Placement

VLSI CAD

Compiled by Oleg Venger

Problem

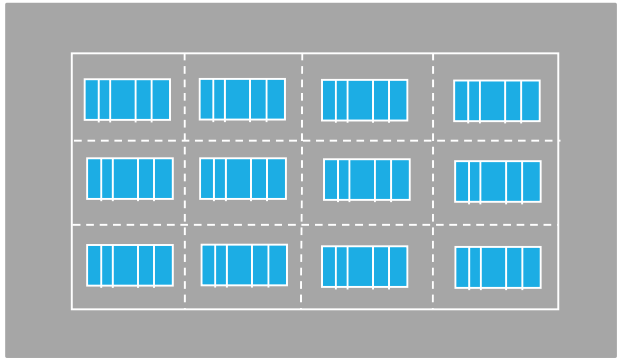
- Given:
  - Standard cell library (defines cell timing models and geometry)
  - Gate-level netlist
  - Placement region
  - Primary pin positions
- Assign cell coordinates (x_i, y_i) such that
  - All cells are inside placement region
  - There is no cell overlap
  - Some objective function is optimized
- Challenge:
  - The number of cells in block can be very large (millions)

Placement objectives

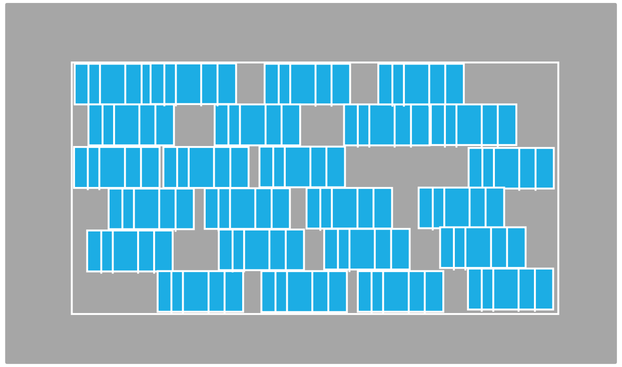
- Total wire length minimization:
  - Indirect ‘global’ delay optimization
  - routing resource usage optimization
- Total negative slack minimization
  - block performance (cycle time) optimization
- Routing congestion minimization
  - routing resource usage optimization

Placement types

- Global placement
  - generate a rough placement that may violate some placement constraints (e.g. there may be overlaps among cells)
  - Approximate cell location is decided by placing them to global bins
- Legalization
  - Makes legal the rough solution from global placement
    - no placement constraint violation
    - cells placed in rows
- Detailed placement
  - Improvement of legal solution



Global placement



Detailed placement

Placement approaches

- Partitioning based (mincut partitioning)
- Simulated annealing
- Analytical approach (best)
- Other
  - E.g. Force-directed placement

Partitioning based placement

Variables: queue of placement bins

Initialize queue with top-level placement bin

while queue not empty :

    Dequeue bin

    if bin small enough :

        Process bin with end-case placer

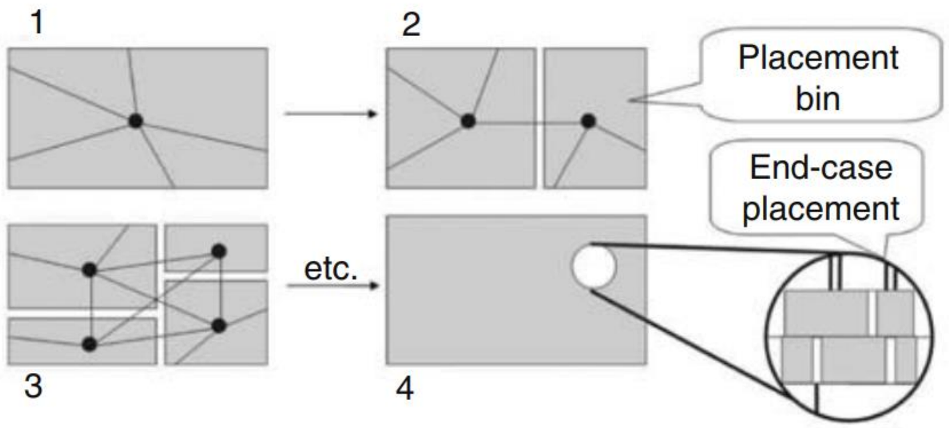
    else :

        Choose a cut-line for the bin (including direction)

        Build partitioning hypergraph from netlist and cells contained in the bin

        Partition the bin into smaller bins (generally via min-cut bi- or quadri-section)

    Enqueue each child bin



Hypergraph partitioning

- K-way partitioning:
  - given a hypergraph H, assign vertices of H to k disjoint non-empty partitions to minimize net cut (i.e. the number of nets that span more than one partition)
- Constraints:
  - Fixed assignment for some vertices
  - Limit on total vertex weight in each partition (balance constraints)
- Finding optimal solution is NP-hard

Fiduccia-Mattheyses heuristic

- Idea: Start with some partitioning and improve current partitioning by moving cells one by one
- Gain of cell move: $Gain = S(x) - E(x)$, where
  - $S(x)$: the number of nets that contain x as the only cell in one part
  - $E(x)$: the number of nets that contain x entirely located in one part
- Move with highest gain is applied at each elementary step
- Sequences of moves organized as passes

Fiduccia-Mattheyses algorithm

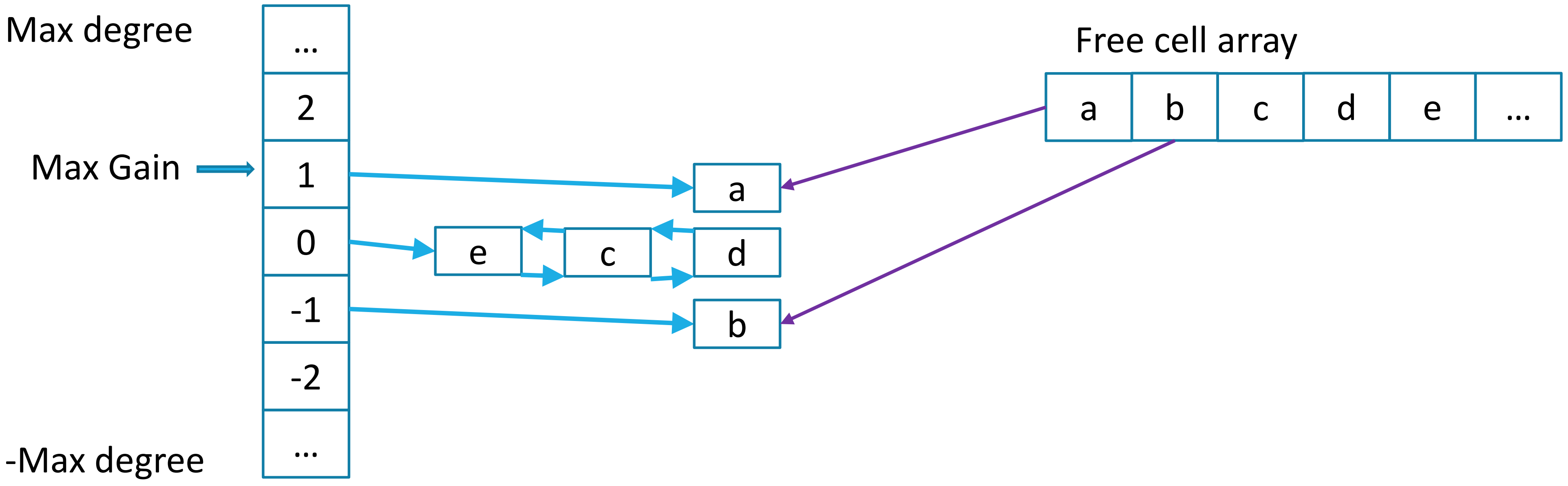
```
function FM (hypergraph, partitionment) :
do :
    initialize gain_container from partitionment
    FMpass (gain_container, partitionment)
while (solution quality improves)
```

```
function FMpass (gain_container, partitionment) :
solution_cost = partitionment.get cost()
while not all vertices locked :
    move = best_feasible_move()
    solution_cost += gain_container.get gain (move)
    gain_container.lock_vertex (move.vertex())
    gain_update (move)
    partitionment.apply (move)
roll back partitionment to best seen solution
gain_container.unlock_all()
```

Fiduccia-Mattheyses algorithm

```
function gain update (move) :

source_part = partition that move.vertex() is in

dest_part = partition where move.vertex() is going

for each hyperedge e incident to move.vertex() :
    if e has no vertices in dest_part before applying move :
        for each vertex v on e :
            gain_container.update(v, dest_part, e.weight())
    if only 1 vertex on e in source_part before applying move :
        for each vertex v on e :
            gain_container.update(v, source_part, -e.weight())

    if only 1 vertex v remains in source_part after applying move :
        gain_container.update(v, dest_part, e.weight())
    if only 1 vertex v in dest_part before applying move :
        gain_container.update(v, source part, -e.weight())
```

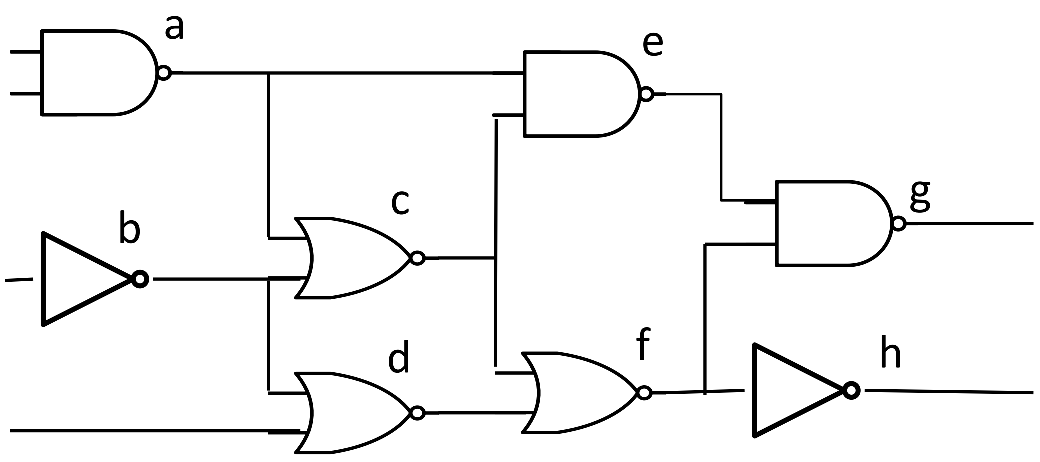
Gain container data structure



For 2 partitions we need two such data structure

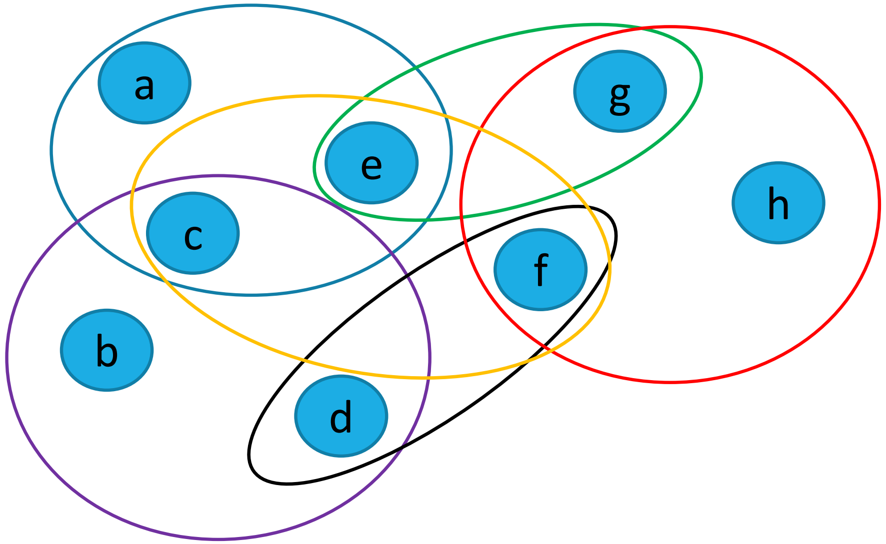
Able to access cells without searching every single array

Fiduccia-Matteyses algorithm: example



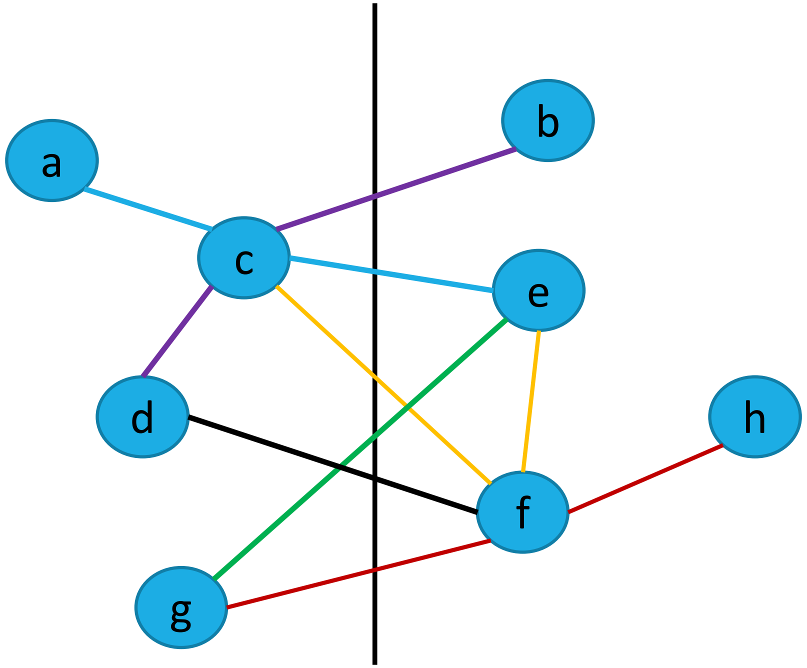
Perform FM algorithm on this circuit

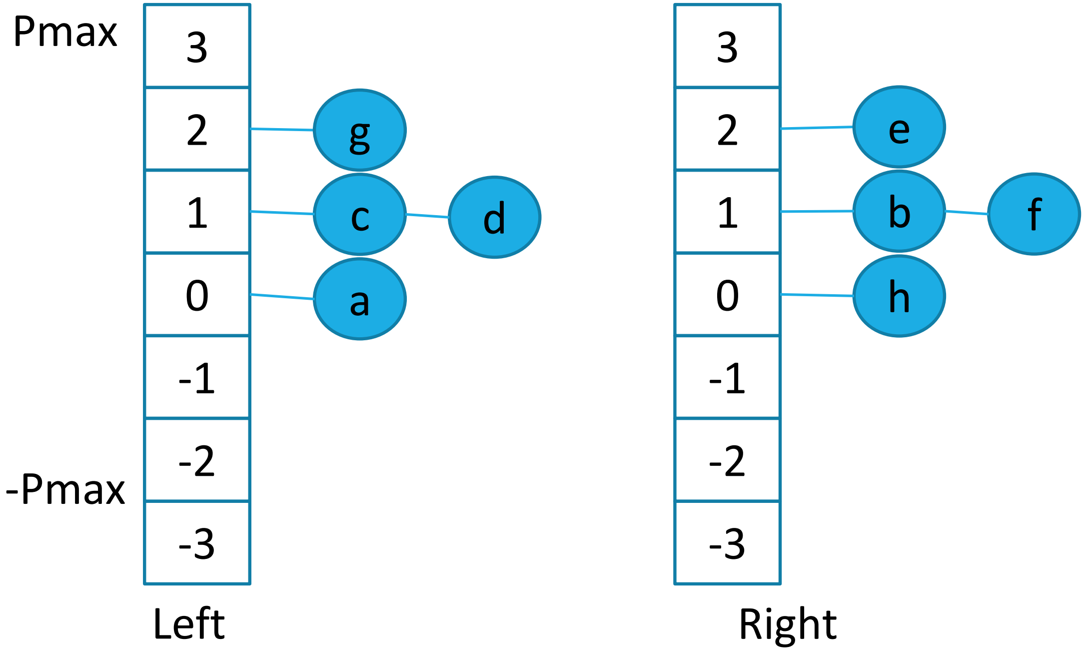
Area constraint = [3, 5]



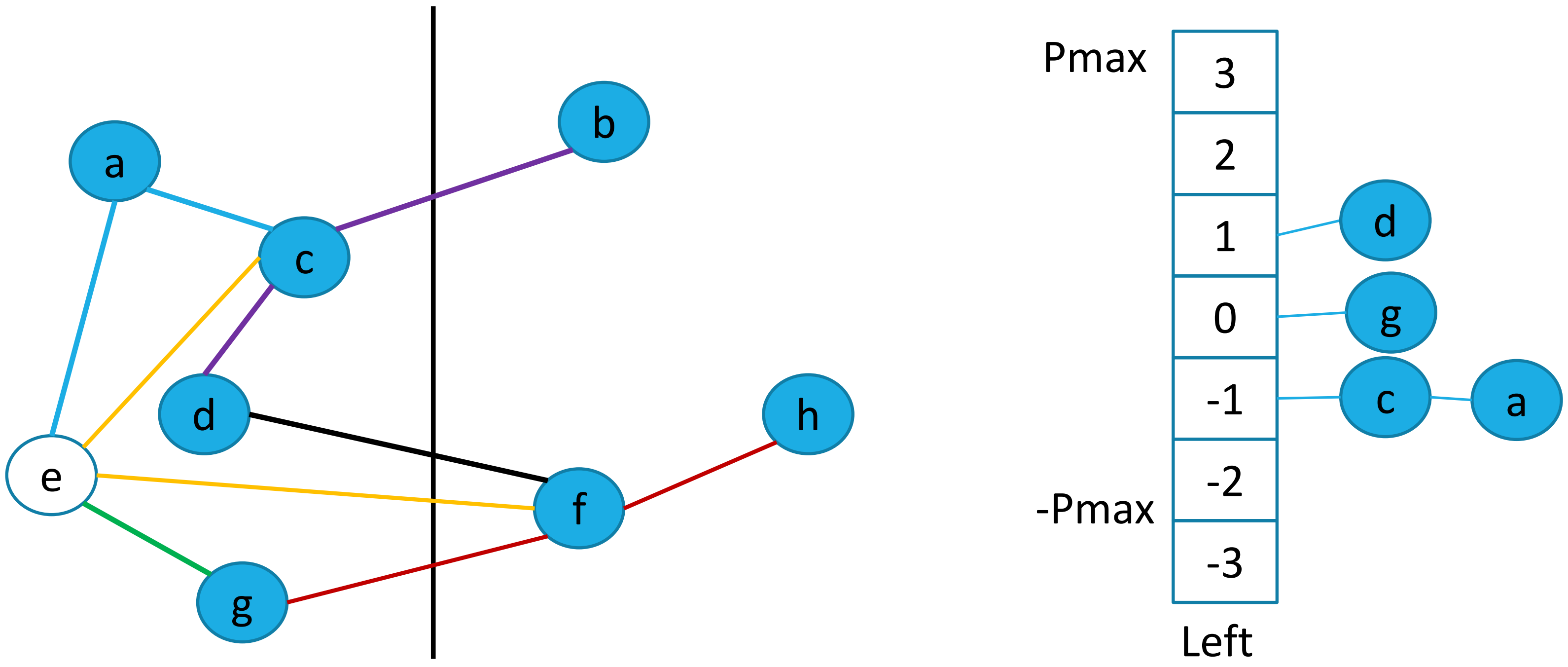
Example: step 0

Initial partitioning: random

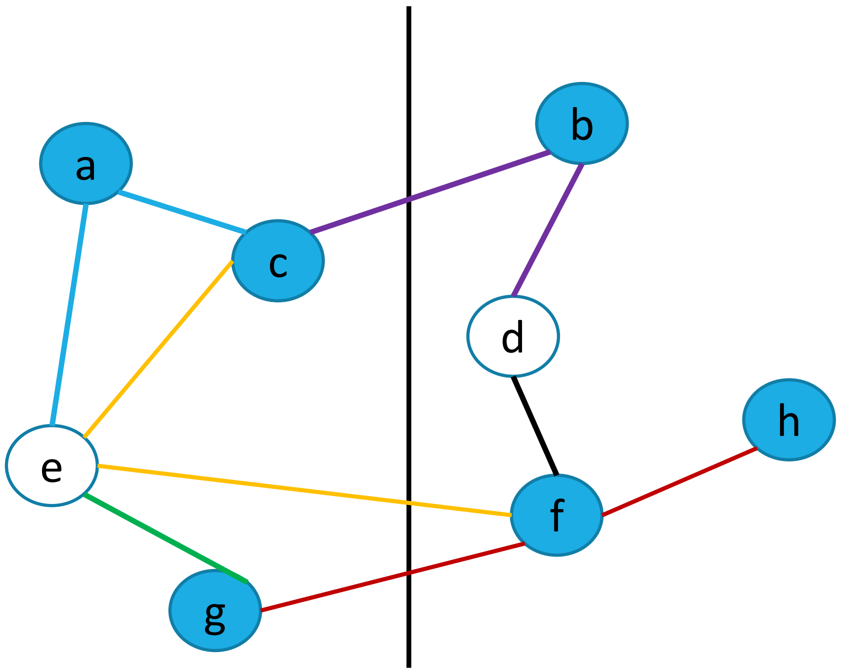


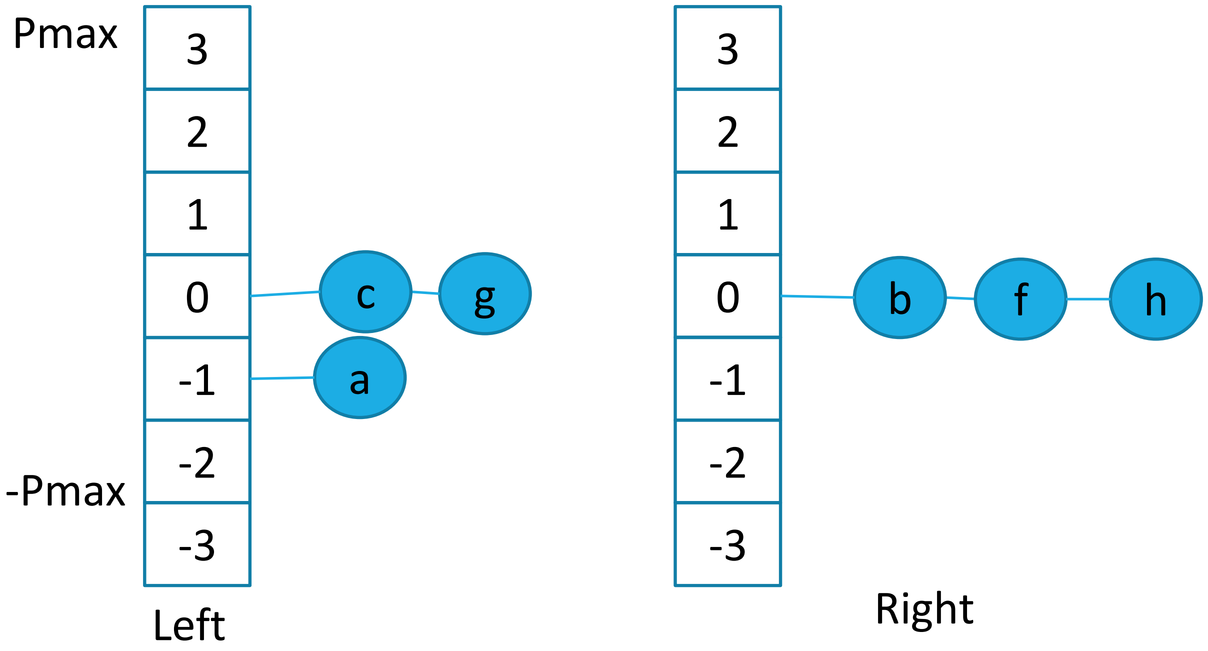


Example: step 1

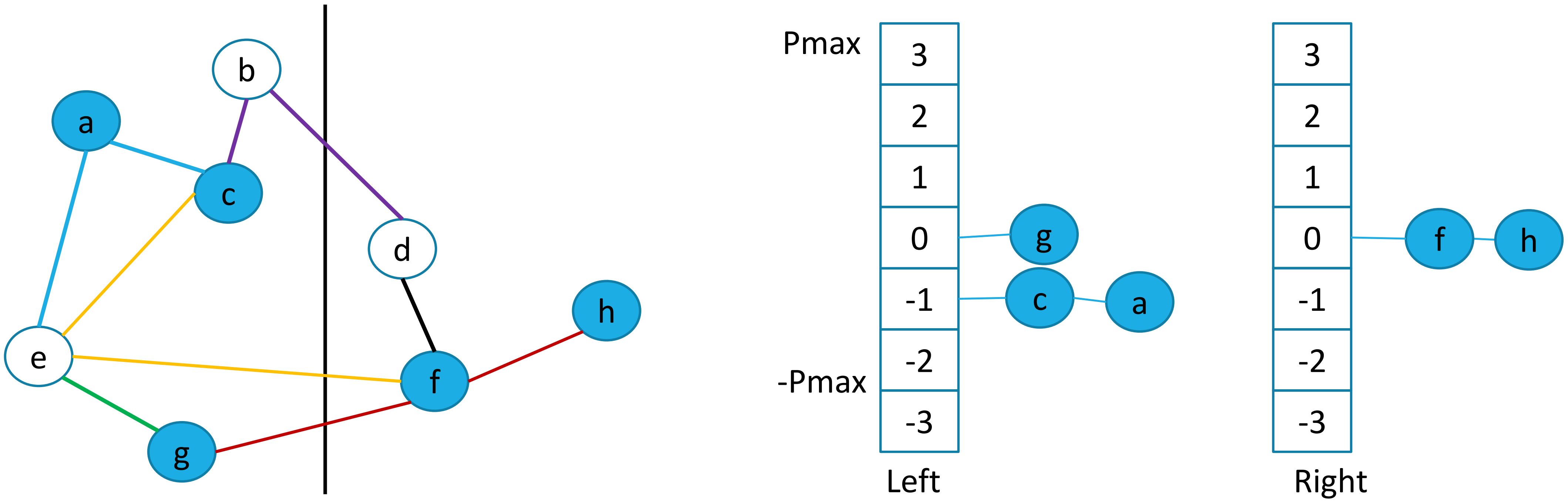


Example: step 2

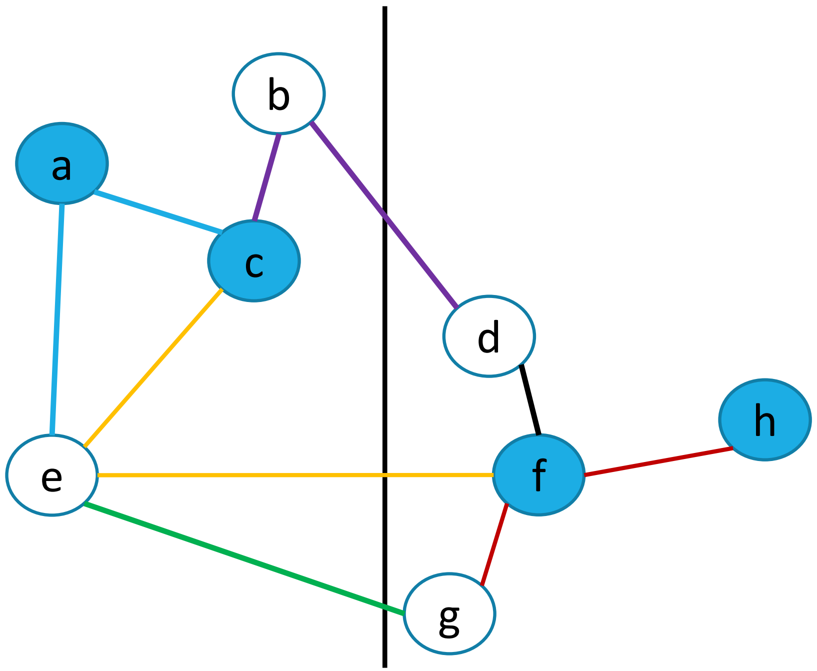


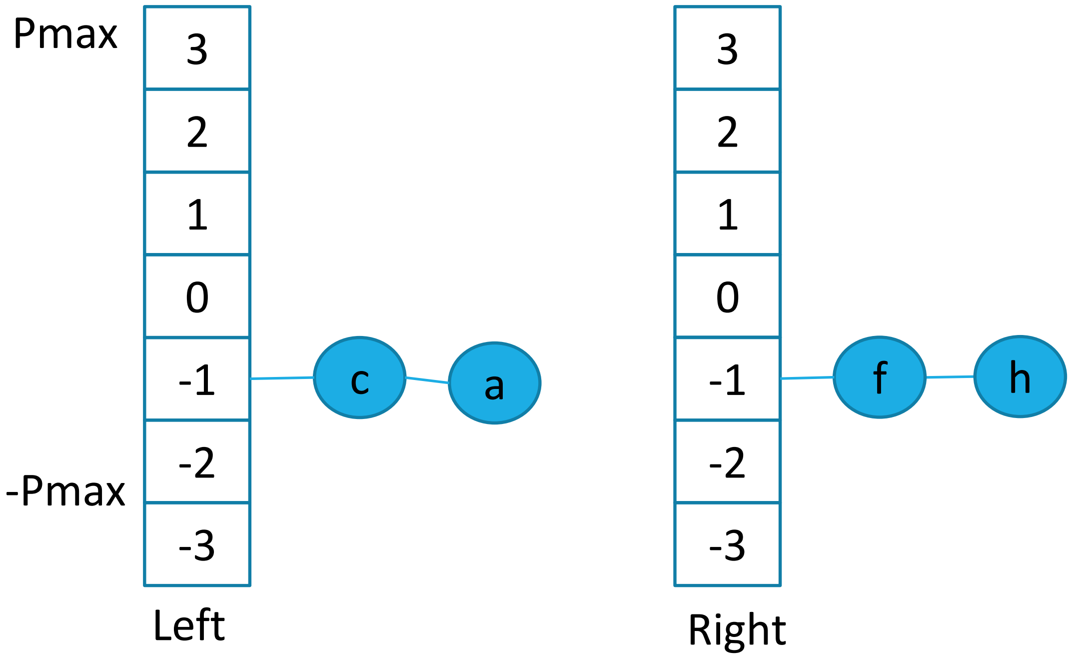


Example: step 3

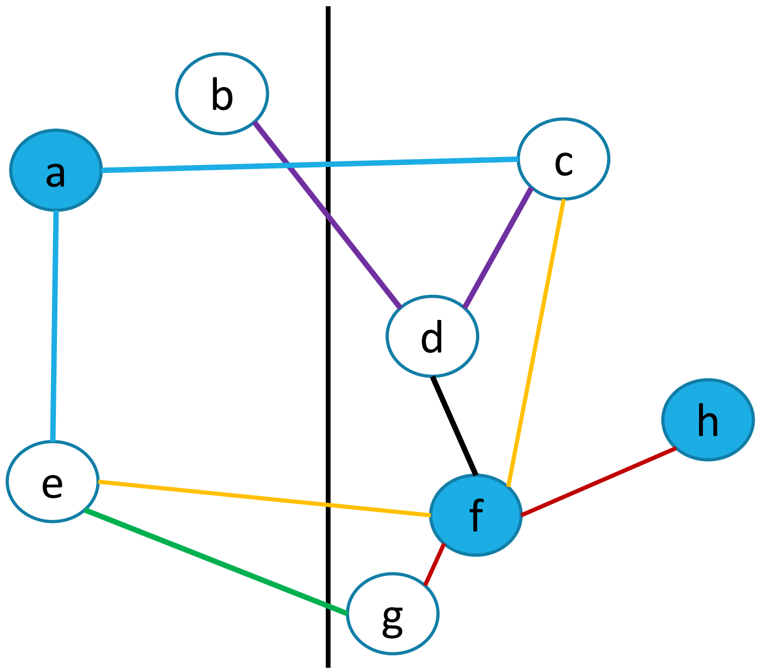


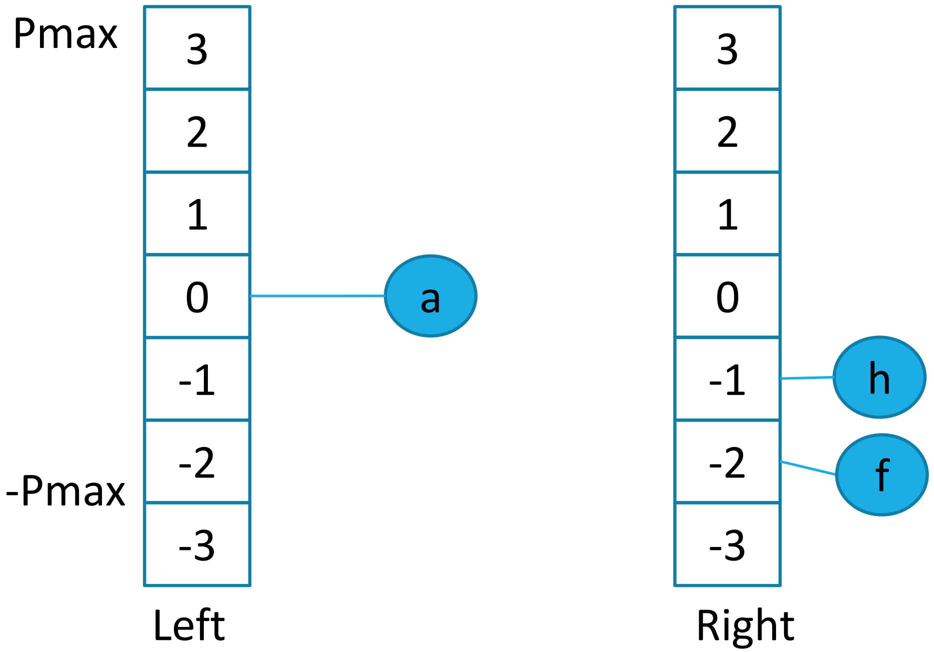
Example: step 4



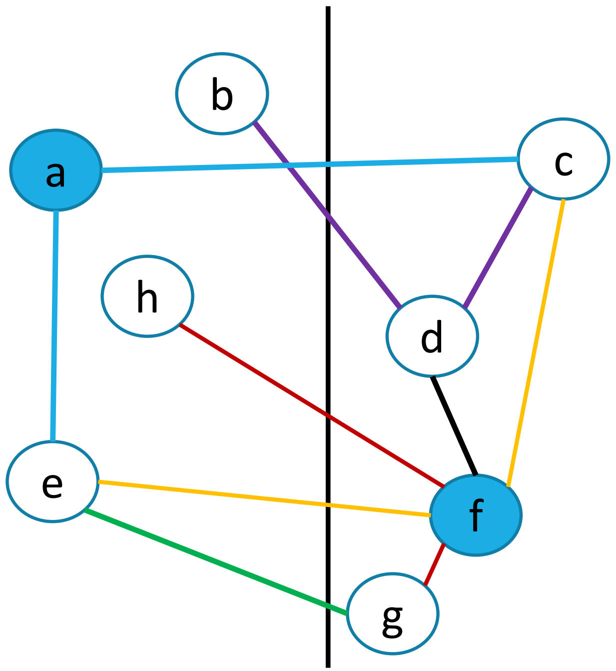


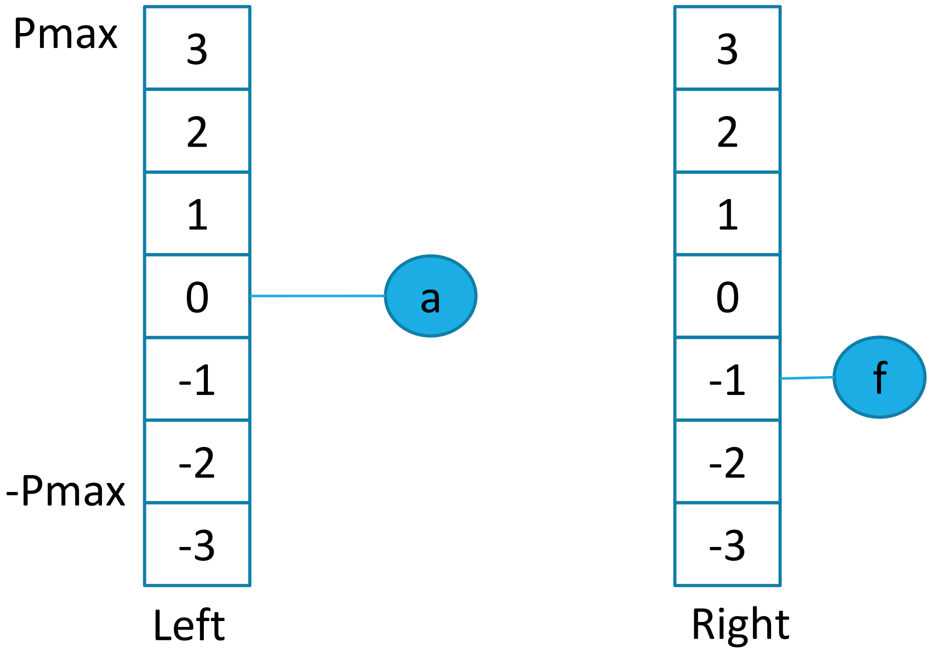
Example: step 5



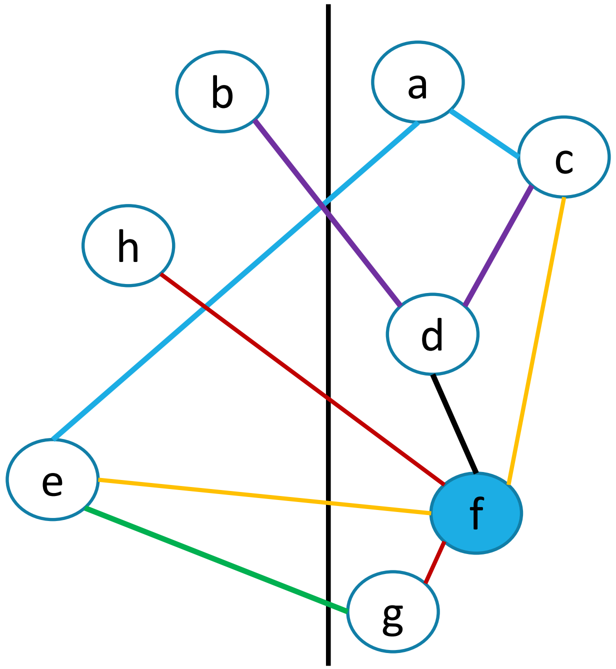


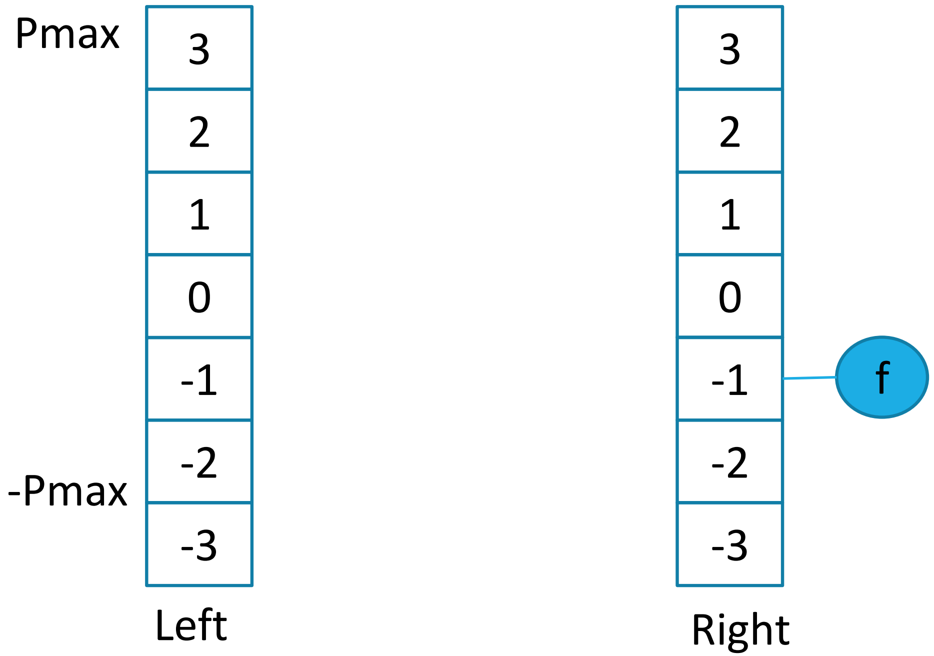
Example: step 6



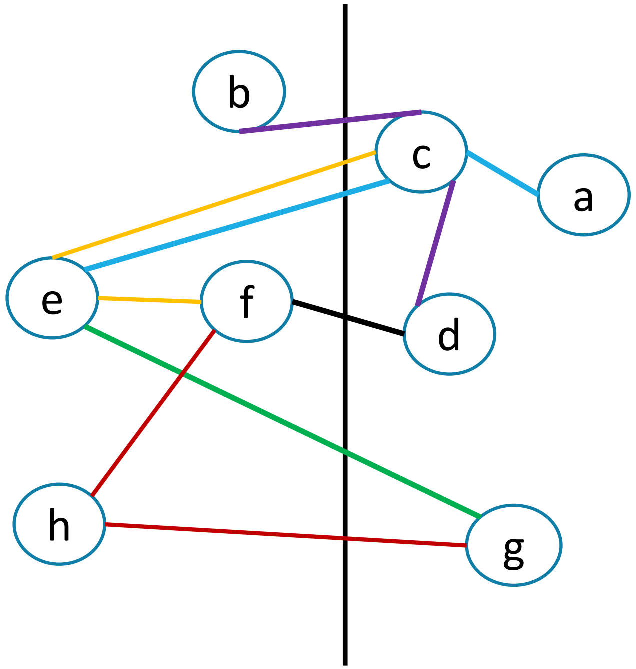


Example: step 7



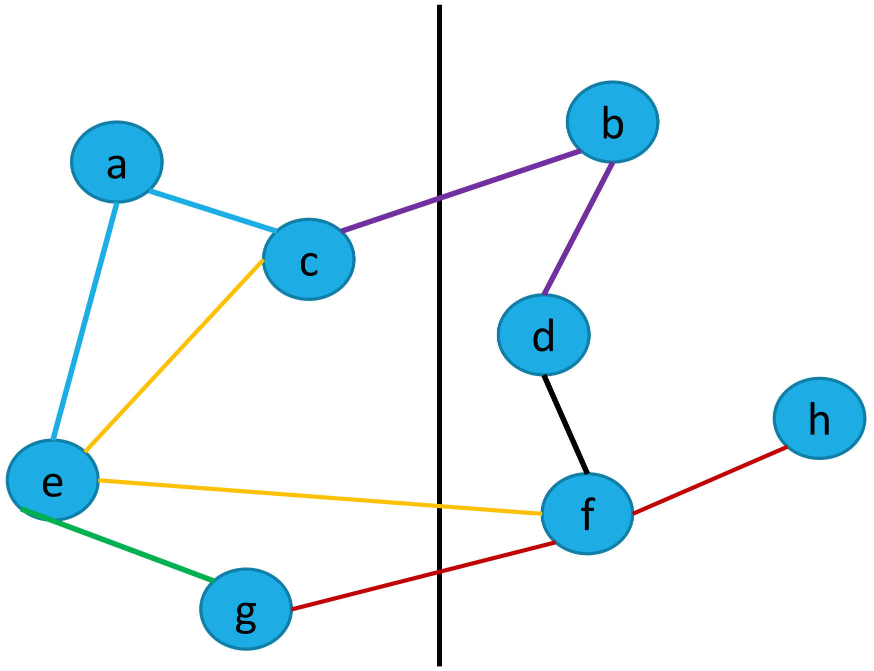


Example: Step 8



Example: Iteration result

- Best solution we seen after move 2
  - Cutsize reduced from 6 to 3
  - Balanced
- Solution after move 3 has the same cutsize but less balanced



After move 2

Multilevel FM partitioning framework

- Empirical study showed that FM partitioning works best for 35 – 200 movable vertices
- For large hypergraph we can apply multilevel clustering (coarsening), followed by top-level partitioning and refinement (uncoarsening).
- Coarsening can be done in various ways, for example, by successively choosing a vertex and a random connected neighbor to merge.
- MLFM explores more effectively the solution space by spending comparatively more effort on smaller coarsened hypergraphs

```
MLFM(hypergraph)
    level = 0;
    hierarchy[level] = hypergraph;

//Coarsening phase
    while(hierarchy[level].vertex_count() > 200)
        next_level = cluster(hierarchy[level]);
        level = level + 1;
        hierarchy[level] = next_level;

//Top level partitioning phase
    partitionment[level]
           = a random initial partitionment for top-level hypergraph;
    FM(hierarchy[level], partitionment[level])

//Refinement phase
    while(level > 0)
        level = level - 1;
        partitionment[level]
           = project(partitionment[level+1], hierarchy[level]);
        FM(hierarchy[level], partitionment[level])

    return partitionment[0];
```

Pros and cons of partitioning based placement

- Pros:
  - Fast algorithm and good scaling
- Cons:
  - Indirect optimization
  - Not resilient: incremental change in input netlist may lead to completely different solution

Simulated annealing placement

- Annealing in metals:
- Heat the solid state metal to a high temperature
- Cool it down very slowly according to a specific schedule
- *If the heating temperature is sufficiently high to ensure random state and the cooling process is slow enough to ensure thermal equilibrium, then the atoms will place themselves in a pattern that corresponds to the global energy minimum of a perfect crystal*

Simulated annealing placement

Step 1: Initialize – Start with a random initial placement. Initialize a very high “temperature”.

Step 2: Move – Perturb the placement through a defined move.

Step 3: Calculate score – calculate the change in the cost due to the move made.

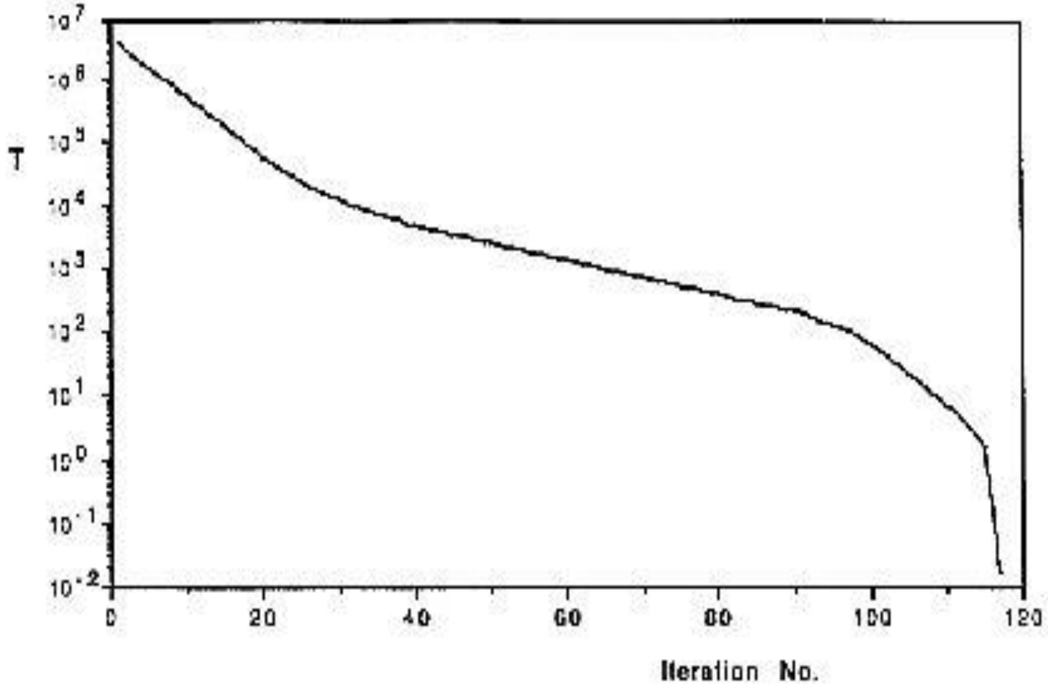
Step 4: Choose – Depending on the score, accept or reject the move. The probability of acceptance depending on the current “temperature”.

$$P = \begin{cases} 1, & if\ \Delta C < 0 \\ e^{-\Delta C/T}, & if\ \Delta C \geq 0 \end{cases}$$

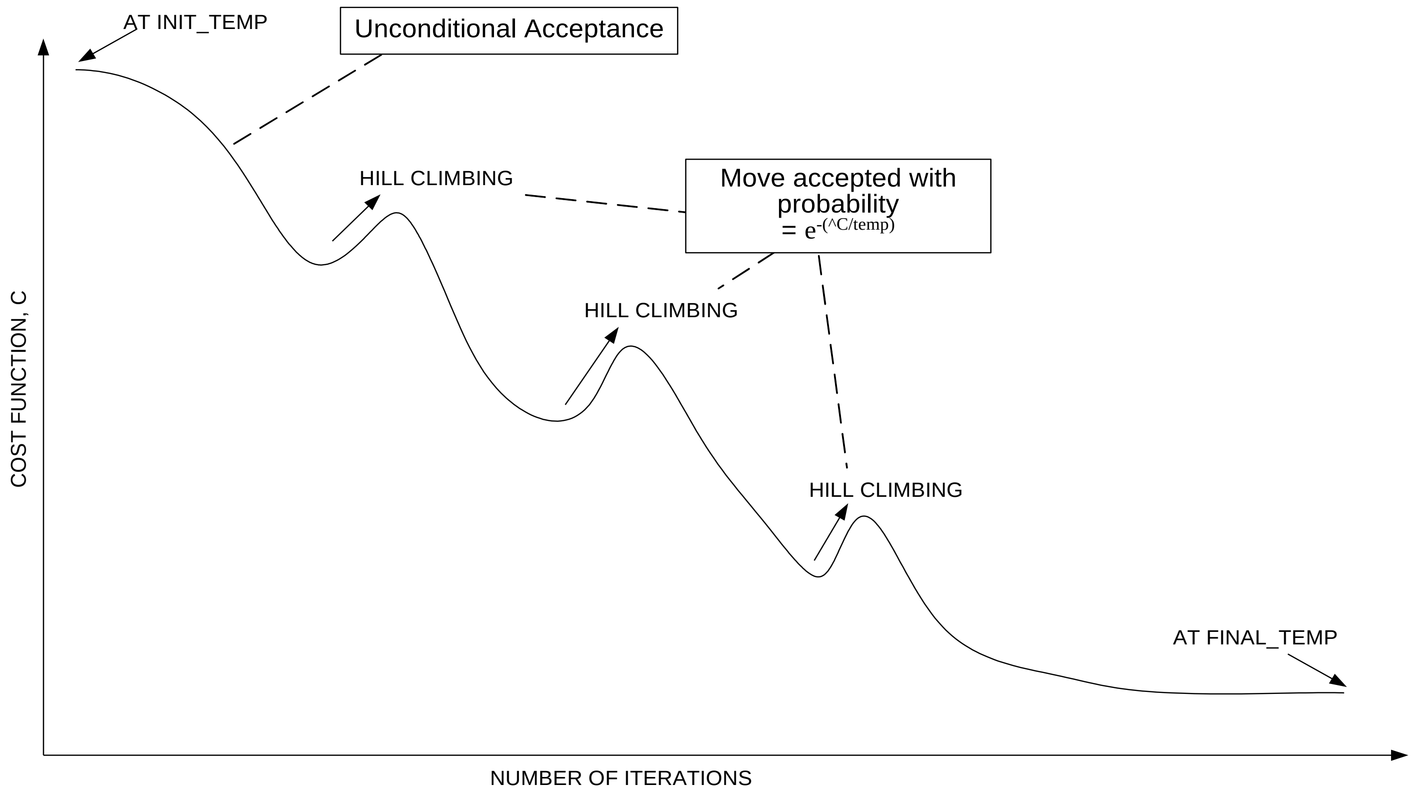
Step 5: Update and repeat– Update the temperature value by lowering the temperature. Go back to Step 2.

The process is done until “Freezing Point” is reached.

Simulated annealing placement



Cooling schedule

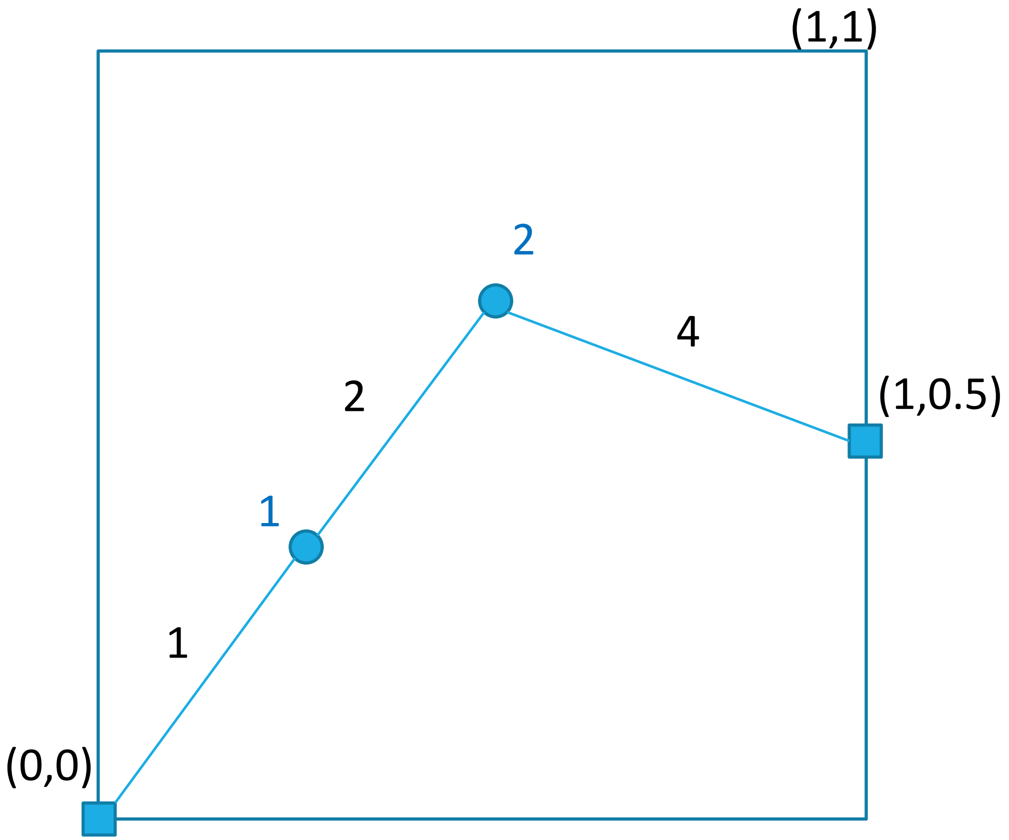


Convergence of simulated annealing

Pros and cons of simulated annealing

- Pros:
  - Statistically finds globally optimal solution (given enough time)
    - Generally gives a good solution
  - Can deal with arbitrary cost functions
  - Easy to code
- Cons:
  - Extremely slow process of reaching good solution
  - Cannot tell whether it has found an optimal solution

Quadratic placement



Squared Euclidian total wirelength here is:

$$4(x_2 - 1)^2 + 4(y_2 - 0.5)^2 +$$
$$2(x_2 - x_1)^2 + 2(y_2 - y_1)^2 +$$
$$1(x_1 - 0)^2 + 1(y_1 - 0)^2$$

The goal is to minimize it.

The minimum is at the point where all partial derivatives are zero.

Can be solved separately for x and y

Partial derivatives

$Q(X) = 4(x_2 \ - 1)^2 + 2(x_2 \ - x_1 \ )^2 + 1(x_1 \ - 0)^2$

$$\frac{\partial Q}{\partial x_1} = 0 + 2 \cdot 2(x_2 - x_1)(-1) + 2x_1 =$$

$6x_1 - 4x_2 = 0$

$$\frac{\partial Q}{\partial x_2} = 4 \cdot 2(x_2 - 1) + 2 \cdot 2(x_2 - x_1) + 0 =$$

$-4x_1 + 12x_2 - 8 = 0$

$Q(Y) = 4(y_2 \ - 0.5)^2 + 2(y_2 \ - y_1 \ )^2 + 1(y_1 \ - 0)^2$

$$\frac{\partial Q}{\partial y_1} = 0 + 2 \cdot 2(y_2 - y_1)(-1) + 2y_1 =$$

$6y_1 - 4y_2 = 0$

$$\frac{\partial Q}{\partial y_2} = 4 \cdot 2(y_2 - 0.5) + 2 \cdot 2(y_2 - y_1) + 0 =$$

$-4y_1 + 12y_2 - 4 = 0$

2 systems of linear equations

$Q(X) = 4(x_2 - 1)^2 + 2(x_2 - x_1)^2 + 1(x_1 - 0)^2$

$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \end{pmatrix}$$

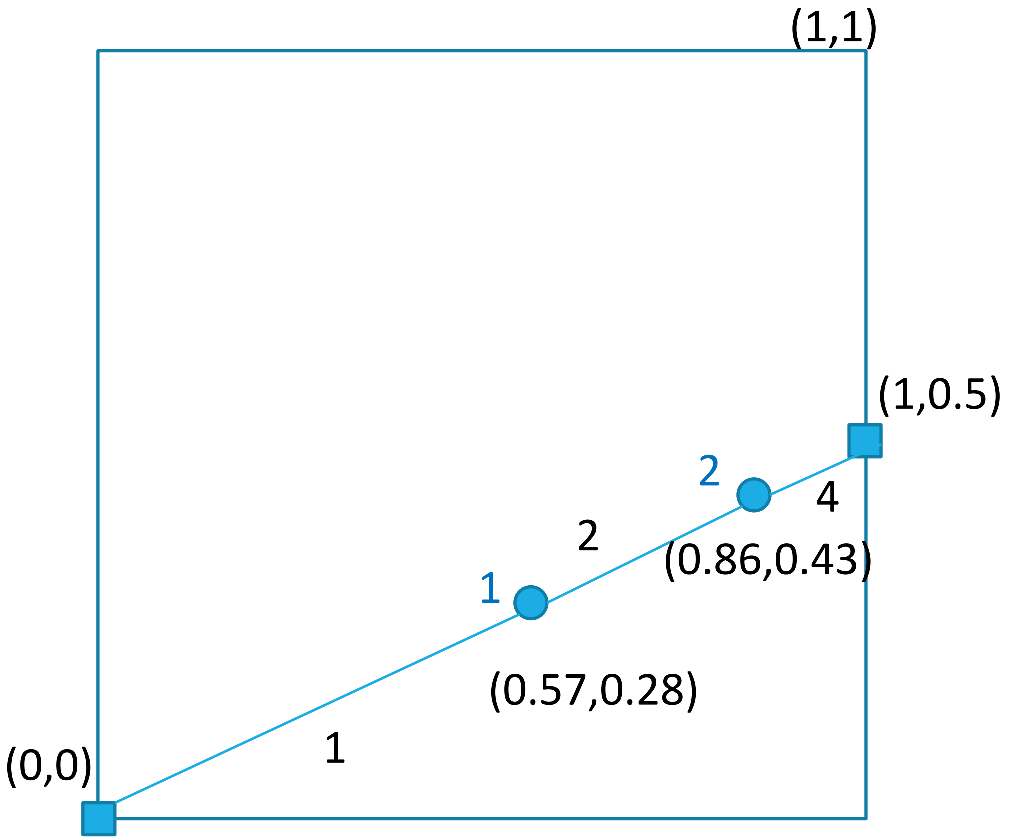
$x_1 = 0.571, x_2 = 0.857$

$Q(Y) = 4(y_2 - 0.5)^2 + 2(y_2 - y_1)^2 + 1(y_1 - 0)^2$

$$\begin{pmatrix} 6 & -4 \\ -4 & 12 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$$

$y_1 = 0.286, y_2 = 0.429$

Placement result



All cells on one line between the pads

Bigger wire weight leads to shorter wire

Building the matrix A

- First, build the NxN connectivity matrix C where :
  - If gate i connected to gate j with wire of weight w, then $c[i,j] = c[j,i] = w$, else = 0
- Then, build the matrix A using C as:
  - Non-diagonal $a[i,j] \; = \; -c[i,j]$
  - Diagonal $a[i,i] = \sum_{j=1}^{N} c[i,j] + \text{total weight of wires connected to pads}$

Building the right hand side vector

For $Ax = b_x$ vector:

If gate i connects to a pad at (x_i, y_i) with a wire that has weight w_i then

$b_x[i] = w_i \cdot x_i$

$$\begin{pmatrix} \\ A \\ \\ \end{pmatrix}\begin{pmatrix} \\ x \\ \\ \end{pmatrix} = \begin{pmatrix} \\ b_x \\ \\ \end{pmatrix}$$

For $Ay = b_y$ vector:

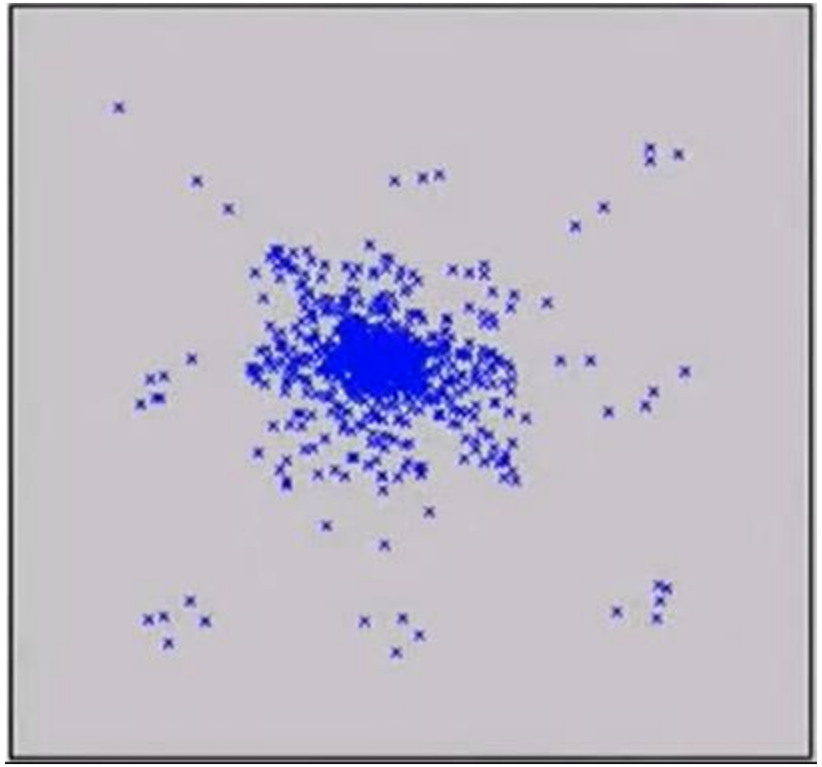
If gate i connects to a pad at (x_i, y_i) with a wire that has weight w_i then

$b_y[i] = w_i \cdot y_i$

$$\begin{pmatrix} \\ A \\ \\ \end{pmatrix}\begin{pmatrix} \\ y \\ \\ \end{pmatrix} = \begin{pmatrix} \\ b_y \\ \\ \end{pmatrix}$$

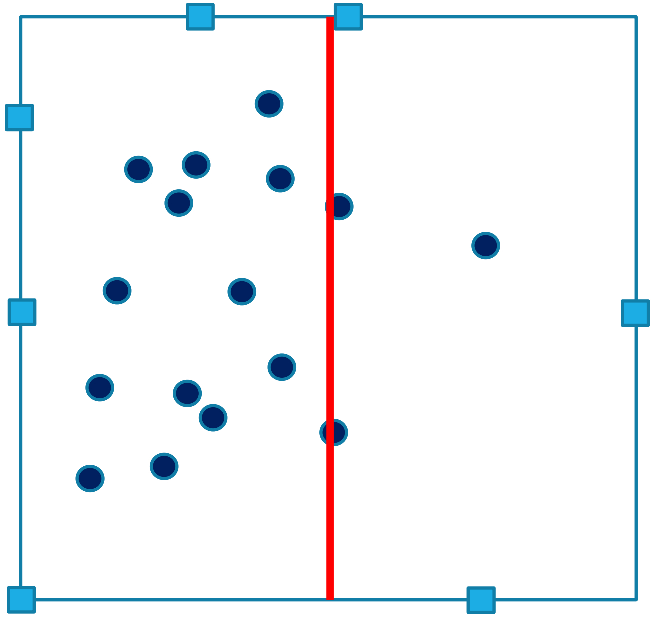
Discussion

- These systems of equations are very easy to solve using iterative approximate solvers
- Multiple pin nets can be modeled as clique of 2-pin edges with normalized weights
- Assuming all weights equal, the quadratic placement over optimizes long wires and under optimized short wires. So, the linear total wirelength is not optimal
- It produces the result with huge overlap between cells



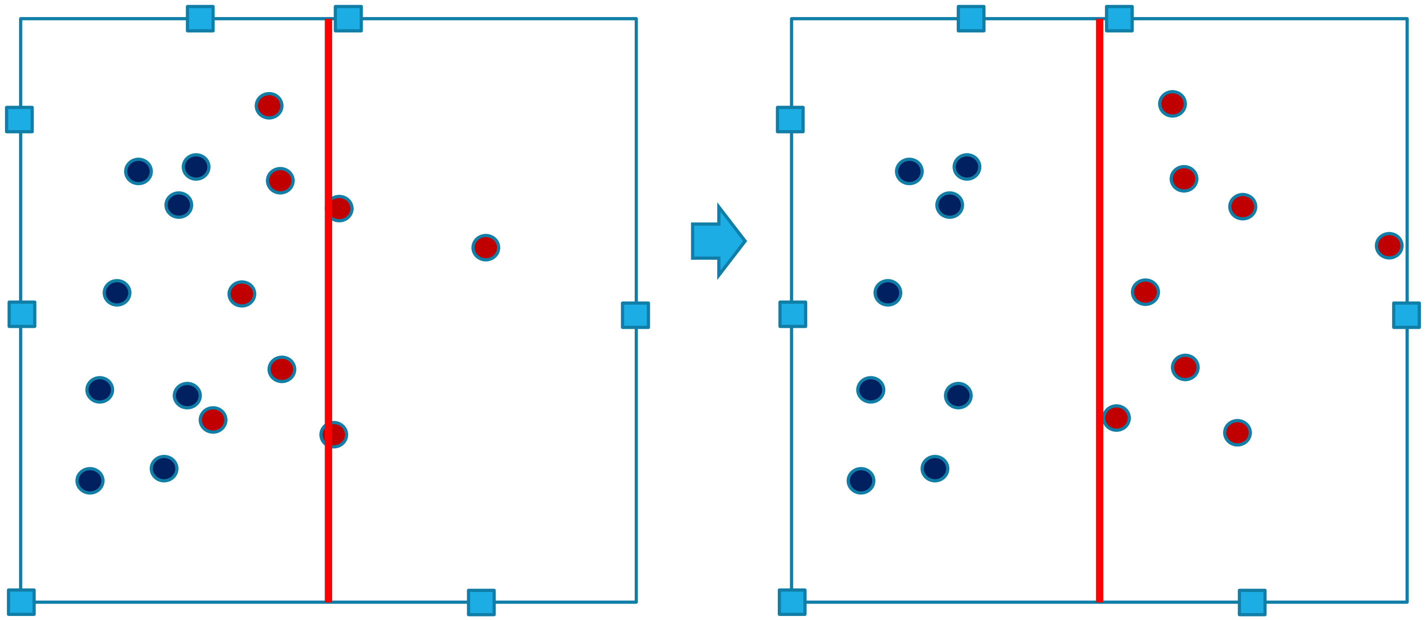
Analytical placement result on real design

Cell spreading with recursive partitioning



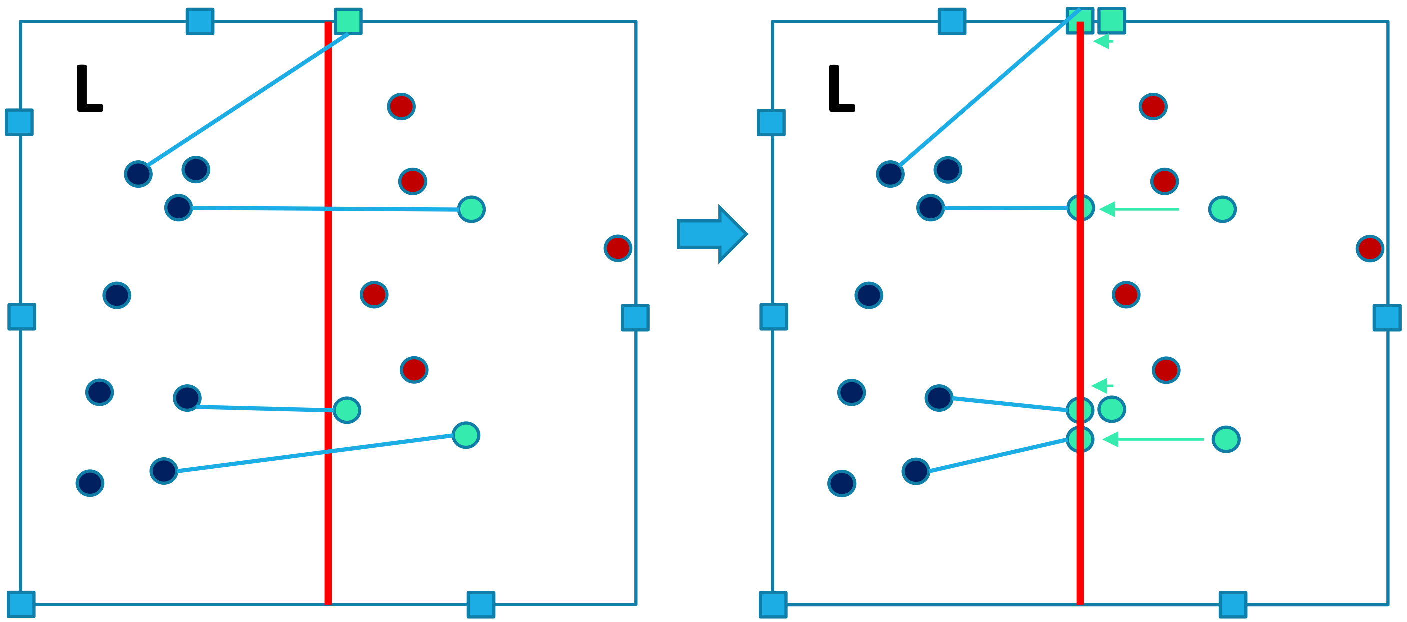
- After first QP, divide placement area in half (for example, vertically).

Assign cells to subregions



- Sort placed gates on x coordinate.
- For N movable gates, first $N/2$ gates assigned to left side, others go to the right side

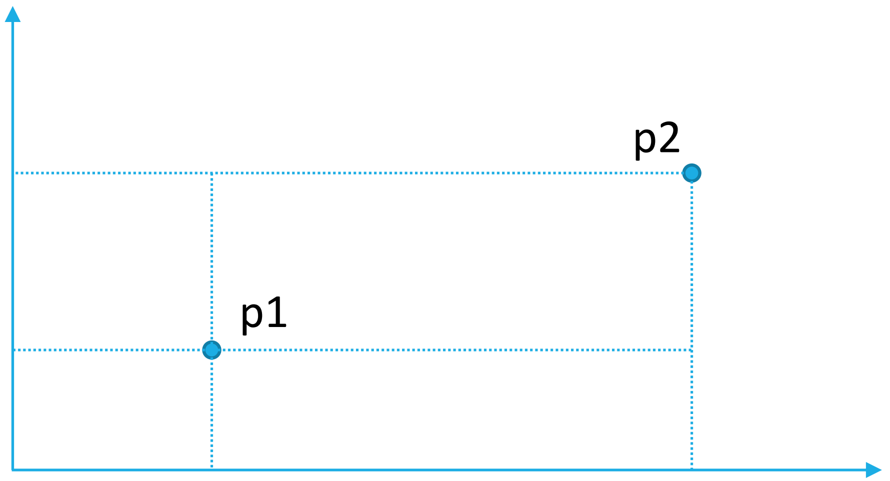
Formulate QP subproblems



- QP subproblem formulation for left side (L):
  - Every vertex not inside L that is connected to a vertex inside the L, is modeled as pad on boundary of L
  - Propagate these outside vertices using their current (x, y) location to the nearest point on boundary of L.
- After solving subproblems, recursively apply the technique
- Stop when problem size becomes small enough

Analytical placement

- Placement problem revisited:
  - $\min HPWL(x,y)$ such that density $D_g(x,y) = D_g$ for each grid bin g
- This is constrained nonlinear optimization problem
- To solve the problem using nonlinear optimization techniques, we need to have smooth wirelength and density functions



$$HPWL(p1, p2) = |x_1 - x_2| + |y_1 - y_2|$$

LOG-SUM-EXP Wirelength Function

- For a net e with pin coordinates $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, the smooth wirelength function is:

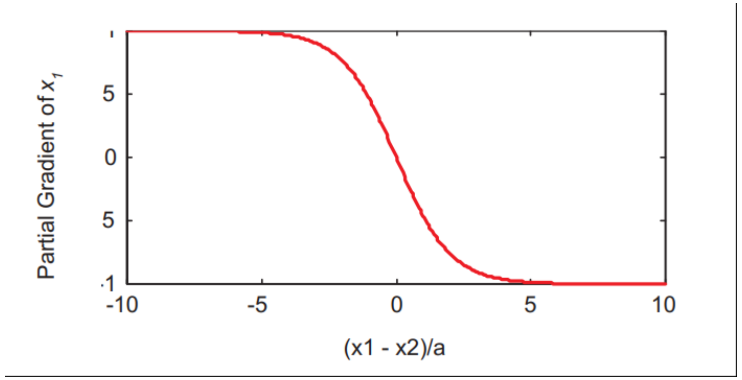
$$WL(e) = \alpha\left(\log\left(\sum e^{x_i/\alpha}\right) + \log\left(\sum e^{-x_i/\alpha}\right)\right) + \alpha(\log\left(\sum e^{y_i/\alpha}\right) + \log(\sum e^{-y_i/\alpha}))$$

Where α is a smoothing parameter

- $WL(e)$ is strictly convex, continuously differentiable and converges to $HPWL(e)$ as α converges to 0

- For a 2-pin net, the partial gradient of $WL(e)$: $\frac{\partial WL}{\partial x_1} = \frac{1}{1+e^{\frac{x_1 - x_2}{\alpha}}} - \frac{1}{1+e^{\frac{x_2 - x_1}{\alpha}}}$

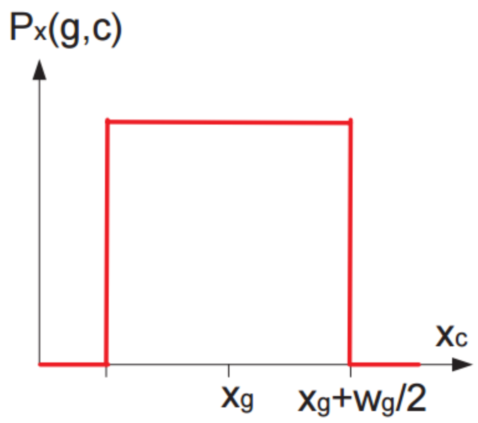
- when the net length $|x_1 - x_2|$ is relatively small compared to α, the partial gradient is close to 0; otherwise, the gradient is close to 1 or -1. It means that the length of long nets (relative to α) will be minimized more efficiently



Bell-Shaped Potential Function

Natural density function $D_g(x, y)$ is also not smooth or differentiable

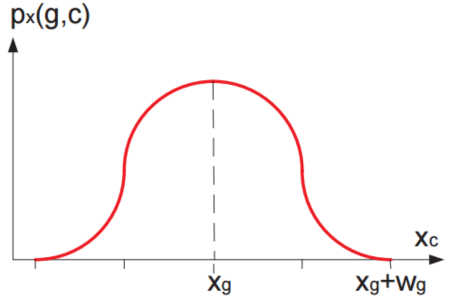
It can be expressed as $D_g(x, y) = \sum_v P_x(g, v) \cdot P_y(g, v)$ where $P_x(g, v)$ and $P_y(g, v)$ denote overlap between grid bin g and cell v along x , y directions



$P_x(g, c)$ smoothening:

$$\begin{cases} 1 - 0.5\, d_x^2 / w_g^2 \ \left(0 \le d_x \le w_g\right) \\ 0.5(d_x - 2w_g)^2 / w_g^2 \ (w_g \le d_x \le 2w_g) \end{cases}$$

Where $d_x = |x_c - x_g|$ is horizontal distance between cell c and grid g and w_g is grid bin width



Smooth density function:

$$SD_g(x, y) = \sum_v P_x(g, v) \cdot P_y(g, v)$$

Quadratic penalty method

- Solve the constrained optimization problem as a sequence of unconstrained minimization problem of the form
- $\min WL(x,y) + \frac{1}{2\mu}\sum_g (SD_g\ (x,y) - D_g\ )^2$
- For a sequence of values $\mu = \mu_k \to 0$
- Use the solution of previous unconstrained problem as initial solution for the next one
- Unconstrained problem can be solved using Conjugate Gradient method

Congestion directed placement

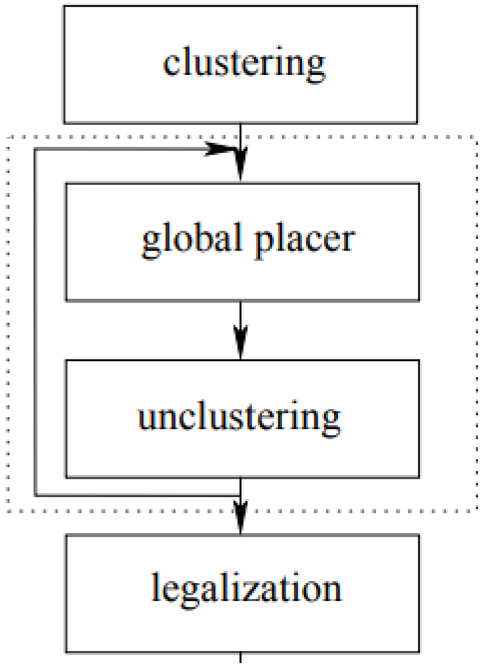
- If a particular grid is determined to be congested (resp. uncongested), the expected total cell potential of the grid is reduced (resp. increased) accordingly.
- For example, expected cell potential can be adjusted as follows:
- $D_g \propto 1 + \gamma(1 - 2\frac{CongestionDemand(g)}{max_g\{CongestionDemand(g)\}})$
- where γ is the congestion adjustment factor and decides the extent of congestion-directed placement.

Timing driven placement

- The basic idea is to put a higher weight for nets that are more timing critical
- $w(e) = 1 + \sum_{e \in \pi}(D(slack(\pi), T) - 1)$
- Where
  - $T = (1 - u)max_{\pi}\{delay(\pi)\}$
  - $slack(\pi) = T - delay(\pi)$ – the slack of a timing critical path π
  - $D(s, T) = \begin{cases} (1 - s/T)^{\delta}, s \leq 0 \\ 1, \qquad\qquad\quad s \geq 0 \end{cases}$
  - δ – criticality exponent

Multi-level algorithm

- Multiple level of clusters
  - Solve for each level of clusters and use the solution of the current level as initial solution for the next level placement problem
- Multiple levels of grids
  - Using an initial larger grid size and wirelength smoothing leads to better global optimization, and speeds up the placer



References

- Physical Design Handbook, Partitioning-based Methods for VLSI Placement , by Jarrod A. Roy and Igor L. Markov
- Handbook of Algorithms for Physical Design Automation. Chapter 16 by W. Swartz. Placement using simulated annealing
- Modern Circuit placement, Chapter 7, APlace: A High Quality, Large-Scale Analytical Placer by Andrew B. Kahng, Sherief Reda, Qinke Wang