & 1.44 & 3.18 & 0.59 & 1.57 & 3.22 \end{pmatrix}$$



# Level 2 Placement

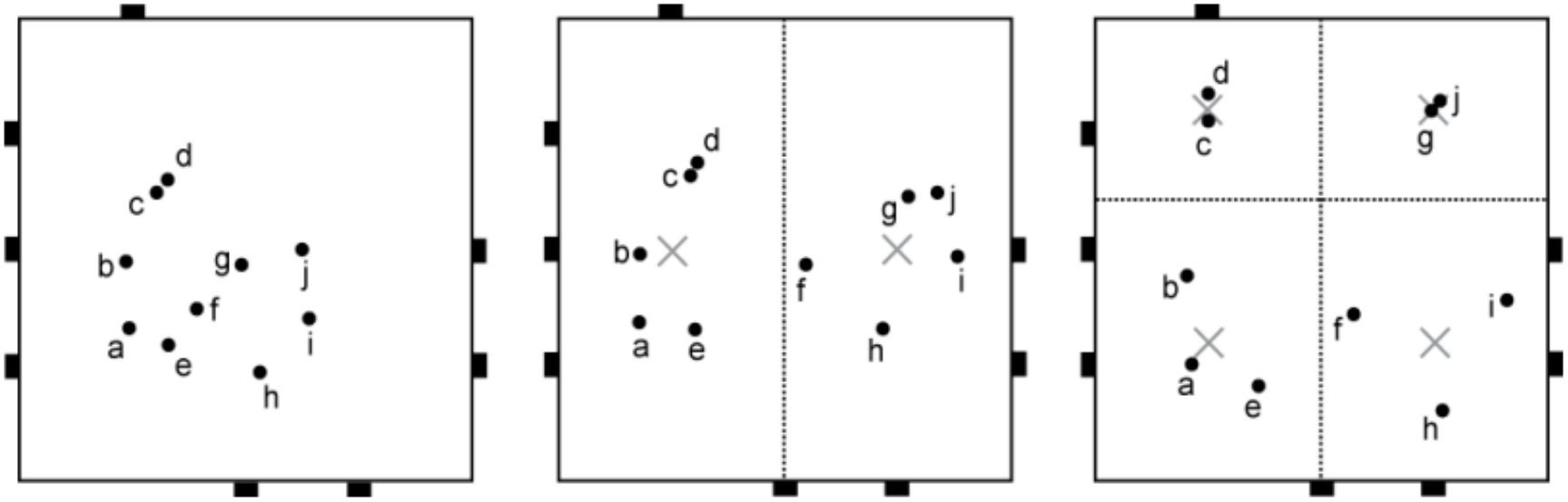
- Clique-based wiring is shown



# Summary

- Center-of-gravity constraint
  - Helps spread the cells evenly while monitoring wirelength
  - Removes overlaps among the cells (with real dimension)



# Partitioning

Other method is thru density

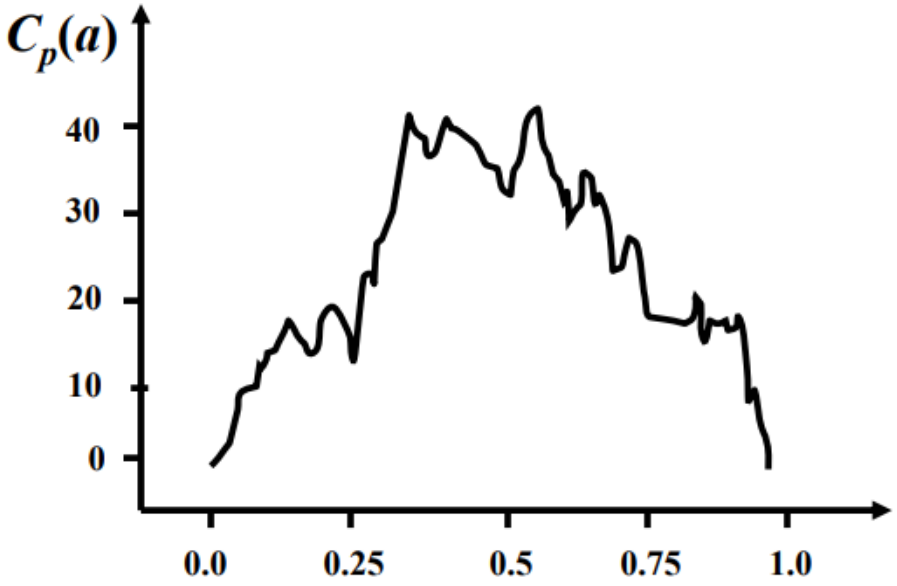
- **Recursive partitioning is needed**
  - **to resolve module overlap in global placement**
  - **global placement problem will be solved again with two additional center_of_gravity constraints**

$$M_p \rightarrow (M_{p'}, M_{p''})$$

$$x_{u'} \leq x_{u''} \; u' \in M_{p'} \text{ and } u'' \in M_{p''}$$

$$\alpha = \sum_{u' \in M_{p'}} F_u \Big/ \sum_{u \in M_p} F_u \approx 0.5$$

$$\text{cut value}: C_p(\alpha) = \sum_{v \in N_C} w_v$$



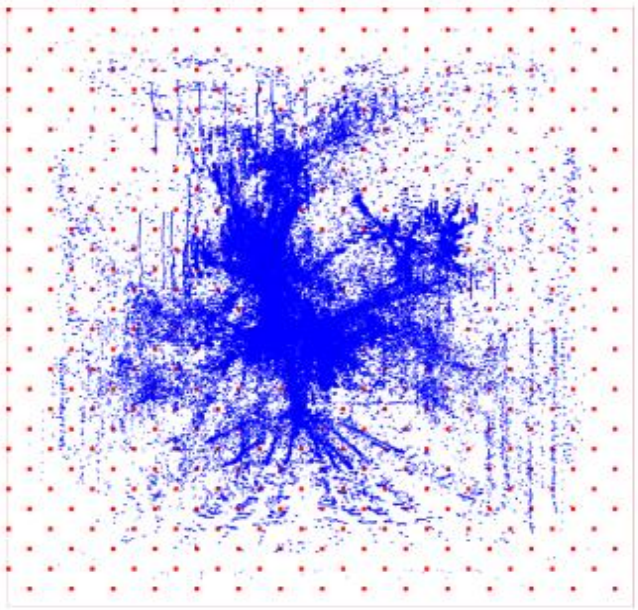
# Repartitioning

- **Module exchange after each cut to improve cut size**
  - **terminal propagation using global placement positions**
- **Repartitioning**
  - **to 'undo' the mistake made at the previous level:**

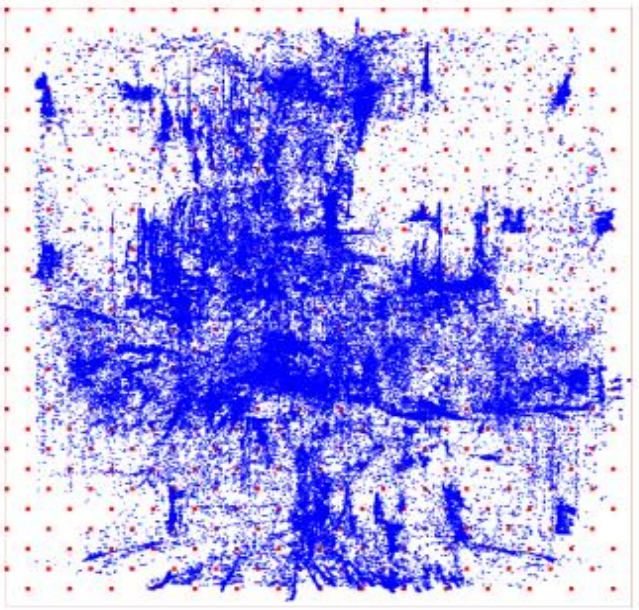
```
Procedure repartition(l)
    if overlap exists
        for each r∈R(l-1)
            merge-regions(r, r', r'');
            partition(r, r', r'');
        setup-constraints(l);
        global-optimize(l);
    endif
```

Please refer to gordian.pdf

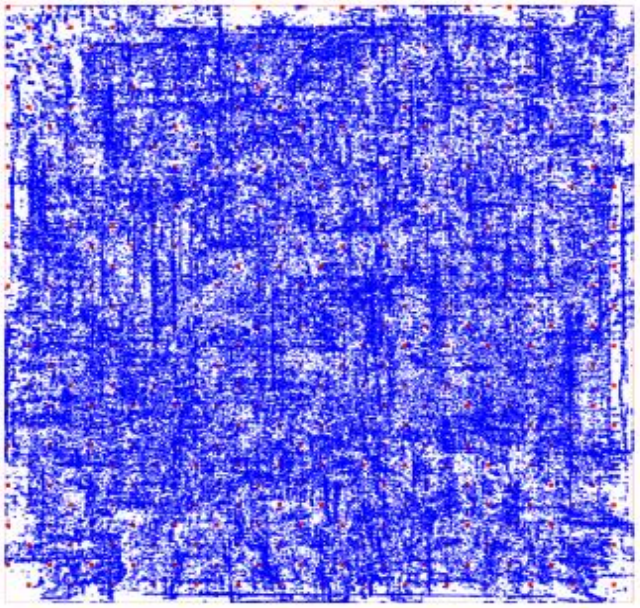
i=0



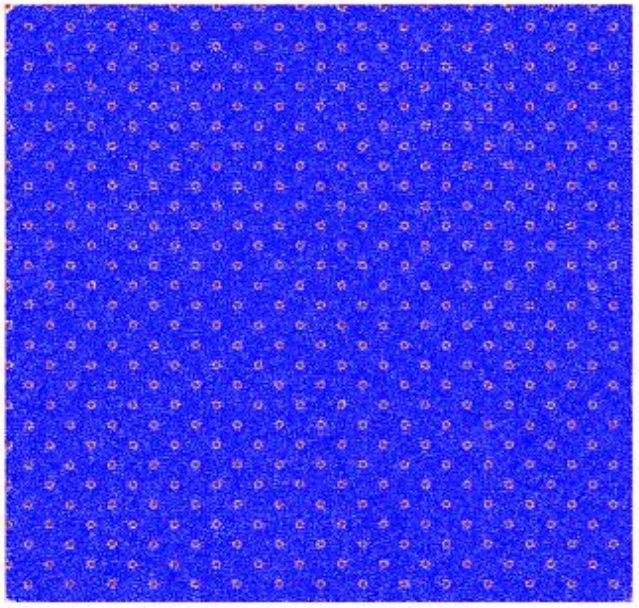
i=29



i=58



i=87



# Experimental Results

**Comparison of Results for Standard Cell Blocks**

| | Area After Routing/mm$^2$ | | |
|---|---|---|---|
| Circuit | GORDIAN | Min-Cut | Annealing |
| scb1 | 2.7 | 3.1 | 2.6 |
| scb2 | 5.8 | 5.3 | 5.0 |
| scb3 | 15.7 | 25.6 | 9.1 |
| scb4 | 14.0 | 16.9 | 13.2 |
| scb5 | 10.6 | 11.3 | 10.9 |
| scb6 | 11.3 | 12.7 | 12.8 |
| scb7 | 16.4 | 20.2 | 19.8 |
| scb8 | 51.7 | 89.2 | 59.5 |
| scb9 | 54.0 | 98.6 | 80.0 |
| CPU-time scb8 | 120s | 366s | 39851s |
| CPU-time scb9 | 135s | 440s | 34709s |
| ratio | 1 | :3 | :300 |

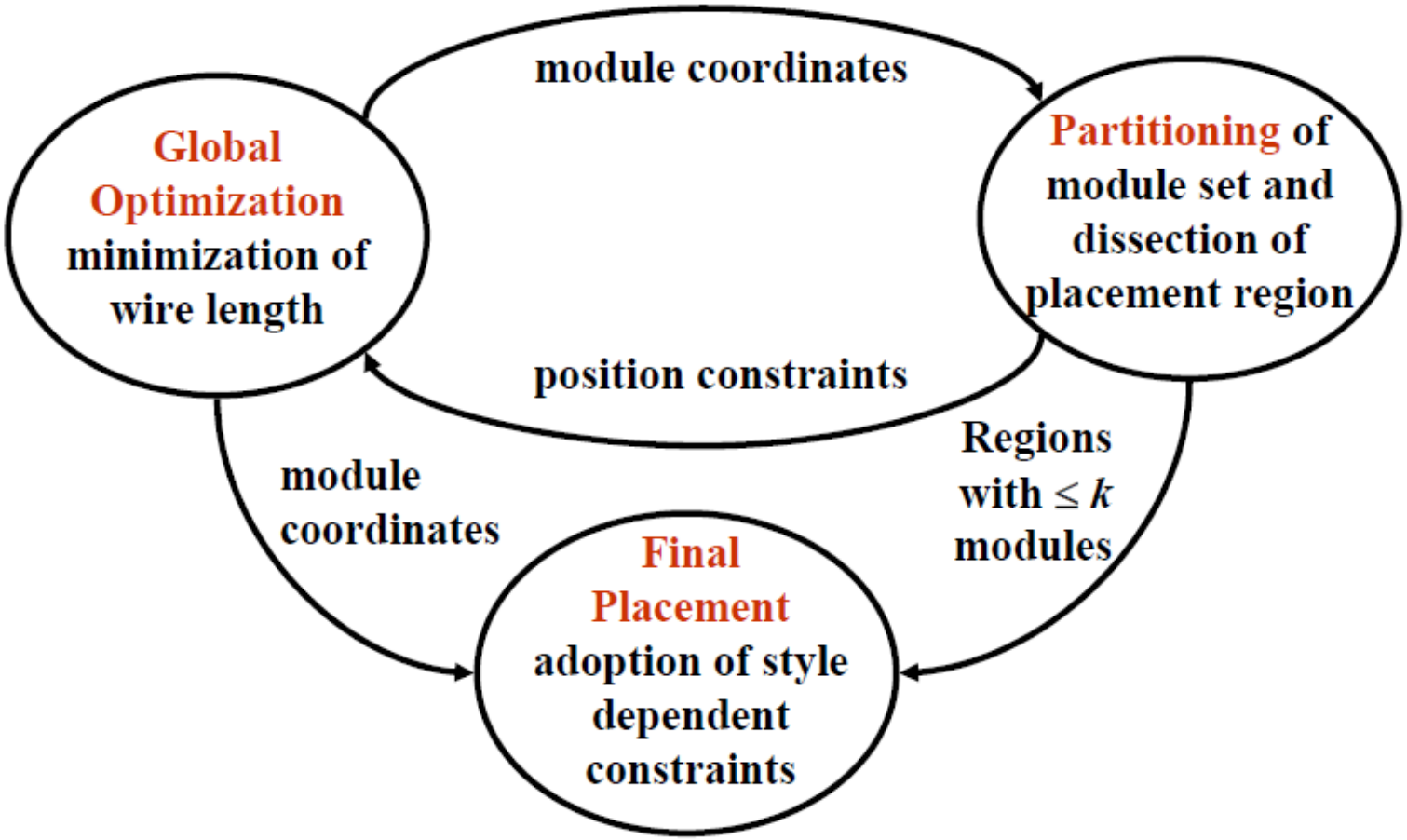
placement_Sung_Kyu_Lim.pdf
(from page 32)

# Overview of Gordian Package

```
Procedure Gordian
    l:=1;
    global-optimize(l);
    while (there exists |M_l|>k)
        for each r ϵ R(l)
            partition(r, r', r");
        l++;
        setup-constraints(l);
        global-optimize(l);
        repartition(l);
    final-placement(l);
endprocedure
```

(a region has more cells – need partition)

# Summary of Gordian




**Complexity:** space = $O(m)$, time = $O(m^{1.5} \log^2 m)$

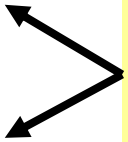
**Final placement:** standard cell, macro-cell & SOG

# Gordian

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.2671&rep=rep1&type=pdf
Quadratic Placement Revisited

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.226.8246&rep=rep1&type=pdf
GORDIAN

https://github.com/KangliC/Gordian_Placement/blob/master/Output%20plots.pdf
A project report using Gordian

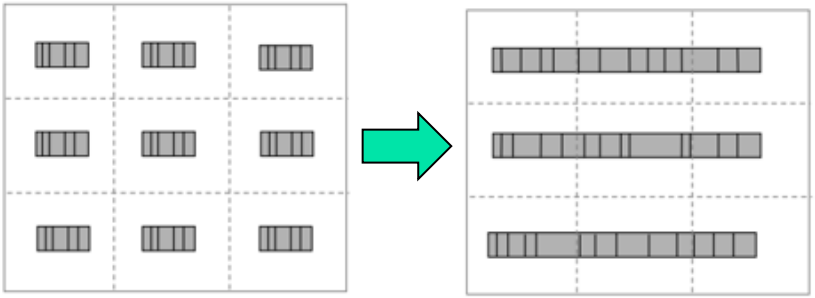


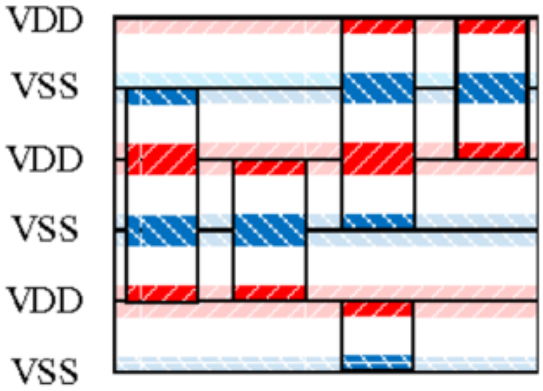
Please bring up Gordian and explain its major modules

https://github.com/ZhenshenQiu/Gordian-VLSI-Placement

# Detailed Placement

- After global placement, need to resolve overlapping to get legal placement locations



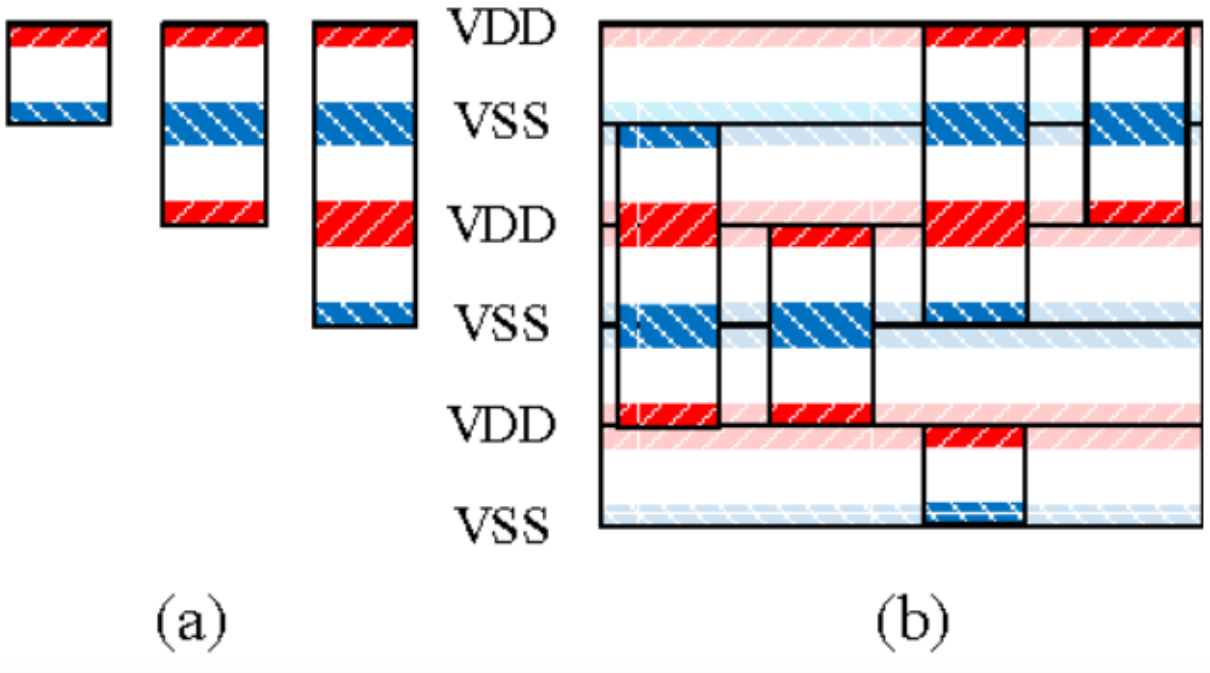



Now we heard 1.5x, 2.5x cell heights

- ECO placement – when there are some gate changes, how to get a placement with minimum displacement?

https://www.semanticscholar.org/paper/Mixed-Cell-Height-Standard-Cell-Placement-Hung-Chou/2fcd5d1503e86a5eae628e1447e5fd8aca25c6ae

# Nowadays Cell Heights Vary




Now we even heard 1.5x, 2.5x cell heights

Figure 1: (a) Standard cells with multiple-row height. (b) Some examples of power-rail alignment with multiple-row-height cells.

# FinFet Coloring Rule Impacting Cell Placement

It is possible, but the current cell row placement rule is defined in a more general way.
LEF can define standard cell edge spacing rules, and therefore foundry can define a minimum spacing between two cell edges.
I found an example in https://iccad-contest.org/2017/Problem_C/default.html

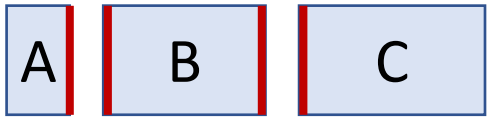
```
PROPERTYDEFINITIONS
   MACRO LEF58_EDGETYPE STRING ;
   LIBRARY LEF58_CELLEDGESPACINGTABLE STRING
"CELLEDGESPACINGTABLE
   EDGETYPE 1 2 0.400
   EDGETYPE 1 1 0.400
   EDGETYPE 2 2 0.000
;" ;
END PROPERTYDEFINITIONS
```

For each cell, two properties are defined for the left and right edges in file "cells_modified.lef". For instance, the following statement is the properties defined for **MACRO ms00f80** of design "fft_2_md2":

```
   PROPERTY LEF58_EDGETYPE "
      EDGETYPE LEFT 2 ;
      EDGETYPE RIGHT 2 ;
" ;
```

So if there is the situation you mentioned, the tech file will include **edgetype** and **celledgespacingtable** to have

1. A_RIGHT to B_LEFT  a_non_zero_spacing
2. B_RIGHT to C_LEFT  a_non_zero_spacing



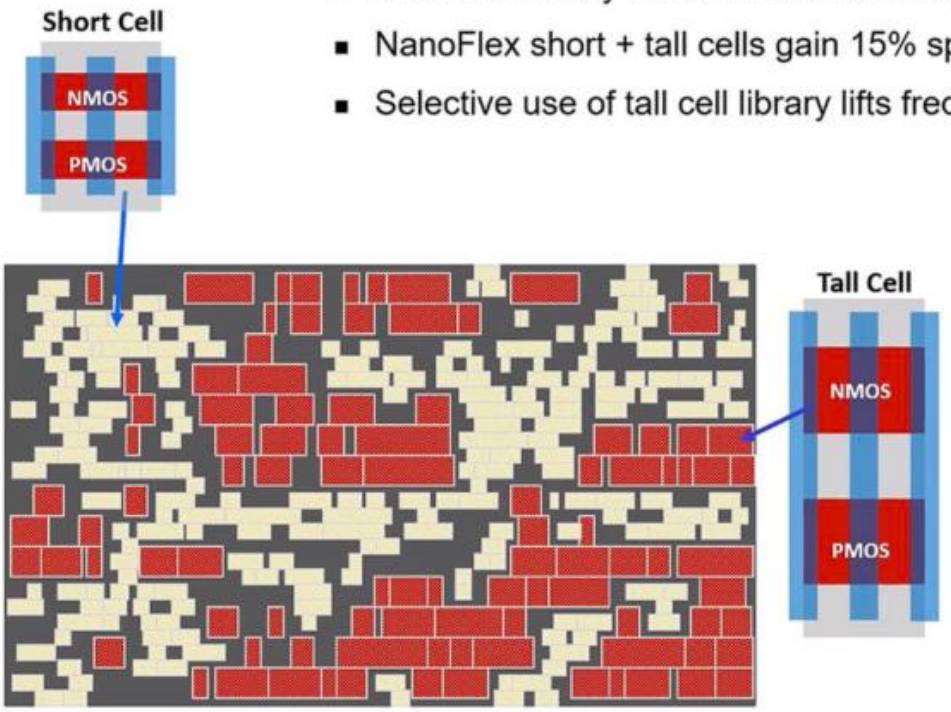

During legalization, the spacing rule needs to be followed to prevent any design rule violations.

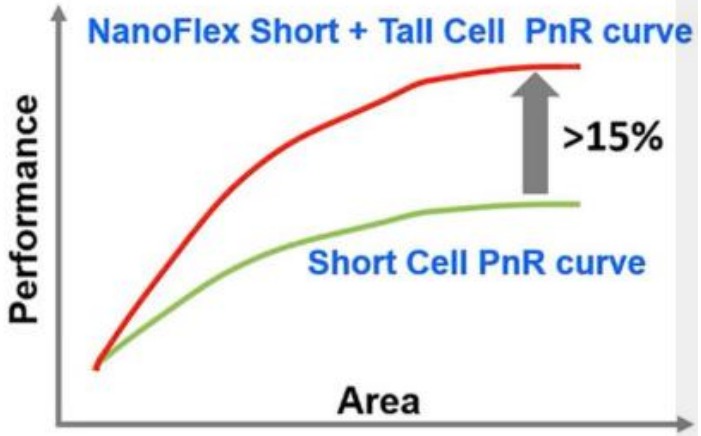
Tung-Chieh Chen

# NanoFlex™ Innovation Maximizes Nanosheet Values

## NanoFlex maximizes nanosheet width modulation for best PPACt

- Short cell library establishes area and power efficiency
- NanoFlex short + tall cells gain 15% speed at same area vs. short cells alone
- Selective use of tall cell library lifts frequency to meet design target




Source: TSMC IEDM 2024 (https://weekly.ascii.jp/elem/000/004/250/4250448/)

# Other Good Works

Prof. Chris Chu – FastPlace 2.0

https://www.aspdac.com/aspdac2006/archives/pdf/2C-2.pdf

And they published FastPlace 3.0 (in 2007)

https://dl.acm.org/doi/10.1109/ASPDAC.2007.357975

RePlace

https://vlsicad.ucsd.edu/Publications/Journals/j126.pdf

RePlAce: Advancing Solution Quality and Routability Validation in Global Placement
Chung-Kuan Cheng, Fellow, IEEE, Andrew B. Kahng, Fellow, IEEE, Ilgweon Kang, Member, IEEE, and Lutong Wang, Student Member, IEEE

https://github.com/The-OpenROAD-Project/RePlAce

(Let's try to bring up this program)

# Global Placement in FastPlace

- **Framework:**

  *repeat*

  *Solve the convex quadratic program* ❶
  *Reduce wirelength by iterative heuristic* ❷
  *Spread the cells* ❸

  *until the cells are evenly distributed* ❹

- **Special features of FastPlace:**
  - **Hybrid Net Model**
    - Speed up solving of convex QP ❶
  - **Cell Shifting**
    - Easy-to-compute technique ❸
    - Enable fast convergence ❹
  - **Iterative Local Refinement**
    - Minimize wirelength based on linear objective ❷

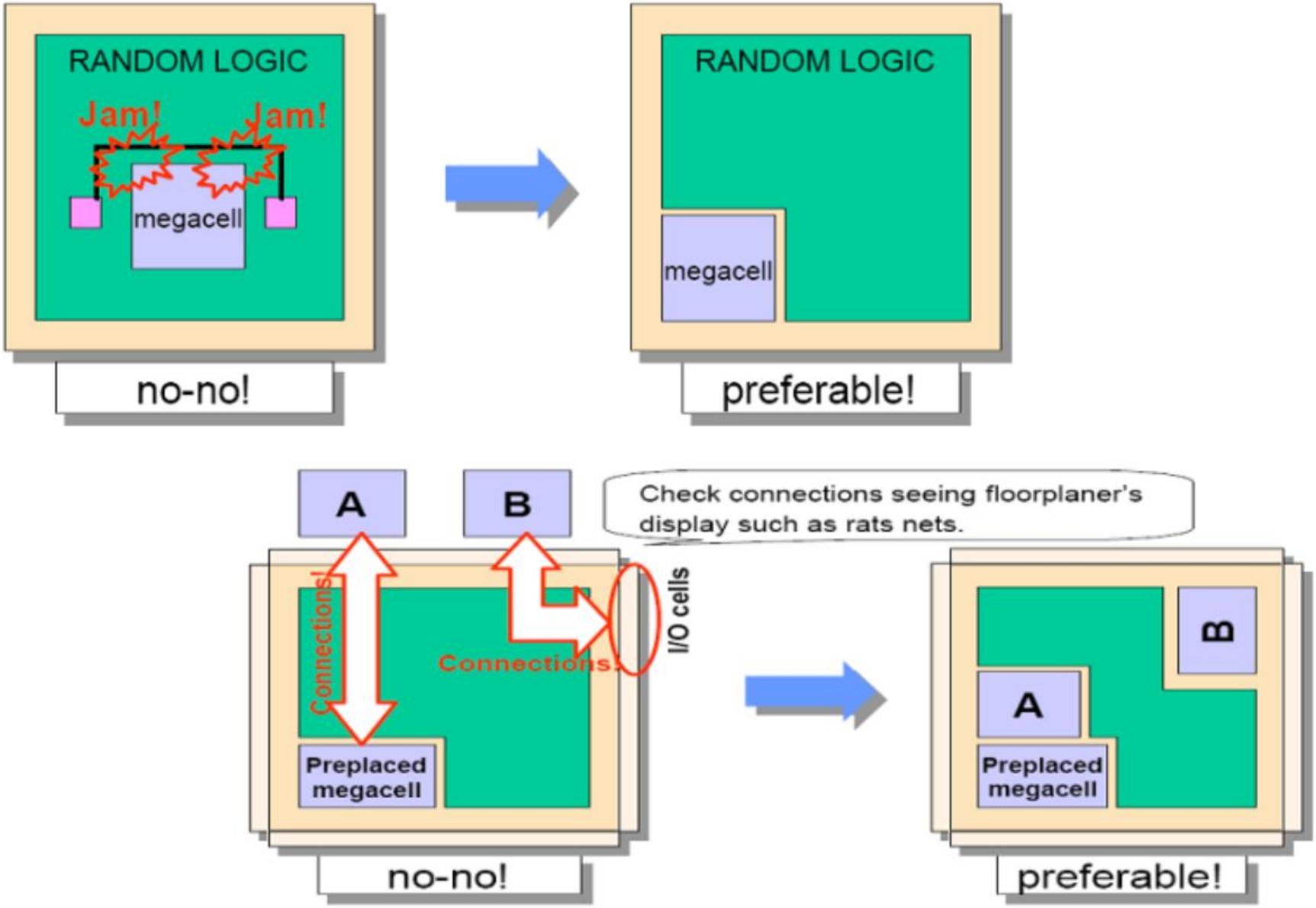
# Mixed-Size Placement

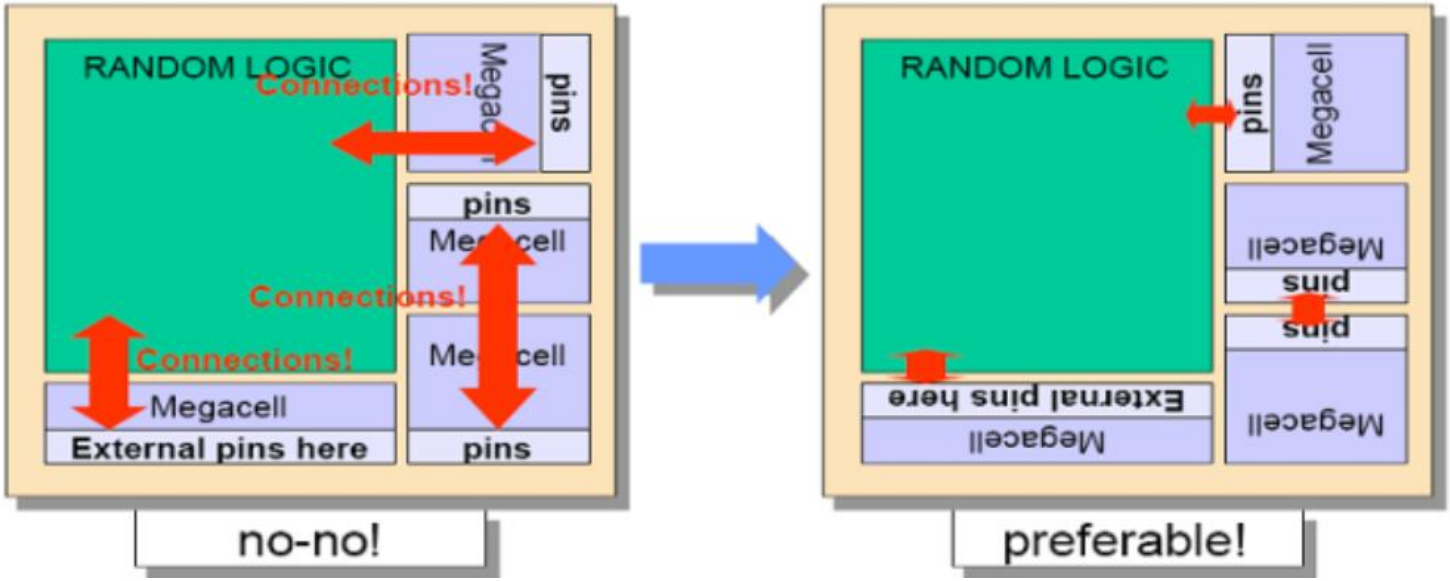
- (by Jason Cong et al) Global placement (likely each cell size is fixed or a point, using analytical method to get one),
  - then, remove overlapping,
  - greedy approach to legalize standard cells;
  - then, sliding-window-based cell swapping to reduce wire length
- (by Adya et al) 1st rely on an arbitrary black-box standard-cell placer to obtain an initial placement; then remove overlaps using a fixed-outline floor planner; then, focus on standard cells; (or using force-directed to get initial placement)
- (by Chen et al) multi-packing tree macro placer; an automatic floorplanning algorithm by using dataflow information and design exploration techniques to obtain high quality mixed macro and cell placement
- (by Chang/MediaTek et al) using reinforcement learning for chip placement
  - https://arxiv.org/pdf/2204.06407.pdf

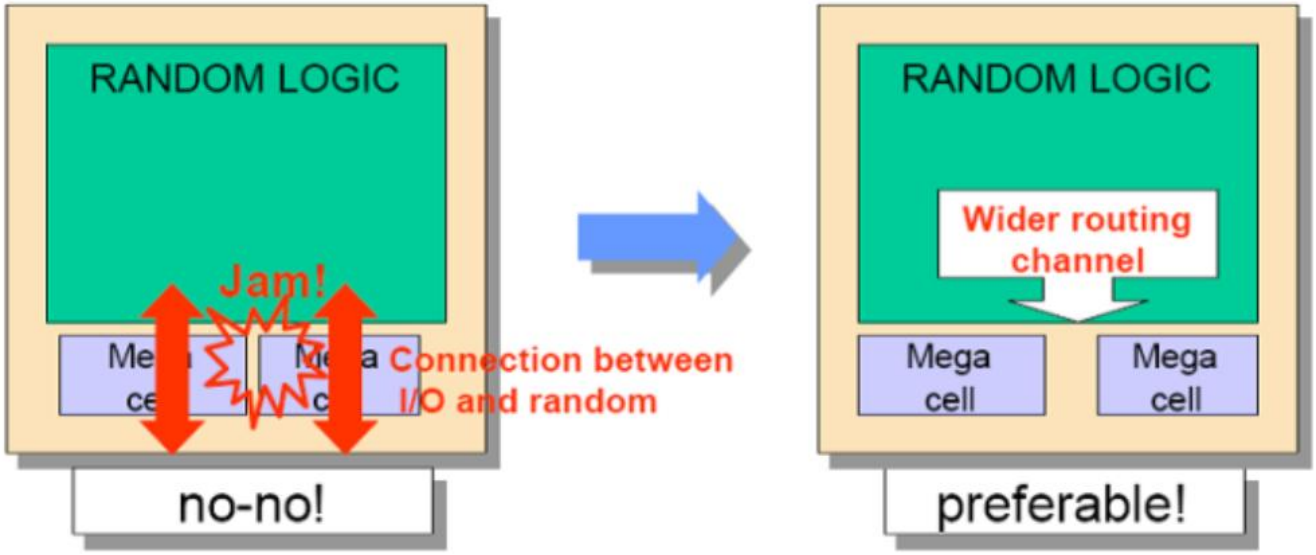
- (paper reading) you can choose floorplanning/placement using ML papers and study/present

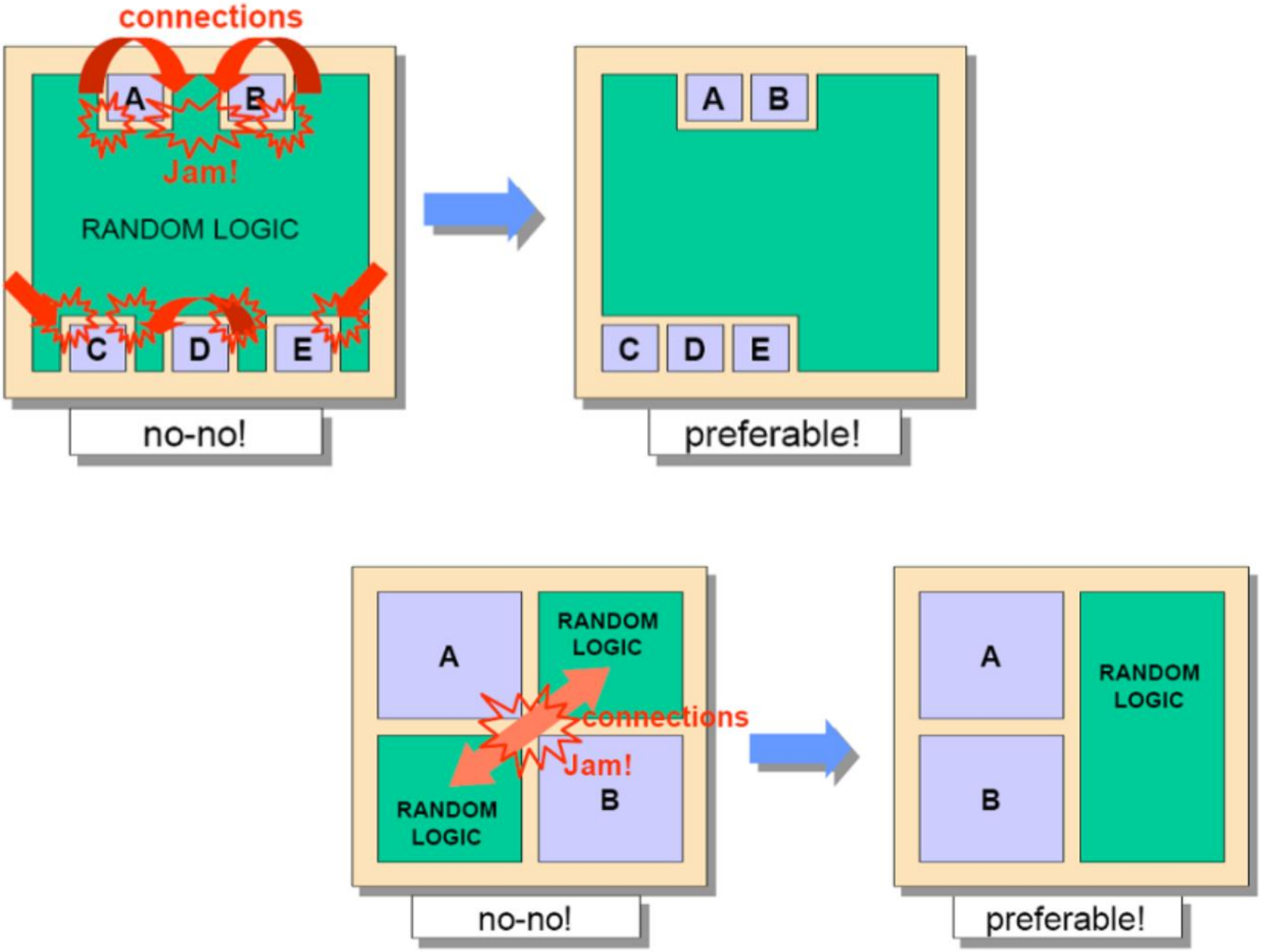
https://www.linkedin.com/pulse/guidelines-recommendations-macro-placement-ahmed-abdelazeem/

# Guidelines and recommendations for macro placement




RANDOM LOGIC
Connections!
pins
Megacell
Connections!
Connections!
Megacell
External pins here
pins
no-no!
RANDOM LOGIC
pins
Megacell
Megacell
pins
pins
Megacell
External pins here
Megacell
preferable!


RANDOM LOGIC
Jam!
Connection between
I/O and random
no-no!
RANDOM LOGIC
Wider routing
channel
Mega
cell
Mega
cell
preferable!


connections
A
B
Jam!
RANDOM LOGIC
C
D
E
no-no!
A
B
C
D
E
preferable!
A
RANDOM LOGIC
connections
Jam!
B
RANDOM LOGIC
no-no!
A
RANDOM LOGIC
B
preferable!

# Analogy To Chess & Go For Placement

# Google's paper

https://www.nature.com/articles/s41586-021-03544-w.epdf?sharing_token=tYaxh2mR5EozfsSL0WHZLdRgN0jAjWel9jnR3ZoTv0PW0K0NmVrRsFPaMa9Y5We9O4Hqf_liatg-lvhiVcYpHL_YQpqkurA31sxqtmA-E1yNUWVMMVSBxWSp7ZFFIWawYQYnEXoBE4esRDSWqubhDFWUPyI5wK_5B_YIO-D_kS8%3D
From Google

Article

# A graph placement methodology for fast chip design

https://doi.org/10.1038/s41586-021-03544-w

Received: 3 November 2020

Accepted: 13 April 2021

Published online: 9 June 2021

Check for updates

Azalia Mirhoseini[1,4]✉, Anna Goldie[1,3,4]✉, Mustafa Yazgan[2], Joe Wenjie Jiang[1], Ebrahim Songhori[1], Shen Wang[1], Young-Joon Lee[2], Eric Johnson[1], Omkar Pathak[2], Azade Nazi[1], Jiwoo Pak[2], Andy Tong[2], Kavya Srinivasa[2], William Hang[3], Emre Tuncer[2], Quoc V. Le[1], James Laudon[1], Richard Ho[2], Roger Carpenter[2] & Jeff Dean[1]

Chip floorplanning is the engineering task of designing the p... computer chip. Despite five decades of research[1], chip floorplanning has defied automation, requiring months of intense effort by physical design engineers to

# Key Takeaways

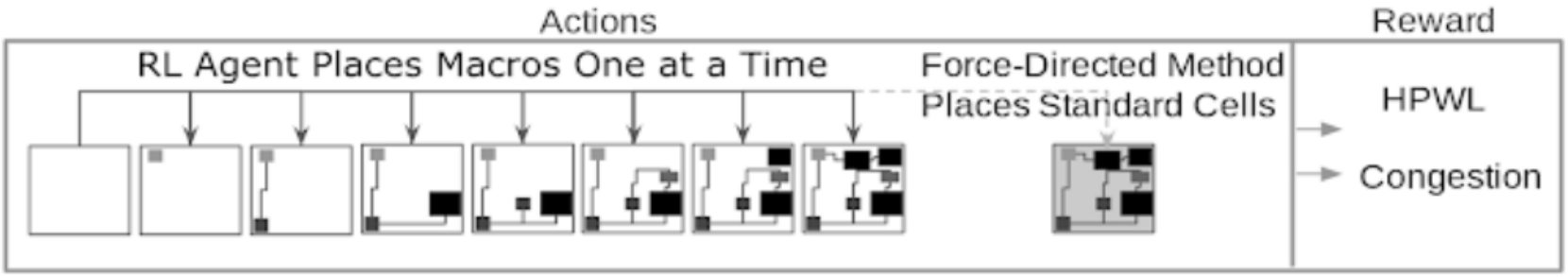
- Deep reinforcement learning method that outperforms/matches human expert performance on chip floorplanning

- Generates placements in under 6 hours, whereas human-expert baselines take weeks or months at a high operation and opportunity cost

- Superhuman chip floorplans generated by this method were used in Google's latest AI accelerator (TPU)!

So far it is to place macros, not standard cells

https://ai.googleblog.com/2020/04/chip-design-with-deep-reinforcement.html




During each training iteration, the macros are placed by the policy one at a time and the standard cell clusters are placed by a force-directed method. The reward is calculated from the weighted combination of approximate wirelength and congestion.

In each iteration of training, **the macros are sequentially placed by the RL** agent, after which the standard cell clusters are placed by a force-directed method, which models the circuit as a system of springs to minimize wirelength. RL training is guided by a fast-but-approximate reward signal calculated for each of the agent's chip placements using the weighted average of approximate wirelength (i.e., the half-perimeter wirelength, HPWL) and approximate congestion (the fraction of routing resources consumed by the placed netlist).

…

For example, the pre-trained policy organically identifies an arrangement that **places the macros near the edges** of the chip with a convex space in the center in which to place the standard cells. This results in lower wirelength between the macros and standard cells without introducing excessive routing congestion.

https://www.youtube.com/watch?v=zR9IusOpEzk

**Google's Chip Designing AI**

https://ivlsi.com/macro-placement-guidelines-vlsi-physical-design/

**Macro Placement Guidelines**

https://dl.acm.org/doi/10.1145/3505170.3506722#.YlhIc9A5lh0.linkedin

**Congestion and Timing Aware Macro Placement** Using **Machine Learning** Predictions from Different Data Sources: Cross-design Model Applicability and the Discerning Ensemble

https://www.physicaldesign4u.com/2020/02/placement.html

**Placement**

[2102.03357] Machine Learning for Electronic Design Automation: A Survey (arxiv.org) https://arxiv.org/pdf/2102.03357.pdf

**A survey**

https://ieee-ceda.org/presentation/webinar/dawn-machine-learning-eda

https://www.cadence.com/en_US/home/resources/datasheets/genus-synthesis-solution-ds.html

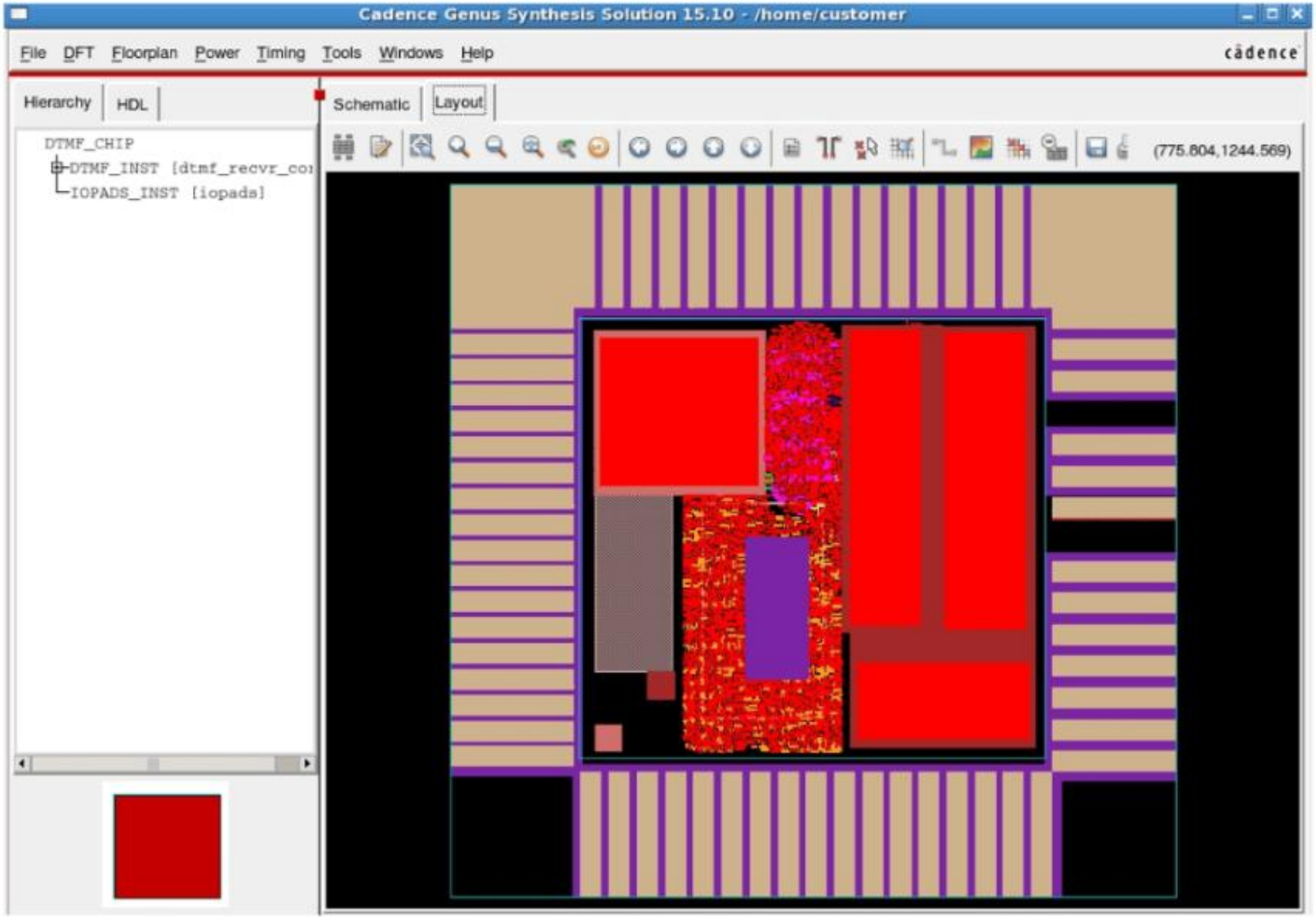



Figure 1: The Genus Synthesis Solution enables timing debug with physical interconnect knowledge built-in. Cross-probe to the physical viewer to see associated wirelengths, floorplan blockages, and estimated routing, and extract the chip-/blocklevel physical context for use in unit-level RTL design.

# Modeling Into Machine Learning / Neural Network

https://www.cerc.utexas.edu/utda/publications/publications.html
Prof. David Pan

https://github.com/limbo018/DREAMPlace/blob/master/README.md
**DREAMPlace**

Design Automation WebiNar (DAWN) - Event 1: Machine Learning for EDA (May 7, 2020)

# Analogy Between NN Training and Placement

$$\min_{\mathbf{w}} \sum_{i}^{n} f(\phi(x_i; \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

Forward Propagation (Compute obj)

Data Instance $(x_i, y_i)$ → Neural Network $\phi(\cdot; \mathbf{w})$ → Error Function $f(\phi(x_i; \mathbf{w}), y_i)$

Backward Propagation (Compute Gradient $\frac{\partial \text{obj}}{\partial \mathbf{w}}$)

**Train a neural network**

$$\min_{\mathbf{w}} \sum_{i}^{n} \text{WL}(e_i; \mathbf{w}) + \lambda D(\mathbf{w})$$

Forward Propagation (Compute obj)

Net Instance $(e_i, 0)$ → Neural Network $\text{WL}(\cdot; \mathbf{w})$ → Error Function $\text{WL}(e_i; \mathbf{w})$

Backward Propagation (Compute Gradient $\frac{\partial \text{obj}}{\partial \mathbf{w}}$)

**Solve a placement**

10

33:04 / 2:06:19

# Analog Placement

- It is different from digital's and will be discussed next semester

- UT Austin published MAGICAL in 2019

# Summary So Far

- Placement techniques
  - Clustering, Min cut, simulated annealing, quadratic placement, …
- Homework-3 and project-2

# Course Ordering Of The Major Topics

- Digital design flow
- CMOS logic gates and Boolean equations
- Algorithms and complexity
- Basic logic synthesis, technology mapping
  - Since there is a project
- Compaction
- Partitioning
- Floorplanning
- Placement
- Routing (next)
- Simulation
- High level synthesis