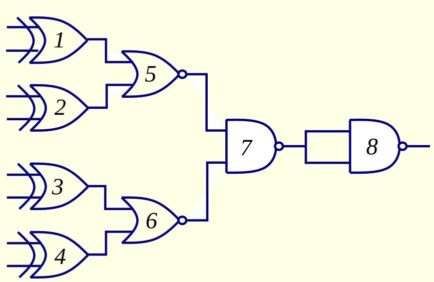
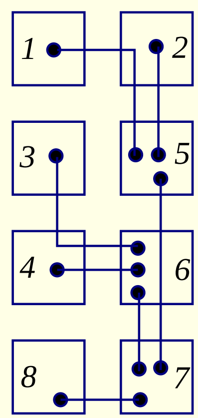
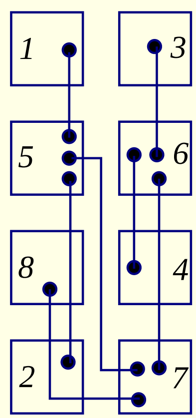
Placement

- The process of arranging the circuit components on a layout surface.
- Inputs: A set of fixed modules, a netlist.
- Goal: Find the best position for each module on the chip according to appropriate cost functions.
  - Considerations: **routability/channel density**, **wirelength**, cut size, performance, thermal issues, I/O pads.

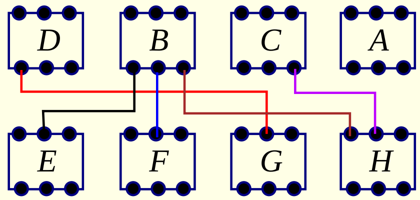




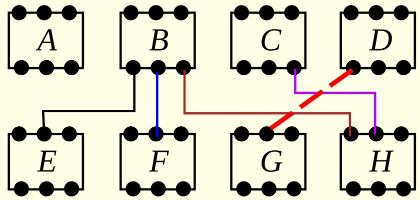
wirelength = 10



wirelength = 12



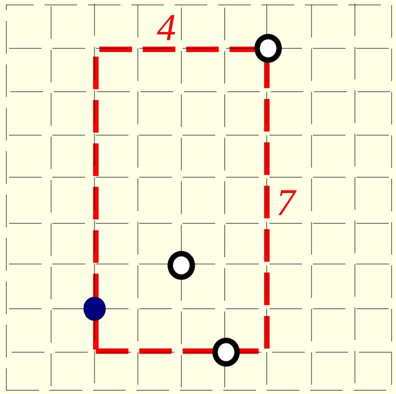
Density = 2 (2 tracks required)



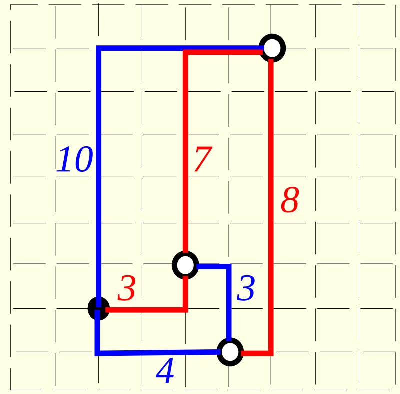
Shorter wirelength, 3 tracks required.

Estimation of Wirelength

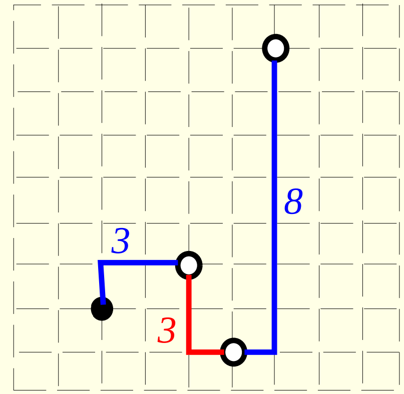
- **Semi-perimeter method:** Half the perimeter of the bounding rectangle that encloses all the pins of the net to be connected. Most widely used approximation!
- **Complete graph:** Since #edges in a complete graph ($\frac{n(n-1)}{2}$) is $\frac{n}{2}\times$ # of tree edges $(n-1)$, $wirelength \approx \frac{2}{n} \sum_{(i,j) \in net} dist(i,j)$.
- **Minimum chain:** Start from one vertex and connect to the closest one, and then to the next closest, etc.
- **Source-to-sink connection:** Connect one pin to all other pins of the net. Not accurate for uncongested chips.
- **Steiner-tree approximation:** Computationally expensive.
- **Minimum spanning tree**



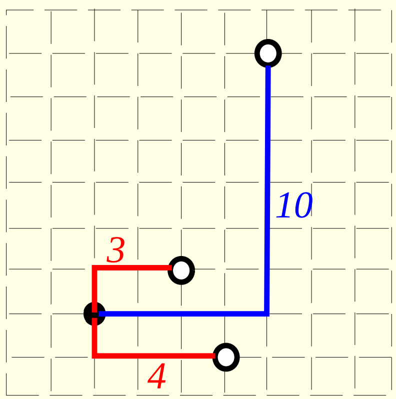
semi−perimeter len = 11



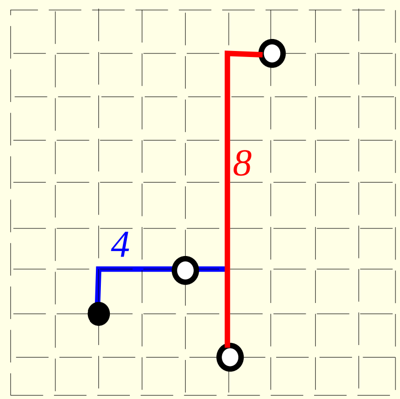
*complete graph len * 2/n = 17.5*



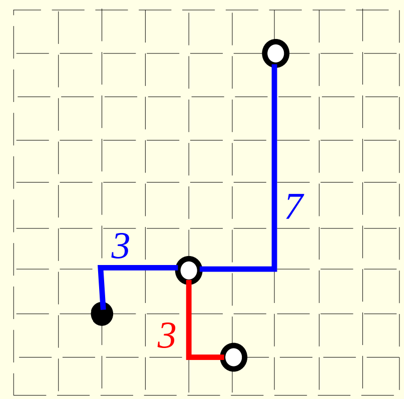
chain len = 14



source−to−sink len = 17



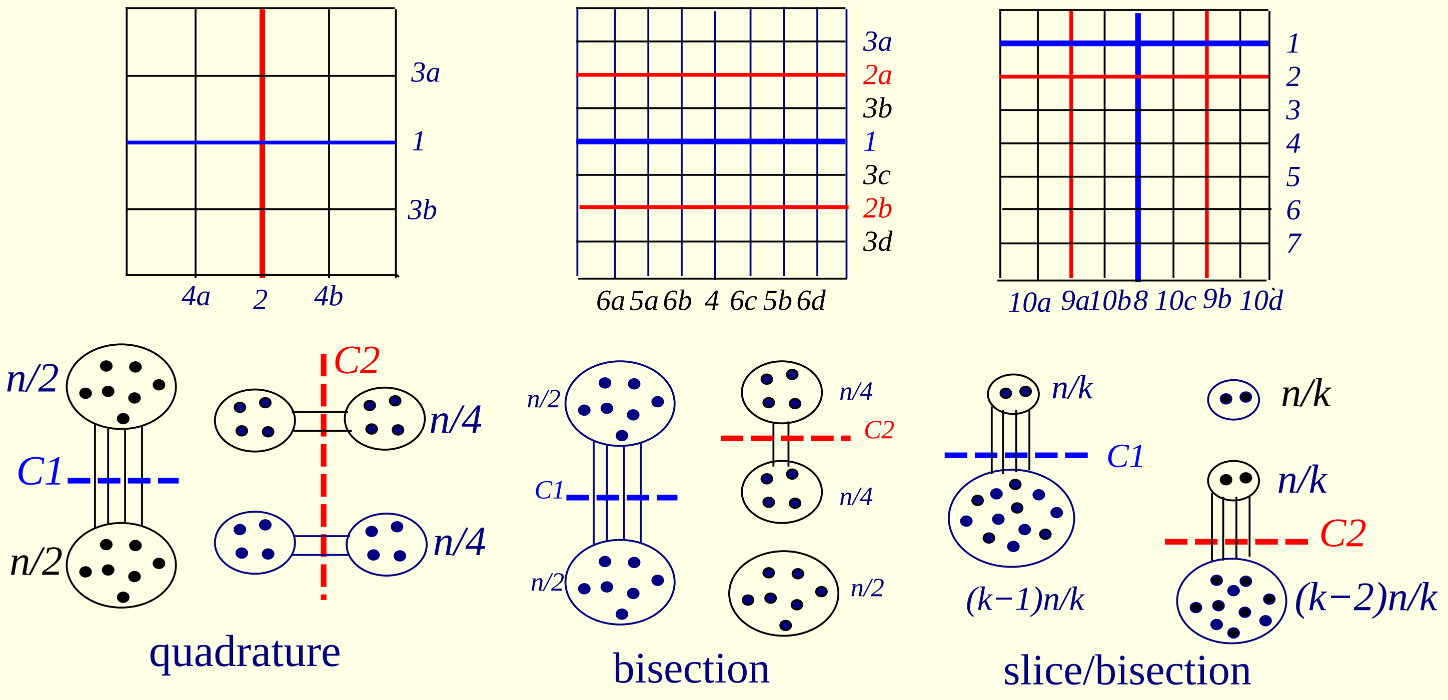
Steiner tree len = 12



Spanning tree len = 13

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.



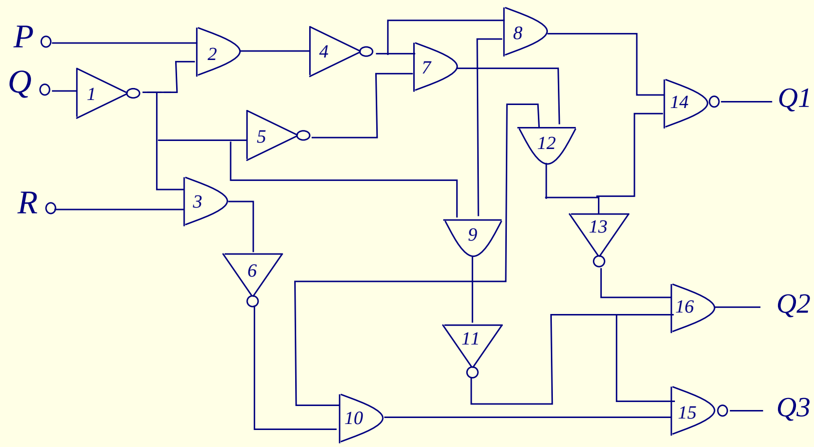
Algorithm for Min-Cut Placement

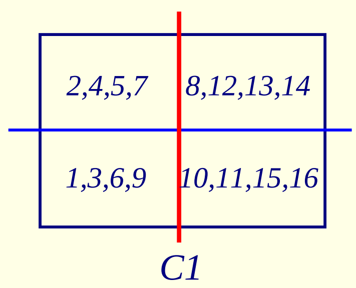
Algorithm: Min_Cut_Placement(N, n, C)
/* N: the layout surface */
/* n: # of cells to be placed */
/* n_0: # of cells in a slot */
/* C: the connectivity matrix */

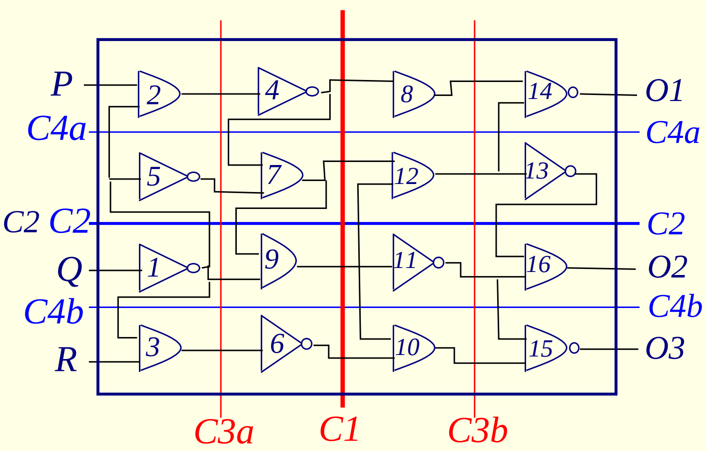
begin
if $(n \leq n_0)$ **then** PlaceCells(N, n, C);
else
  (N_1, N_2) $\leftarrow$ CutSurface(N);
  (n_1, C_1), (n_2, C_2) $\leftarrow$ Partition(n, C);
  Call Min_Cut_Placement(N_1, n_1, C_1);
  Call Min_Cut_Placement(N_2, n_2, C_2);
end

Quadrature Placement Example

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

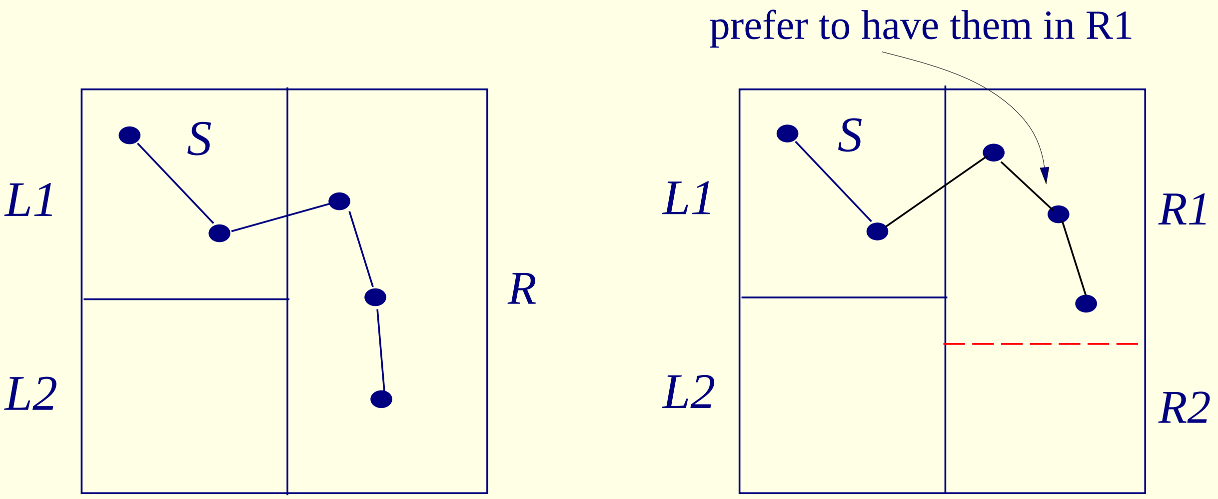






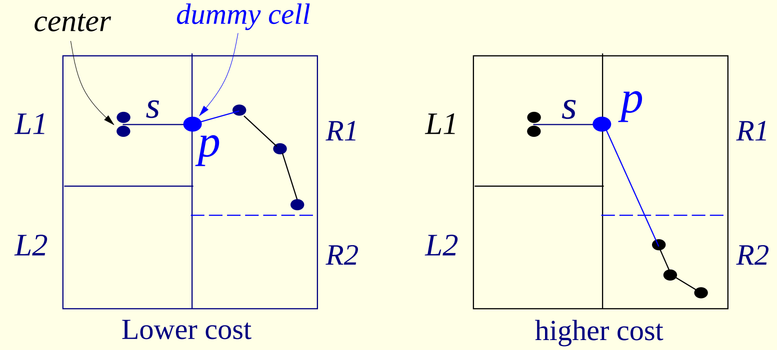
Min-Cut Placement with Terminal Propagation

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



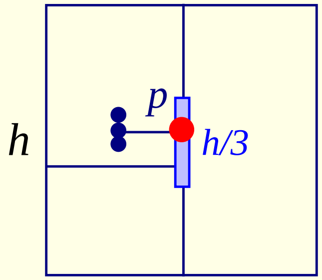
Terminal Propagation

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

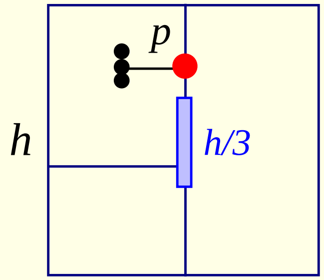


P will stay in R1 for the rest of partitioning!

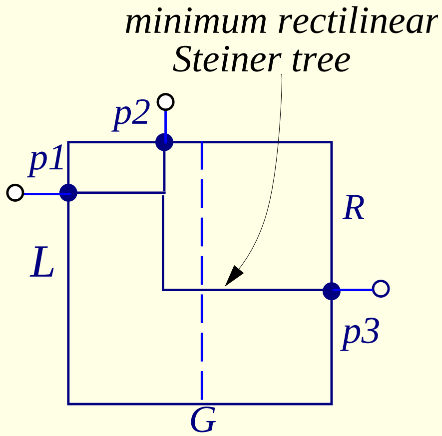
- When not to use p to bias partitioning? Net s has cells in many groups?



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

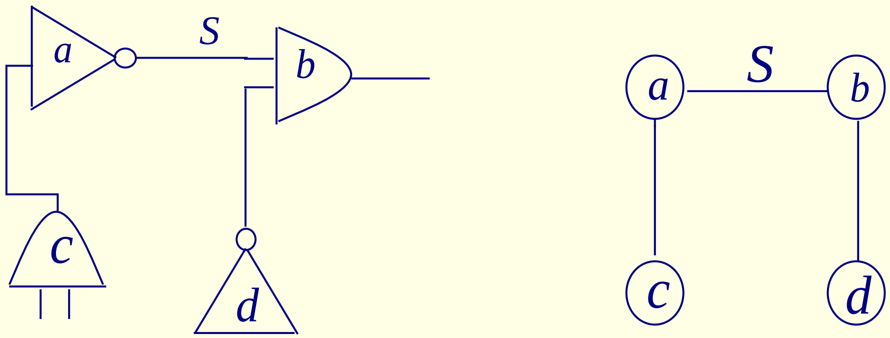


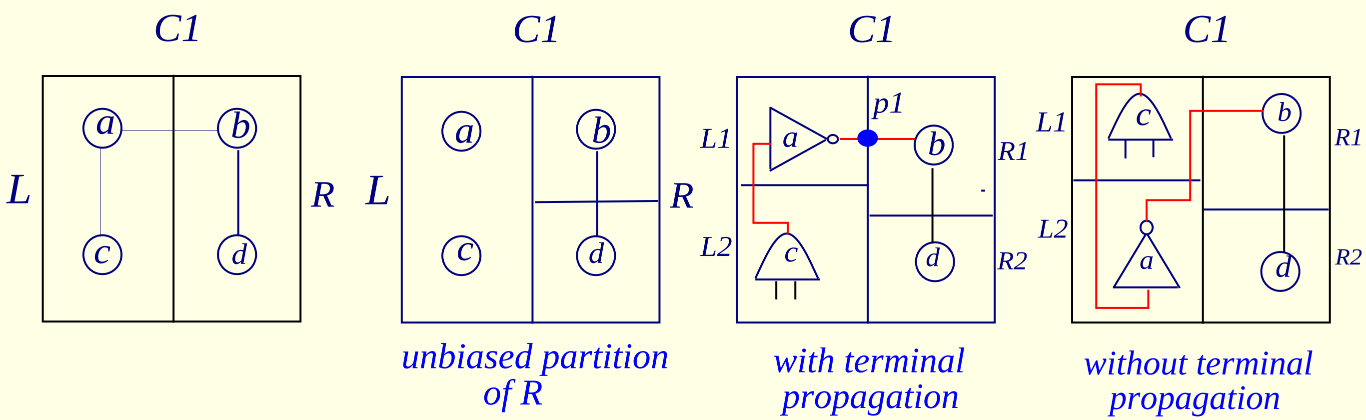
Use p!



Terminal Propagation Example

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



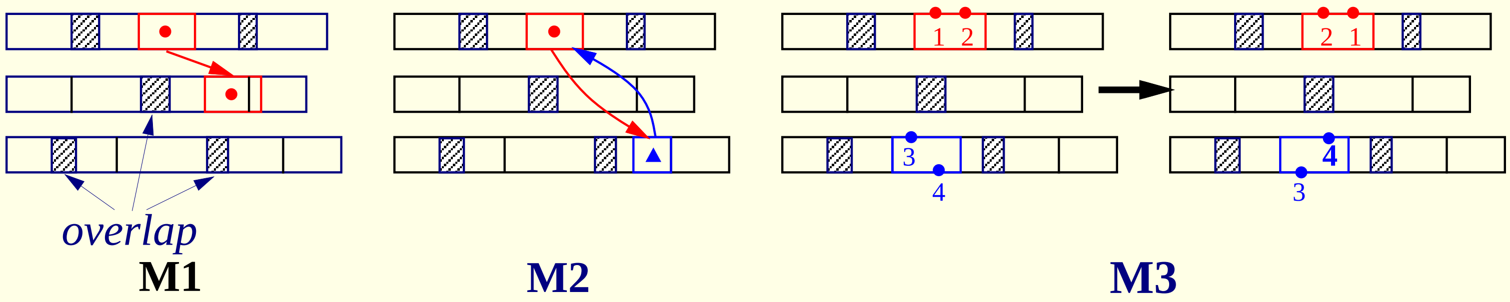


Placement by Simulated Annealing

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

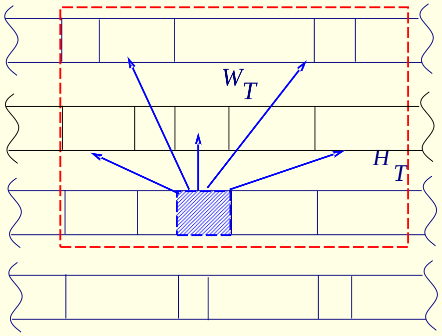
Solution Space & Neighborhood Structure

- **Solution Space:** All possible arrangements of the modules into rows, possibly with overlaps.
- **Neighborhood Structure:** 3 types of moves
  - M_1: Displace a module to a new location.
  - M_2: Interchange two modules.
  - M_3: Change the orientation of a module.



Neighborhood Structure

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



Cost Function

- Cost function: $C = C_1 + C_2 + C3$.
- C_1: **total estimated wirelength**.
  - $C_1 = \sum_{i \in Nets} (\alpha_i w_i + \beta_i h_i)$
  - α_i, β_i are horizontal and vertical weights, respectively. ($\alpha_i = 1, \beta_i = 1 \Rightarrow \frac{1}{2}\times$ perimeter of the bounding box of Net i.)
  - Critical nets: Increase both α_i and β_i.
  - If vertical wirings are "cheaper" than horizontal wirings, use smaller vertical weights: $\beta_i < \alpha_i$.
- C_2: **penalty function for module overlaps.**
  - $C_2 = \gamma \sum_{i \neq j} O_{ij}^2$, γ: penalty weight.
  - O_{ij}: amount of overlaps in the x-dimension between modules i and j.
- C_3: **penalty function that controls the row length.**
  - $C_2 = \delta \sum_{r \in Rows} |L_r - D_r|$, δ: penalty weight.
  - D_r: desired row length.
  - L_r: sum of the widths of the modules in row r.

Annealing Schedule

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