Physical Design Automation

Speaker:

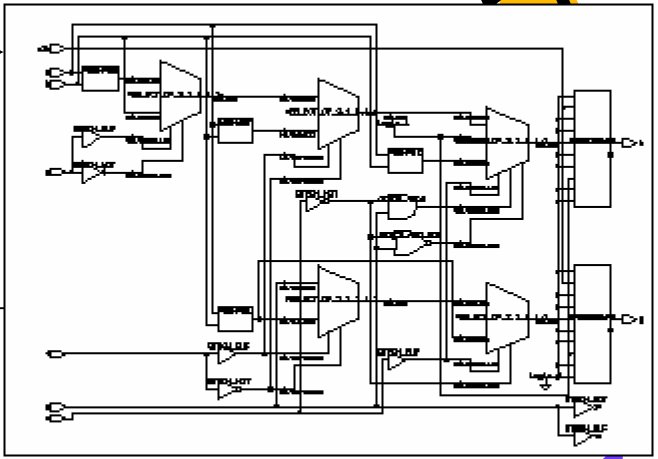
Debdeep Mukhopadhyay
Dept of Comp. Sc and Engg
IIT Madras, Chennai

Synthesis Flow

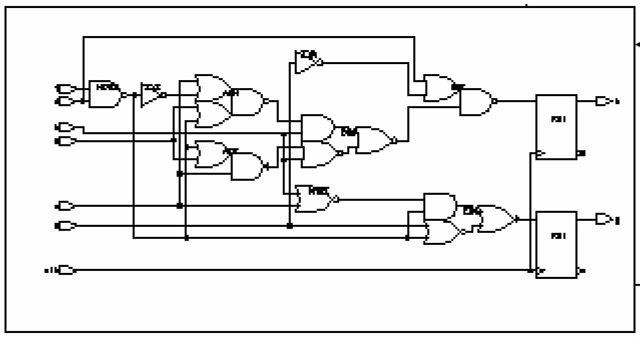
```
module example (clk, a, b, c, d, e, f, g, h);
input clk, a, b, c, d, e, f;
output g, h; reg g, h;

always @(posedge clk) begin
    g = a | b;
    if (d) begin
        if (e) h= a & ~b;
        else h= b;
        if (f) g= c; else h= a^ b;
        end else
            if (c) h= 1; else h= a ^ b;
        end
endmodule
```

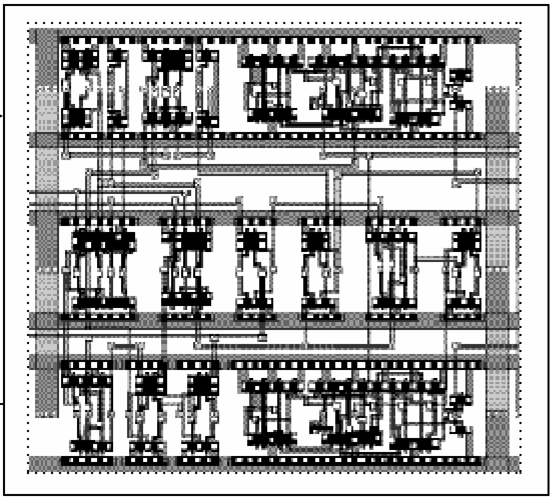
High-Level Synthesis



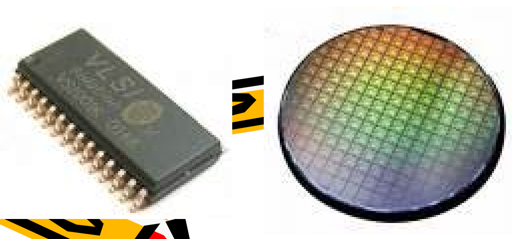
Logic Synthesis



Physical Design

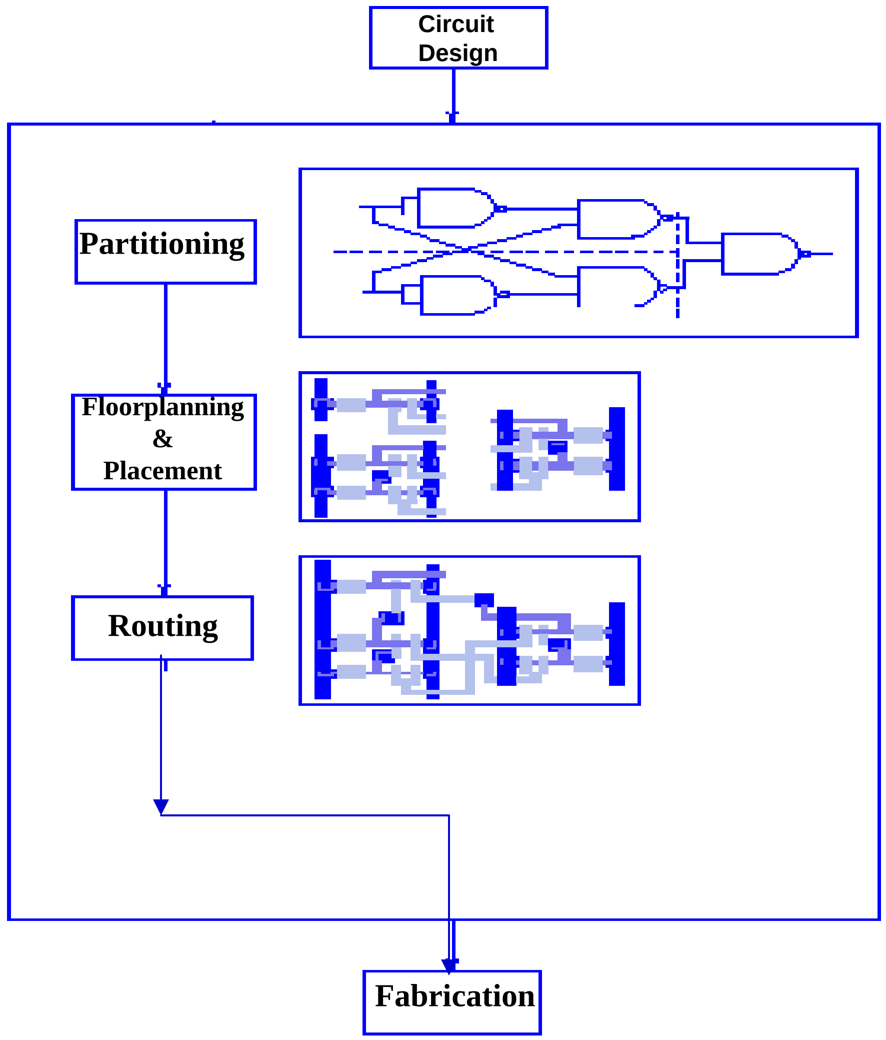


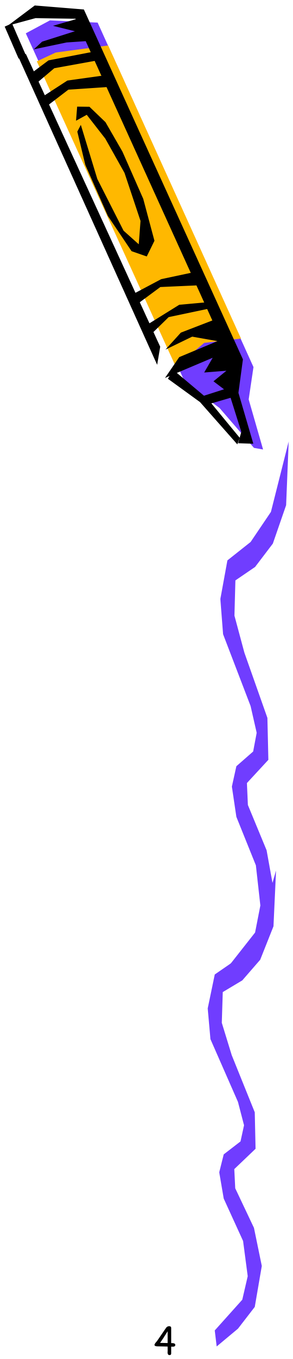
Fabrication and Packaging



Figures adopted with permission from Prof. Ciesielski, UMASS

Physical Design

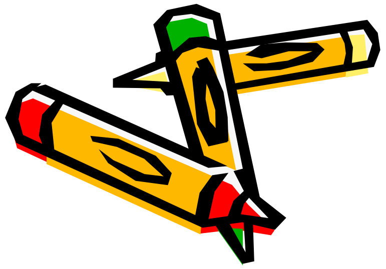




What is Backend?

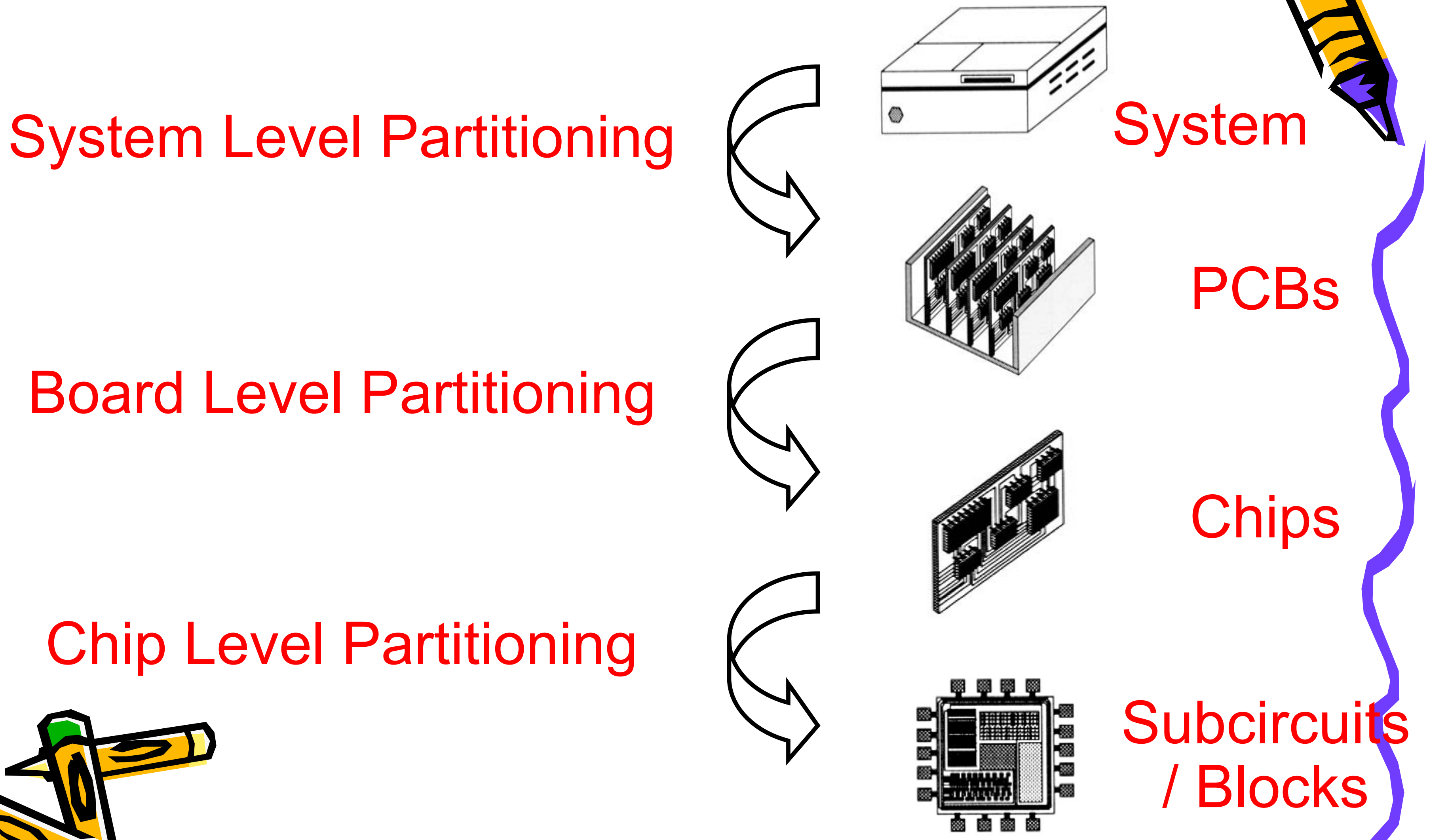
- Physical Design:

1. FloorPlanning : Architect's job
2. Placement : Builder's job
3. Routing : Electrician's job

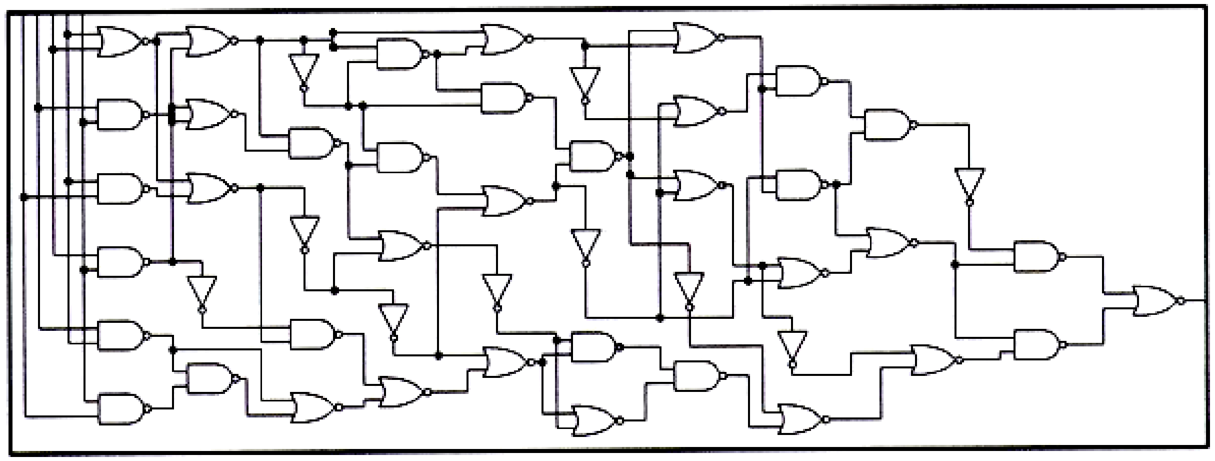


At sub-micron level

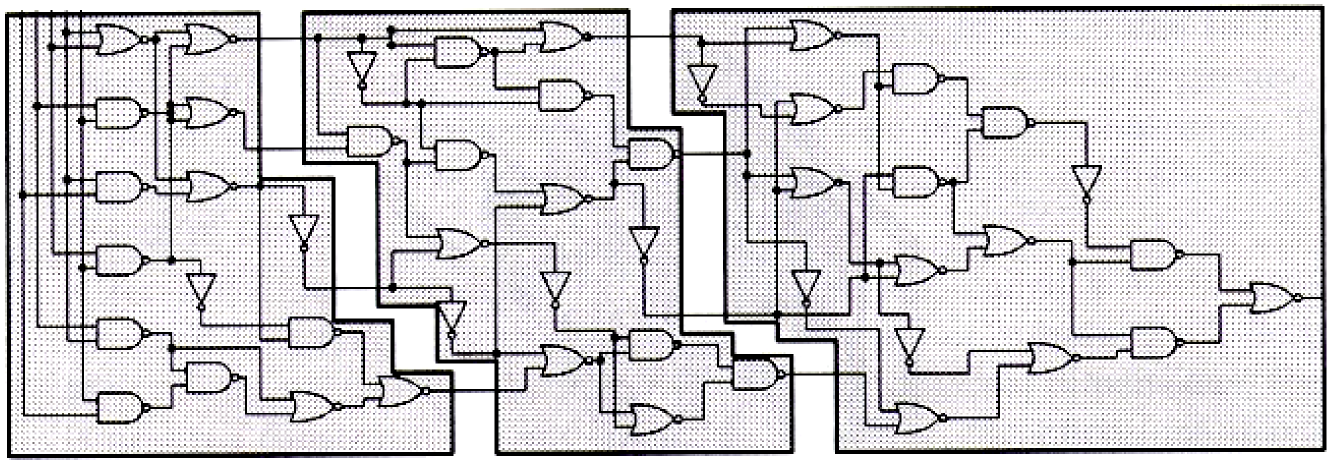
So, what is Partitioning?



Partitioning of a Circuit

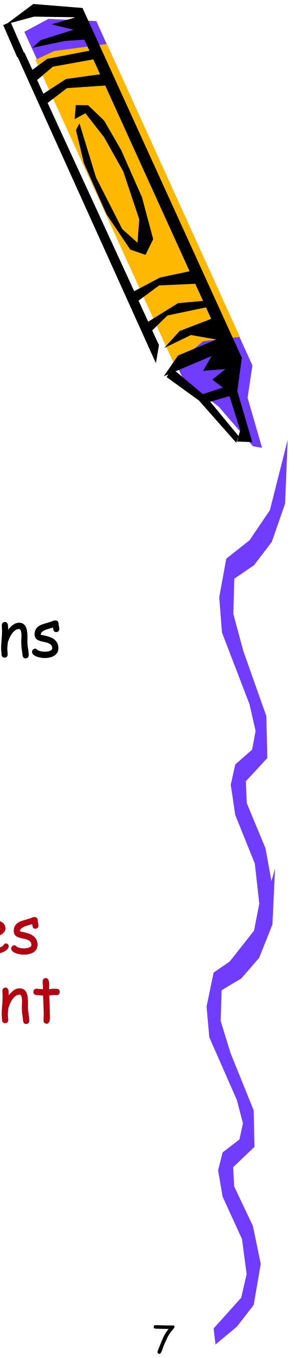


(a)

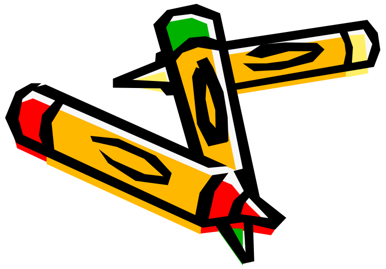


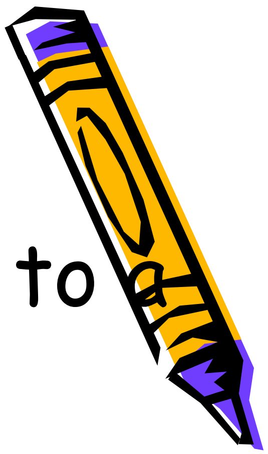
(b)

Why partition ?



- Ask Lord Curzon ☺
  - The most effective way to solve problems of high complexity : *Parallel CAD Development*
- System-level partitioning for multi-chip designs
  - Inter-chip interconnection delay dominates system performance
- IO Pin Limitation
- In deep-submicron designs, partitioning defines local and global interconnect, and has significant impact on circuit performance





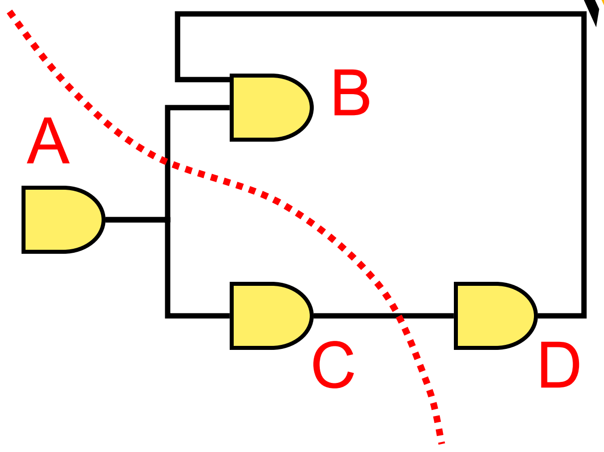
Objectives

- Since each partition can correspond to a chip, interesting objectives are:
  - Minimum number of partitions
    - Subject to maximum size (area) of each partition
  - Minimum number of interconnections between partitions
    - Since they correspond to off-chip wiring with more delay and less reliability
    - Less pin count on ICs (larger IO pins, much higher packaging cost)

Balanced partitioning given bound for area of each partition

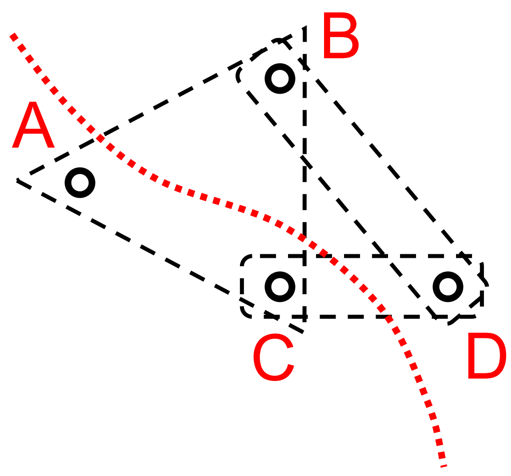
Circuit Representation

- Netlist:
  - Gates: A, B, C, D
  - Nets: {A,B,C}, {B,D}, {C,D}



- Hypergraph:
  - Vertices: A, B, C, D
  - Hyperedges: {A,B,C}, {B,D}, {C,D}

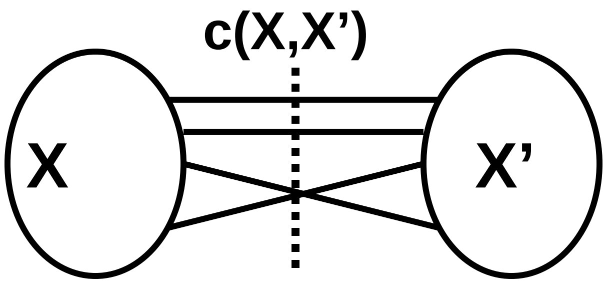
  - Vertex label: Gate size/area
  - Hyperedge label: Importance of net (weight)



Circuit Partitioning: Formulation

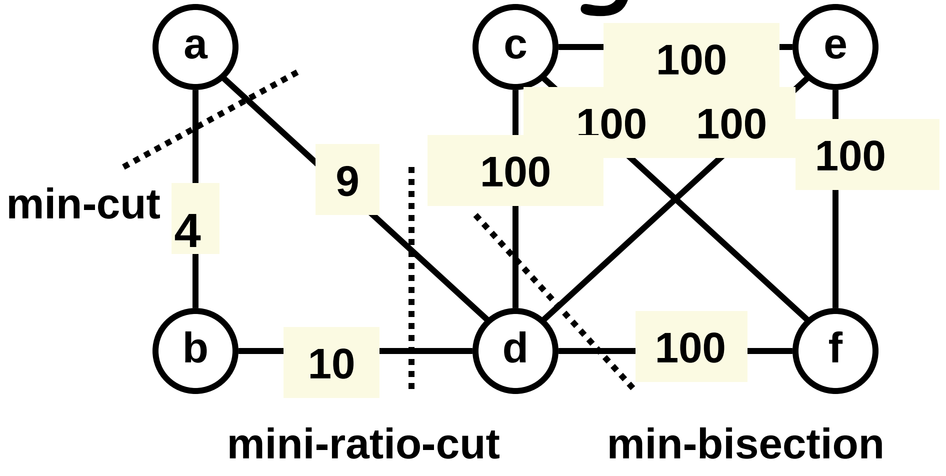
Bi-partitioning formulation:

Minimize interconnections between partitions



- Minimum cut: min c(x, x')
- minimum bisection: min c(x, x') with |x|= |x'|
- minimum ratio-cut: min c(x, x') / |x||x'|

A Bi-Partitioning Example



Min-cut size=13

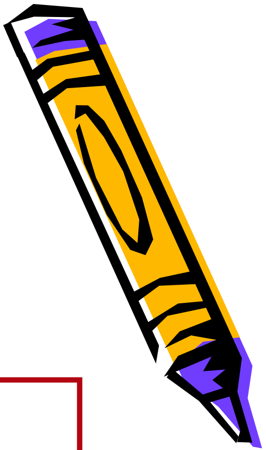
Min-Bisection size = 300

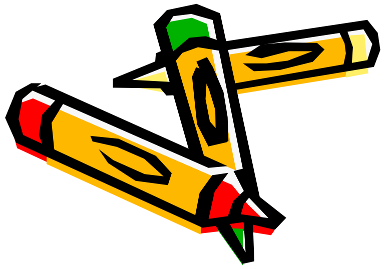
Min-ratio-cut size= 19

Ratio-cut helps to identify natural clusters

Iterative Partitioning Algorithms

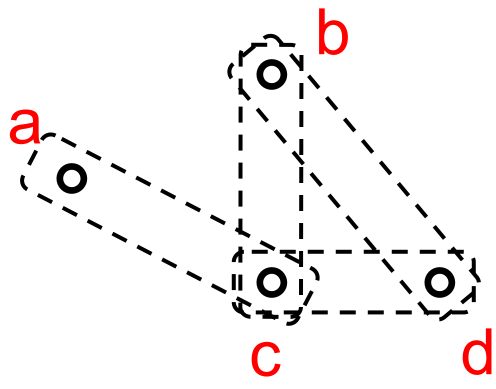
- Greedy iterative improvement method (Deterministic)
  - [Kernighan-Lin 1970]
- Simulated Annealing (Non-Deterministic)



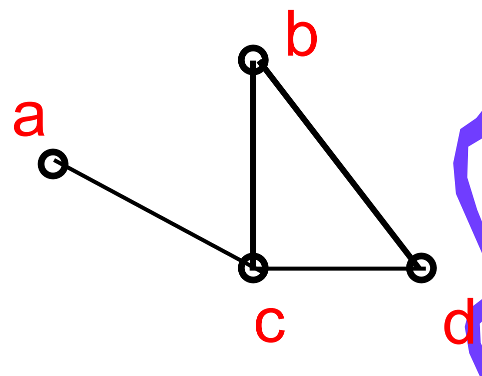


Restricted Partition Problem

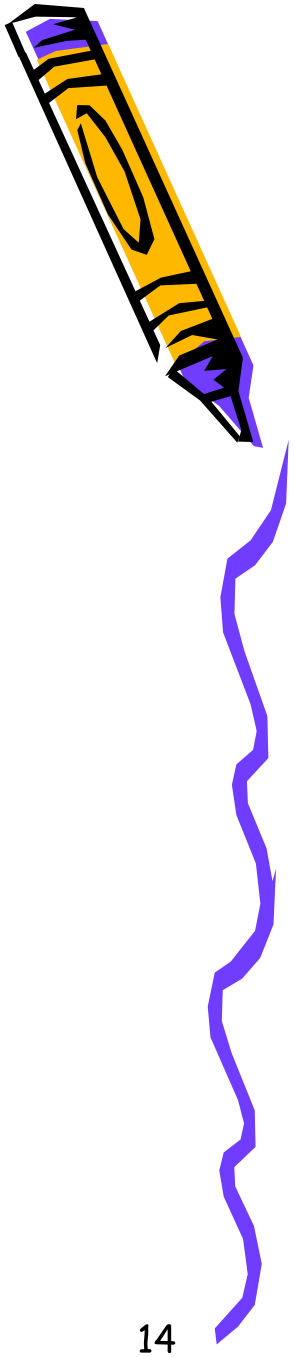
- Restrictions:
  - For Bisectioning of circuit
  - Assume all gates are of the same size
  - Works only for 2-terminal nets
- If all nets are 2-terminal, hypergraph → graph



Hypergraph Representation

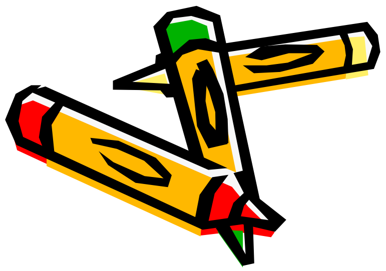


Graph Representation



Problem Formulation

- Input: A graph with
  - Set vertices V (|V| = 2n)
  - Set of edges E (|E| = m)
  - Cost c_{AB} for each edge {A, B} in E
- Output: 2 partitions X & Y such that
  - Total cost of edge cuts is minimized
  - Each partition has n vertices
- This problem is NP-Complete!!!!!

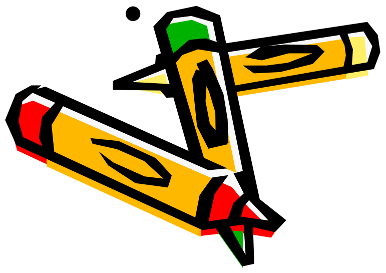


A Trivial Approach

- Try all possible bisections and find the best one
- If there are 2n vertices,

  # of possibilities = $(2n)! / n!^2 = n^{O(n)}$

- For 4 vertices (a,b,c,d), 3 possibilities
  1. X={a,b} & Y={c,d}
  2. X={a,c} & Y={b,d}
  3. X={a,d} & Y={b,c}
- For 100 vertices, 5×10^{28} possibilities
- Need 1.59×10^{13} years if one can try 100M possbilities per second

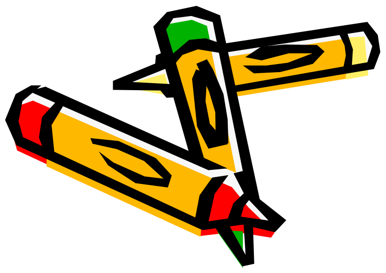


Definitions

- **Definition 1**: *Consider any node* a *in block* X. *The contribution of node* a *to the cutset is called the external cost of* a *and is denoted as* Ea, *where*

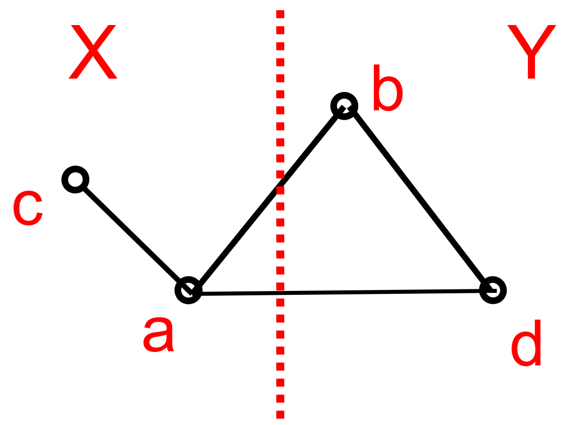
  $E_a = \Sigma c_{av}$ (for all v in Y)
- **Definition 2:** *The internal cost* Ia *of node* a in X *is defined as follows:*

  $I_a = \Sigma c_{av}$ (for all v in X)



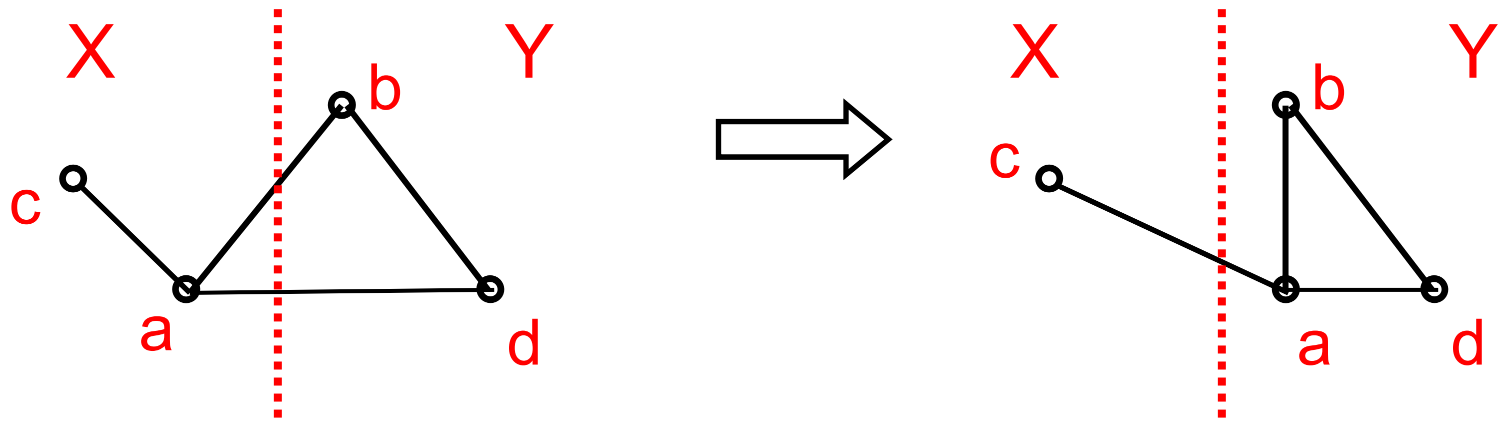
Example

- External cost (connection) $E_a = 2$
- Internal cost $I_a = 1$



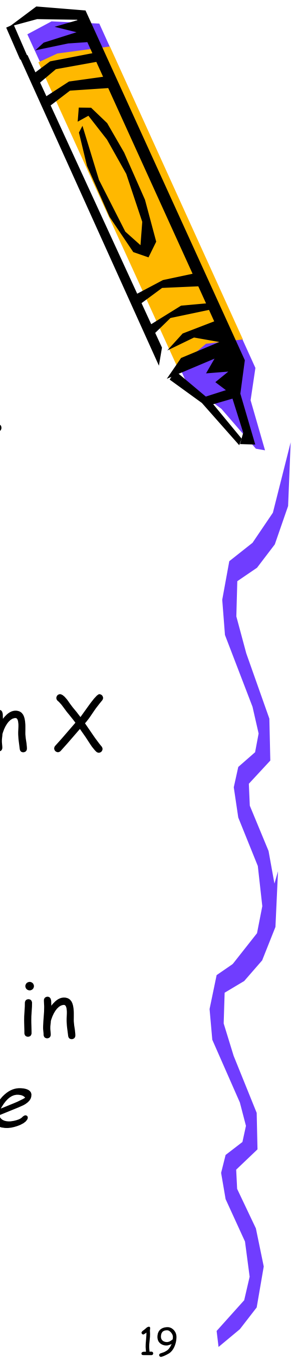
Idea of KL Algorithm

- D_a = Decrease in cut value if moving a = E_a-I_a
  - Moving node a from block X to block Y would decrease the value of the cutset by E_a and increase it by I_a



D_a = 2-1 = 1
D_b = 1-1 = 0

Useful Lemmas

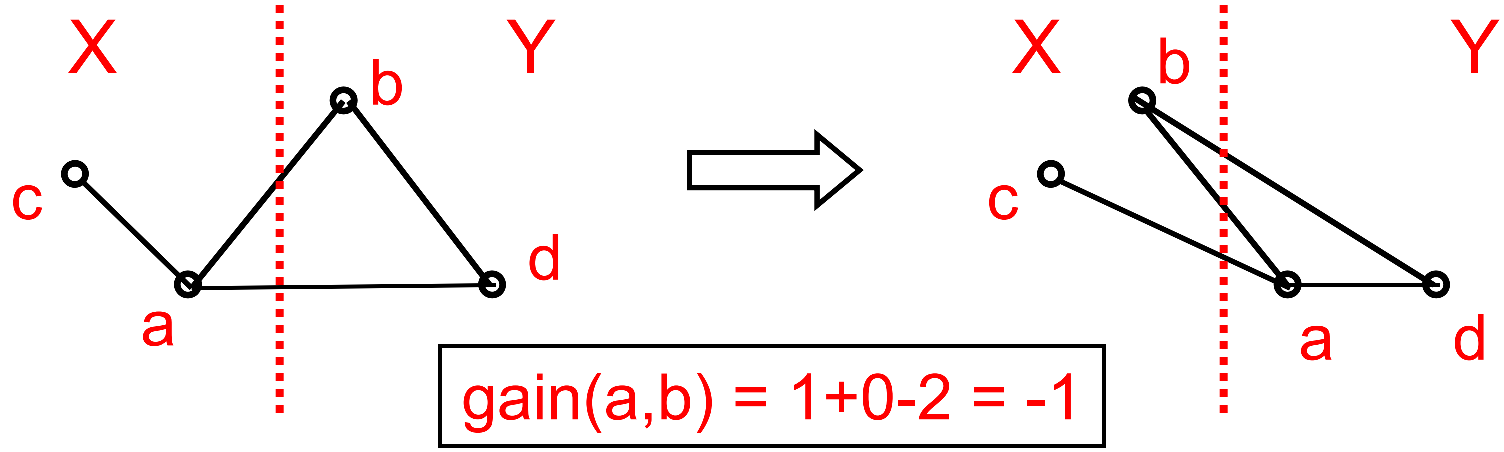


- To maintain balanced partition, we must move a node from Y to X each time we move a node from X to Y
- The effect of swapping two modules a in X with b in Y is characterized by the following lemma:
- **Lemma 1:** *If two elements* a in X *and* b in Y *are interchanged, the reduction in the cost is given by:*

$$\text{gain}(a,b) = g_{ab} = D_a + D_b - 2c_{ab}$$

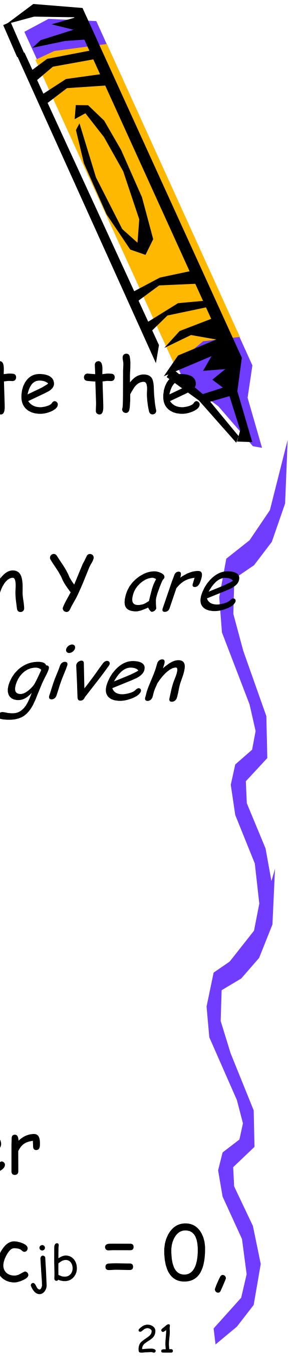
Example

- If switch a & b, gain(a,b) = $D_a+D_b-2c_{ab}$
  - c_{ab}: edge cost for ab



gain(a,b) = 1+0-2 = -1

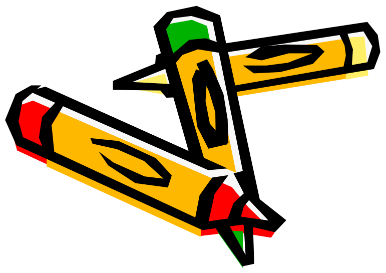
Useful Lemmas



- The following lemma tells us how to update the D- values after a swap.
- **Lemma 2:** *If two elements* a in X *and* b in Y *are interchanged, then the new* D-*values are given by*

  $D'_k = D_k + 2c_{ka} - 2c_{kb}$; for all k in X – {a}

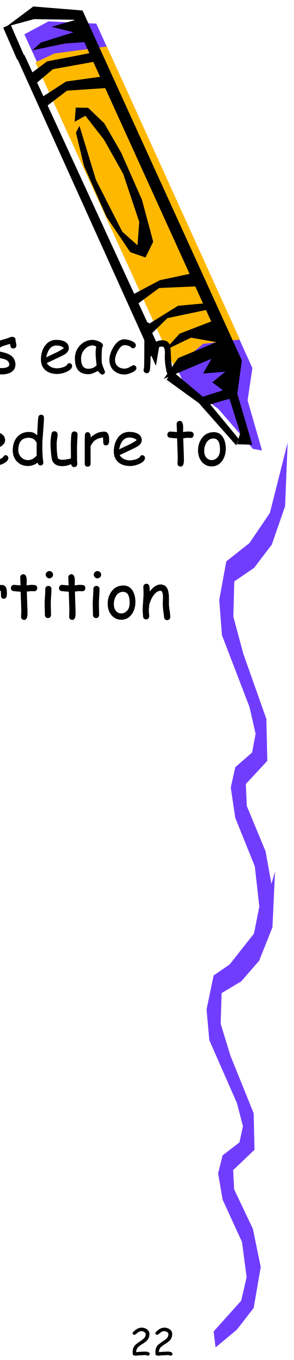
  $D'_m = D_m + 2c_{mb} - 2c_{ma}$; for all m in Y – {b}

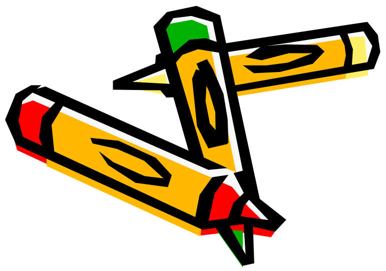


- Notice that if a module j is neither connected to a nor to b then $c_{ja} = c_{jb} = 0$, and, $D_j=D'_j$

Overview of KL Algorithm

- Start from an initial partition {X,Y} of n elements each
- Use lemmas 1 and 2 together with a greedy procedure to identify two subsets A in X, and B in Y, of equal cardinality, such that when interchanged, the partition cost is improved
- A and B may be empty, indicating in that case that the current partition can no longer be improved



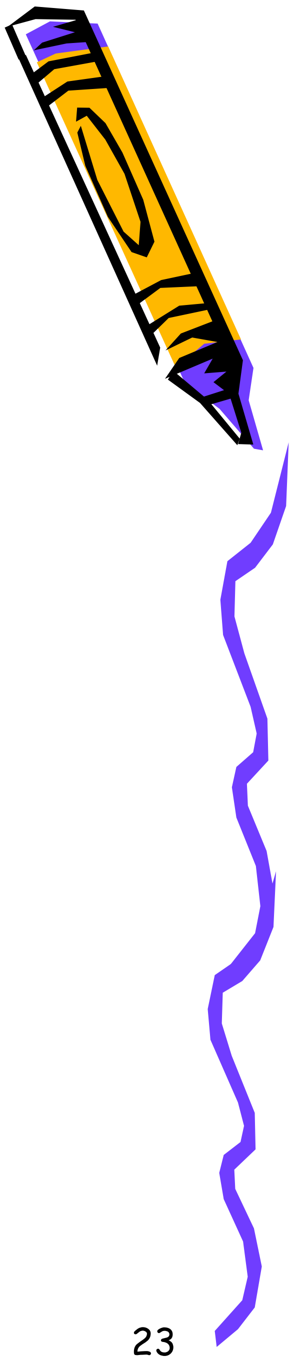


Idea of KL Algorithm

- Start with any initial legal partitions X and Y
- A pass (exchanging each vertex exactly once) is described below:

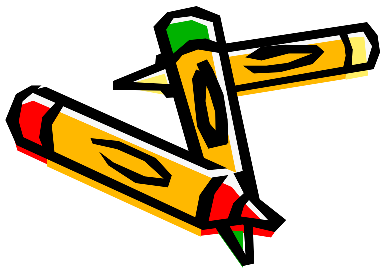
```
1. For i := 1 to n do
      From the unlocked (unexchanged) vertices,
        choose a pair (A,B) s.t. gain(A,B) is largest
      Exchange A and B. Lock A and B.
      Let $g_i$ = gain(A,B)
2. Find the k s.t. $G=g_1+...+g_k$ is maximized
3. Switch the first k pairs
```

- Repeat the pass until there is no improvement (G=0)



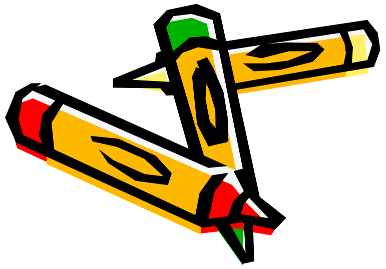
Greedy Procedure to Identify A, B at Each Iteration

1. Compute g_{ab} for all a in X and b in Y
2. Select the pair (a_1, b_1) with maximum gain g_1 and lock a_1 and b_1
3. Update the D-values of remaining free cells and recompute the gains
4. Then a second pair (a_2, b_2) with maximum gain g_2 is selected and locked. Hence, the gain of swapping the pair (a_1, b_1) followed by the (a_2, b_2) swap is $G_2 = g_1 + g_2$.



Greedy ….(contd.)

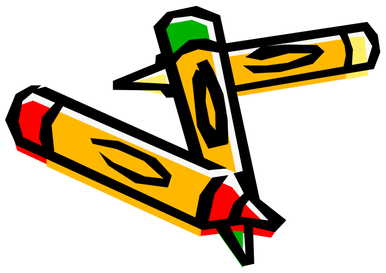
5. Continue selecting (a_3, b_3), … , (a_i, b_i), … , (a_n, b_n) with gains g_3, … , g_i, … , g_n
6. The gain of making the swap of the first k pairs is $G_k = g_1 + \ldots + g_k$. If there is no k such that $G_k > 0$ then the current partition cannot be improved; otherwise choose the k that maximizes Gk, and make the interchange of $\{a_1, a_2, \ldots, a_k\}$ with $\{b_1, b_2, \ldots, b_k\}$ permanent



Partitioning: Simulated Annealing

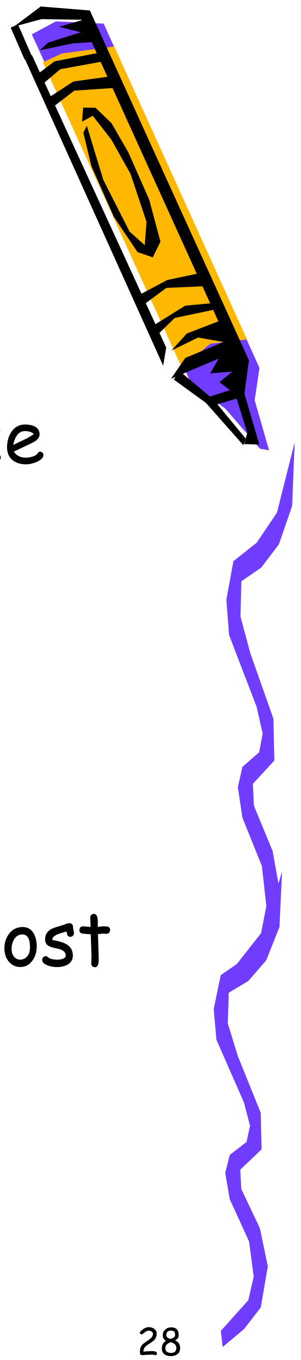
State Space Search Problem

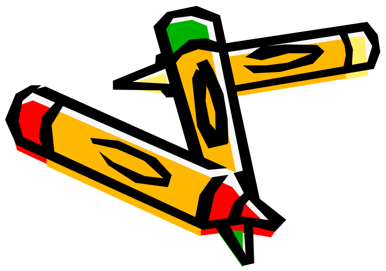
- Combinatorial optimization problems (like partitioning) can be thought as a State Space Search Problem.
- A State is just a configuration of the combinatorial objects involved.
- The State Space is the set of all possible states (configurations).
- A Neighbourhood Structure is also defined (which states can one go in one step).
- There is a cost corresponding to each state.
- Search for the min (or max) cost state.



Greedy Algorithm

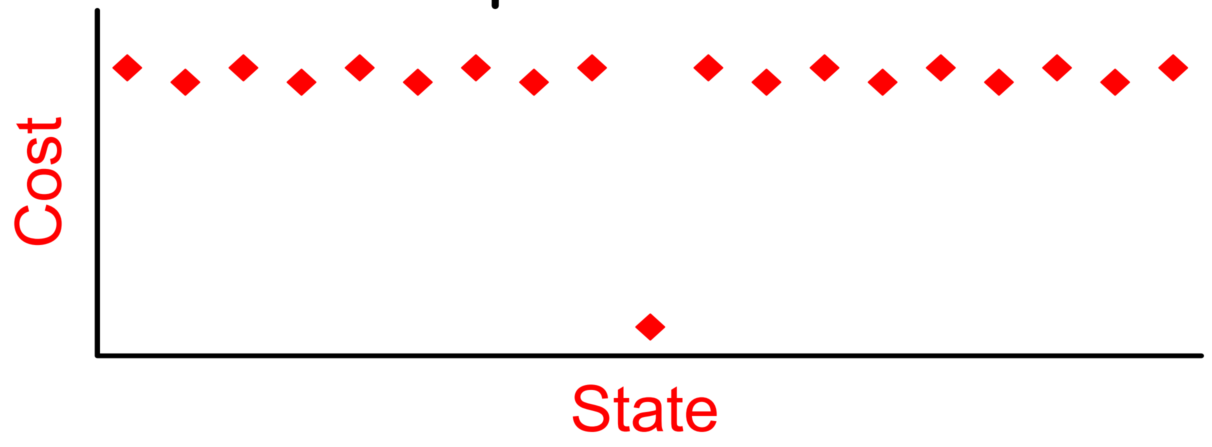
- A very simple technique for State Space Search Problem.
- Start from any state.
- Always move to a neighbor with the min cost (assume minimization problem).
- Stop when all neighbors have a higher cost than the current state.



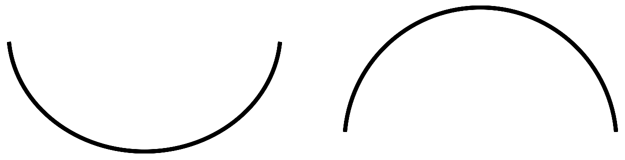


Problem with Greedy Algorithms

- Easily get stuck at local minimum.
- Will obtain non-optimal solutions.

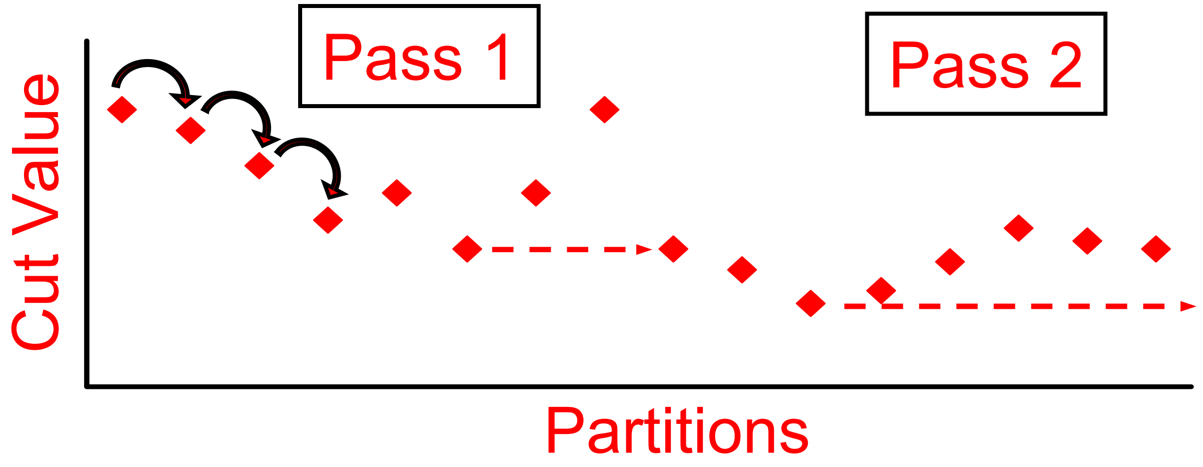


- Optimal only for convex (or concave for maximization) funtions.

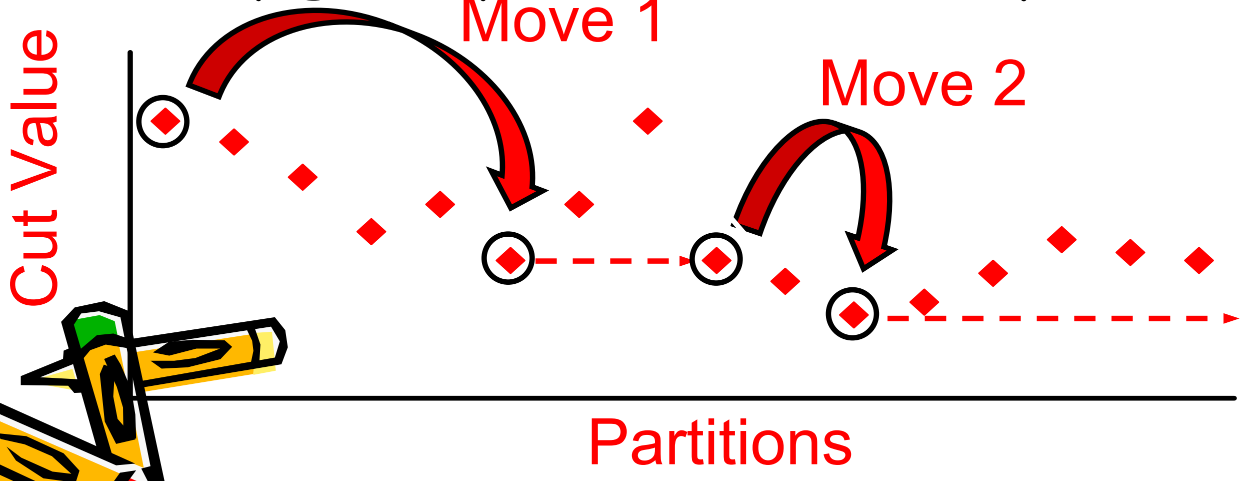


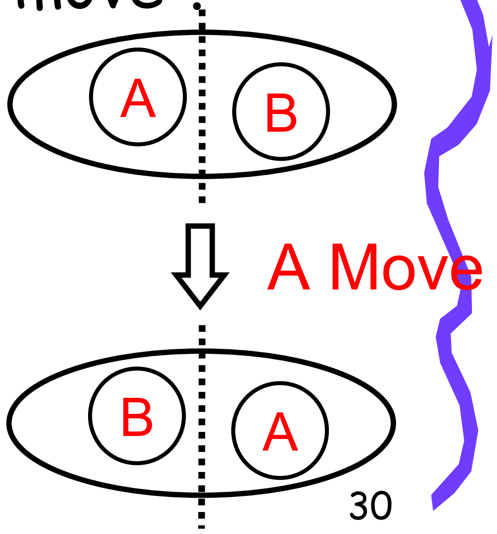
Greedy Nature of KL

- KL is *almost* greedy algorithms.

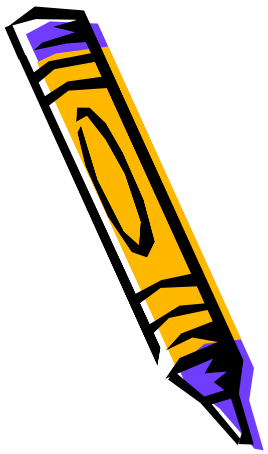


- Purely greedy if we consider a pass as a "move".

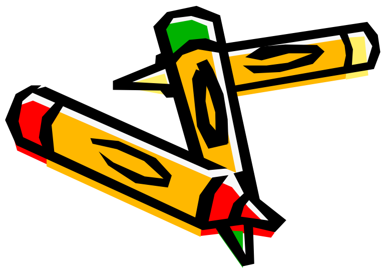




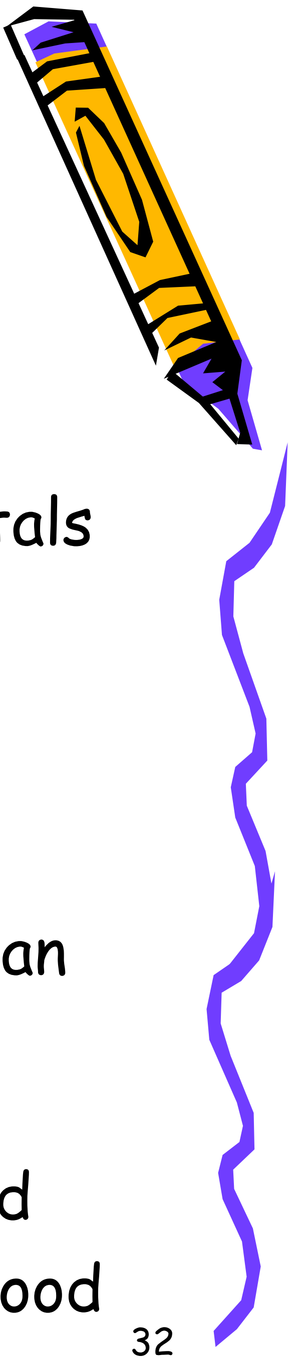
Simulated Annealing



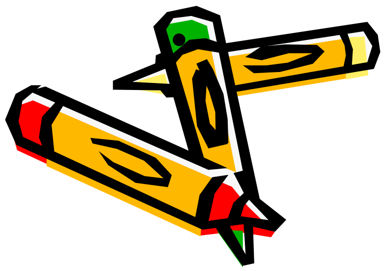
- Very general search technique.
- Try to avoid being trapped in local minimum by making probabilistic moves.
- Popularize as a heuristic for optimization by:
  - Kirkpatrick, Gelatt and Vecchi, "Optimization by Simulated Annealing", Science, 220(4598):498-516, May 1983.



Basic Idea of Simulated Annealing

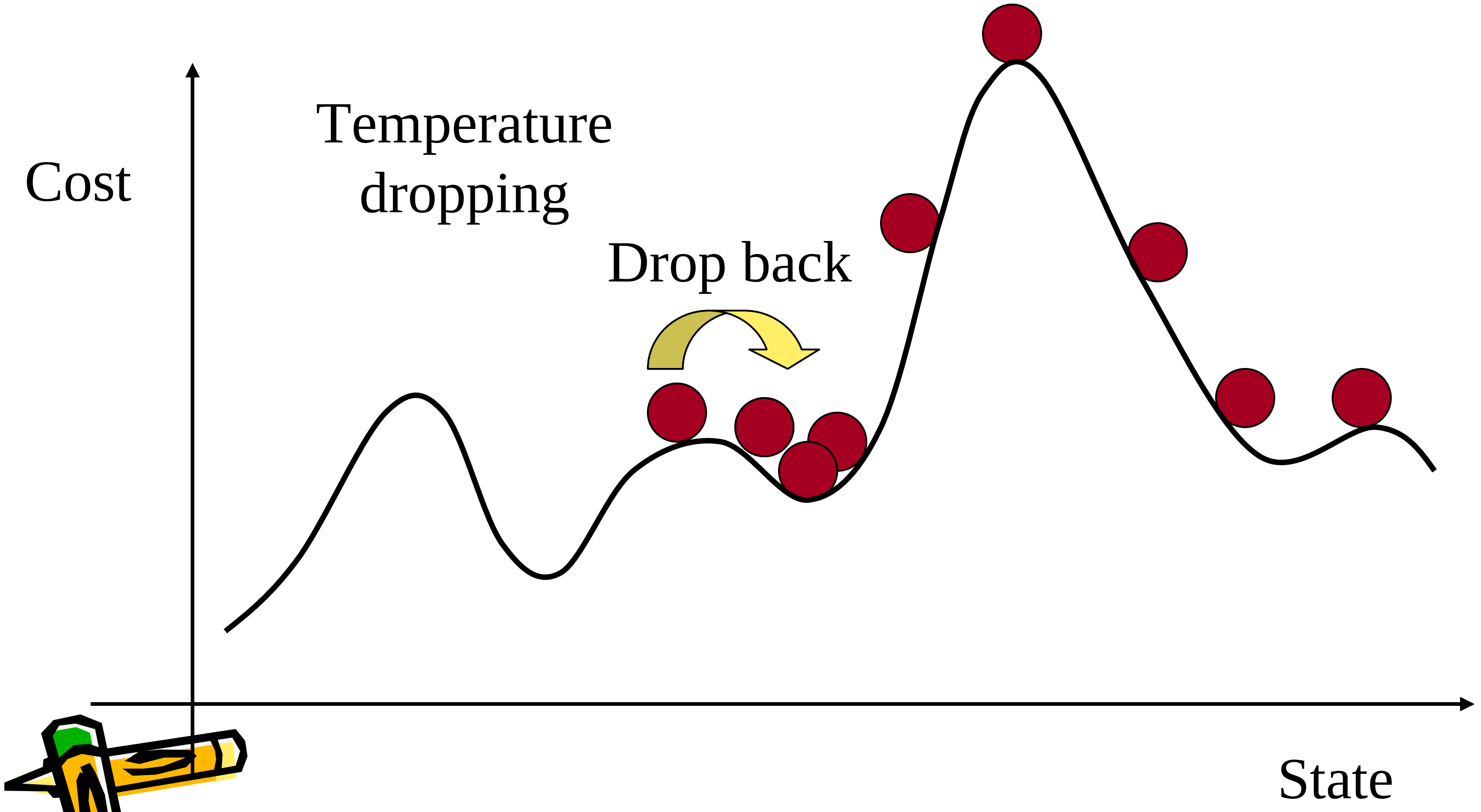


- Inspired by the *Annealing Process*:
  - The process of carefully cooling molten metals in order to obtain a good crystal structure.
  - First, metal is heated to a very high temperature.
  - Then slowly cooled.
  - By cooling at a proper rate, atoms will have an increased chance to regain proper crystal structure.

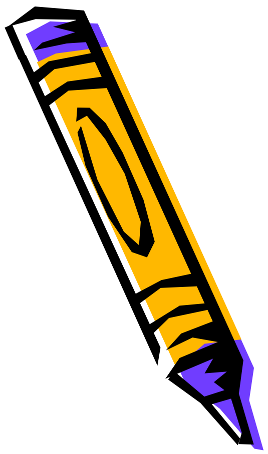


Attaining a min cost state in simulated annealing is analogous to attaining a good crystal structure in annealing.

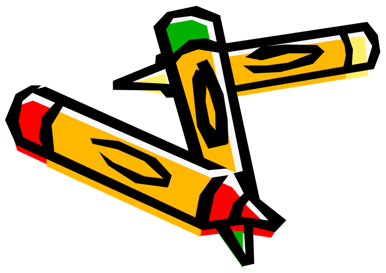
Simulated Annealing



The Simulated Annealing Procedure



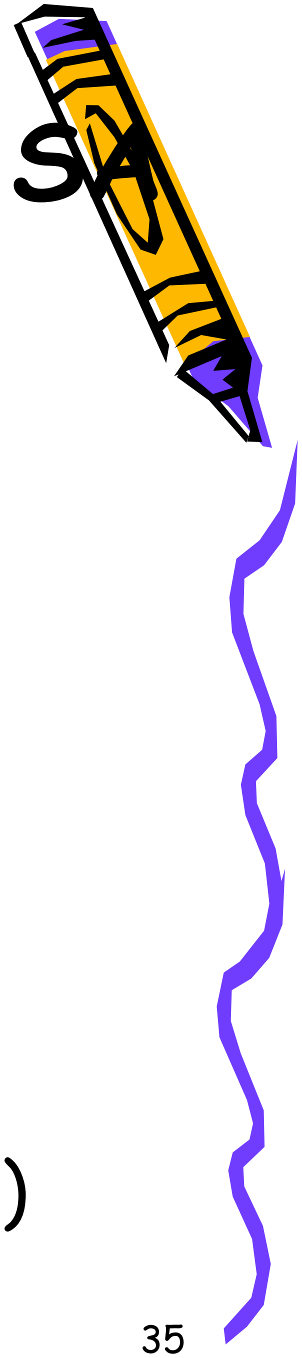
```
Let t be the initial temperature.
Repeat
    Repeat
      – Pick a neighbor of the current state randomly.
      – Let c = cost of current state.
        Let c' = cost of the neighbour picked.
      – If $c' < c$, then move to the neighbour (downhill
        move).
      – If $c' > c$, then move to the neighbour with
        probablility $e^{-(c'-c)/t}$ (uphill move).
    Until equilibrium is reached.
          Reduce t according to cooling schedule.
          Until Freezing point is reached.
```

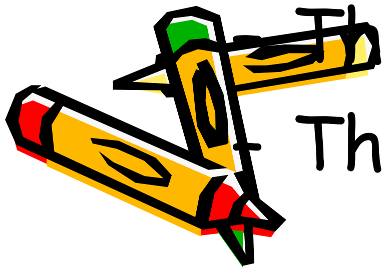


Things to decide when using SA

- When solving a combinatorial problem,

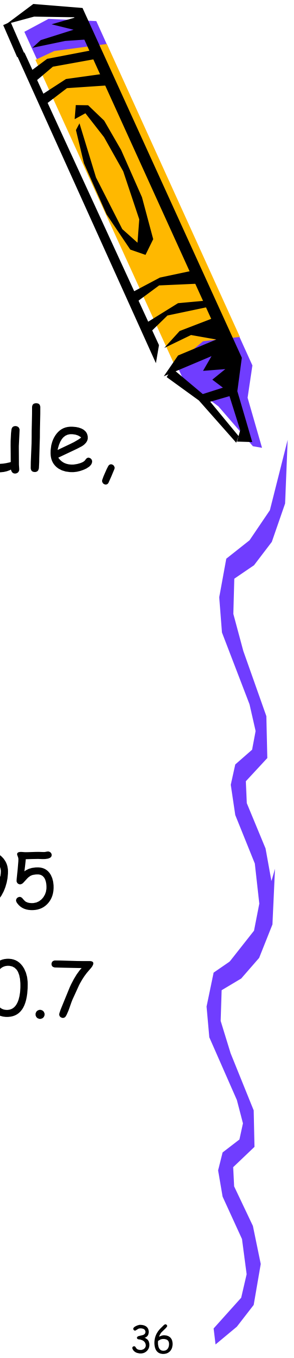
  we have to decide:
  - The state space
  - The neighborhood structure
  - The cost function
  - The initial state
  - The initial temperature
  - The cooling schedule (how to change t)
  - The freezing point

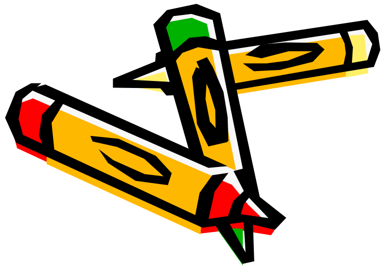




Common Cooling Schedules

- Initial temperature, Cooling schedule, and freezing point are usually experimentally determined.
- Some common cooling schedules:
  - $t = \alpha t$, where α is typically around 0.95
  - $t = e^{-\beta t}\, t$, where β is typically around 0.7
  - ......



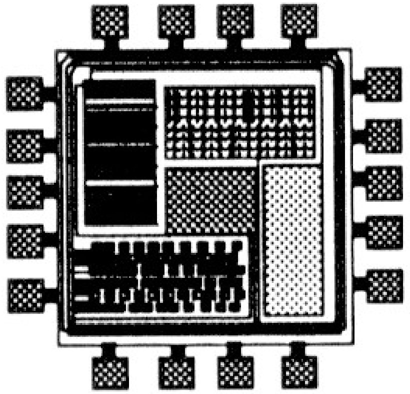


Hierarchical Design

- Several blocks after partitioning:

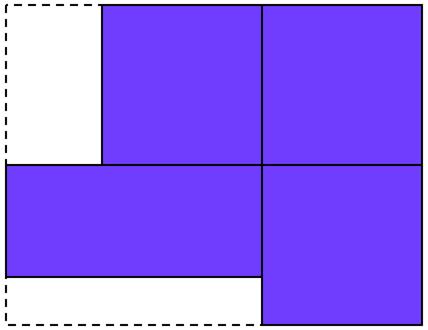
- Need to:
  - Put the blocks togeth
  - Design each block.

  Which step to go first?



Hierarchical Design

- How to put the blocks together without knowing their shapes and the positions of the I/O pins?
- If we design the blocks first, those blocks may not be able to form a tight packing.



Floorplanning

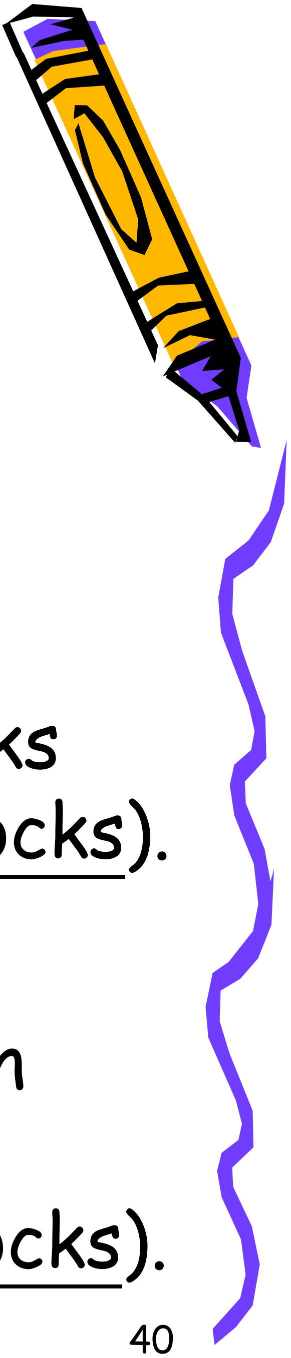
The floorplanning problem is to plan the *positions* and *shapes* of the modules at the beginning of the design cycle to optimize the circuit performance:

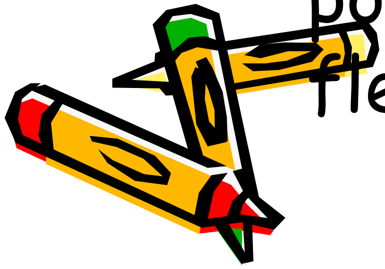
- chip area
- total wirelength
- delay of critical path
- routability

others, e.g., noise, heat dissipation, etc.

Floorplanning v.s. Placement

- Both determines block positions to optimize the circuit performance.
- Floorplanning:
  - Details like shapes of blocks, I/O pin positions, etc. are not yet fixed (blocks with flexible shape are called soft blocks).
- Placement:
  - Details like module shapes and I/O pin positions are fixed (blocks with no flexibility in shape are called hard blocks).





Floorplanning Problem

- Input:
  - n Blocks with areas $A_1, \ldots, A_n$
  - Bounds r_i and s_i on the aspect ratio of block B_i
- Output:
  - Coordinates (x_i, y_i), width w_i and height h_i for each block such that $h_i w_i = A_i$ and

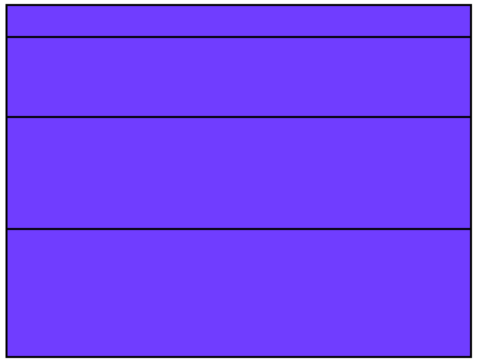
$$r_i \leq h_i/w_i \leq s_i$$

- Objective:
  - To optimize the circuit performance.

Bounds on Aspect Ratios

If there is no bound on the aspect ratios, can we pack everything tightly?

- Sure!

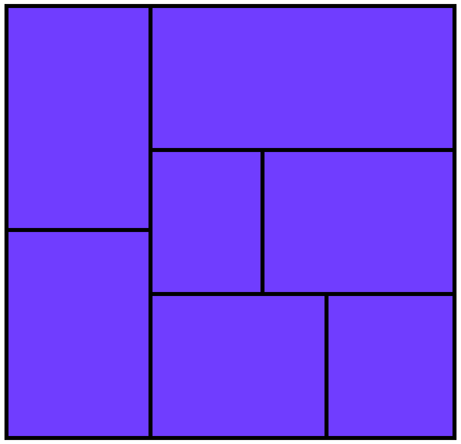


But we don't want to layout blocks as long strips, so we require

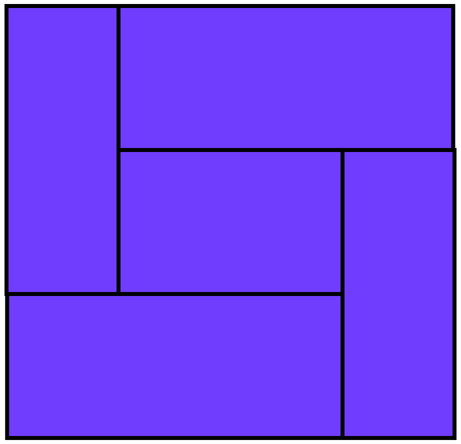
$$r_i \leq h_i/w_i \leq s_i \text{ for each } i.$$

Slicing and Non-Slicing Floorplan

- Slicing Floorplan:
  One that can be obtained by repetitively subdividing (slicing) rectangles horizontally or vertically.



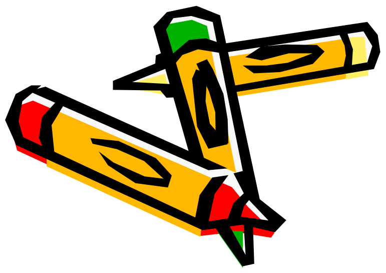
- Non-Slicing Floorplan:
  One that may not be obtained by repetitively subdividing alone.



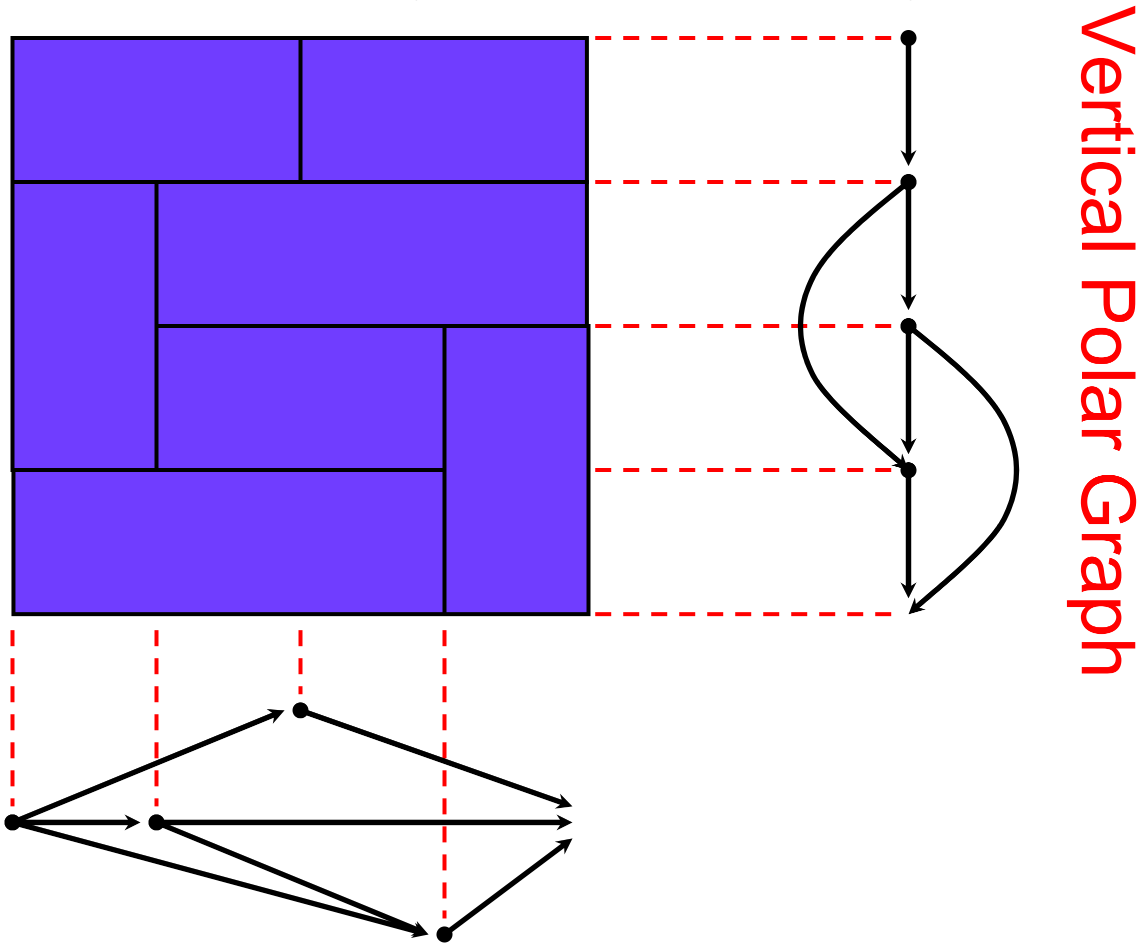
Polar Graph Representation

- A graph representation of floorplan.
- Each floorplan is modeled by a pair of directed acyclic graphs:
  - Horizontal polar graph
  - Vertical polar graph
- For horizontal (vertical) polar graph,
  - Vertex: Vertical (horizontal) channel
  - Edge: 2 channels are on 2 sides of a block
  - Edge weight: Width (height) of the block

Note: There are many other graph representations.



Polar Graph: Example



Horizontal Polar Graph

Simulated Annealing using Polish Expression Representation

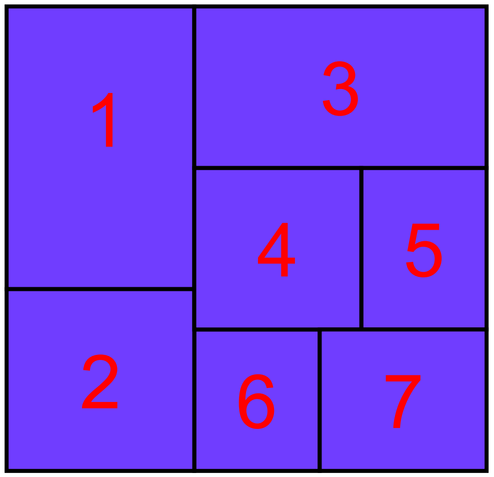
D.F. Wong and C.L. Liu,

"A New Algorithm for Floorplan Design"

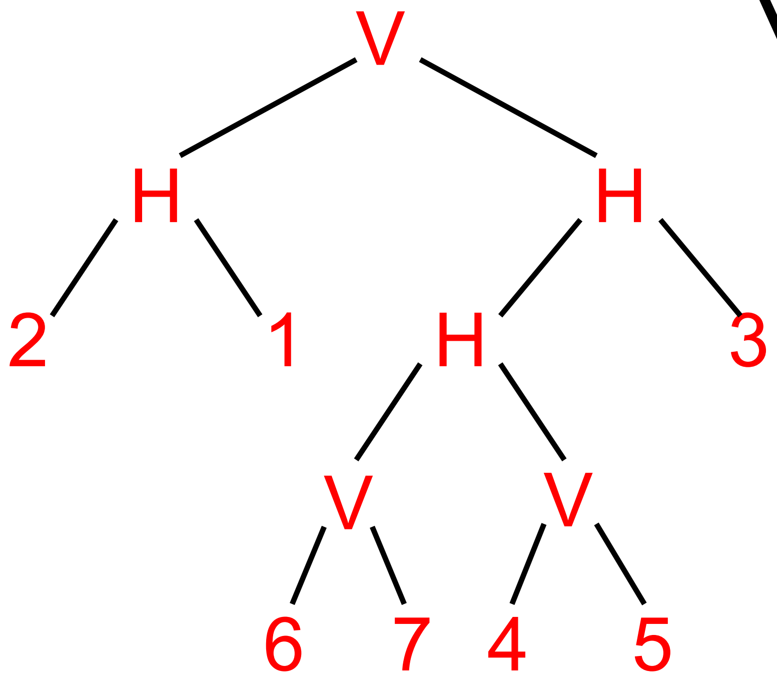
DAC, 1986, pages 101-107.

Representation of Slicing Floorplan

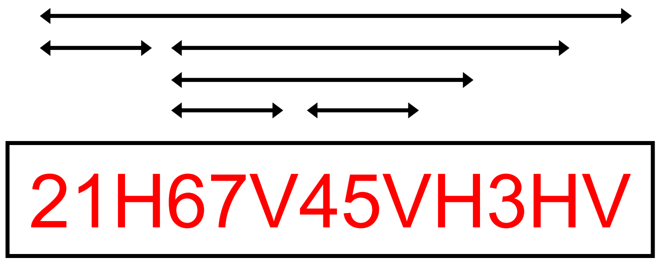
Slicing Floorplan



Slicing Tree

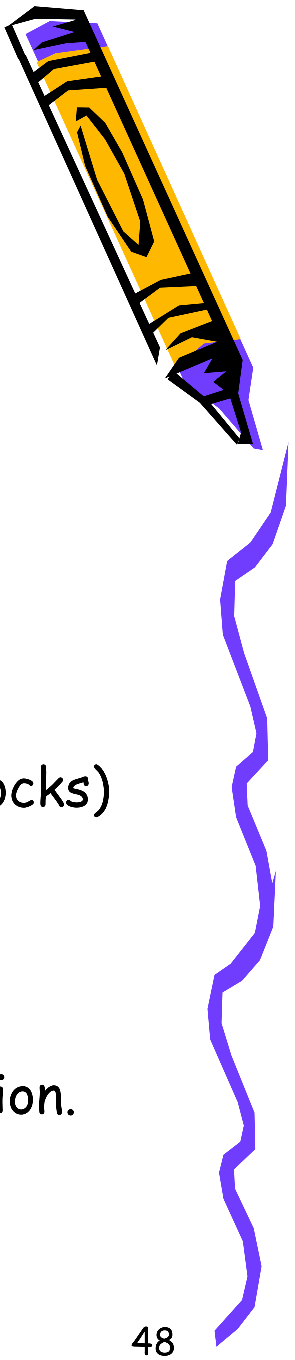


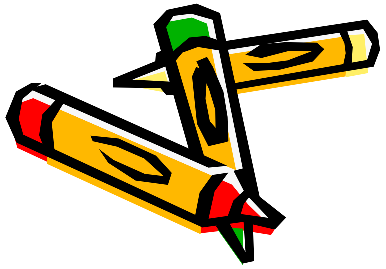
Polish Expression
(postorder traversal of slicing tree)



Polish Expression

- Succinct representation of slicing floorplan
  - roughly specifying relative positions of blocks
- Postorder traversal of slicing tree
  1. Postorder traversal of left sub-tree
  2. Postorder traversal of right sub-tree
  3. The label of the current root
- For *n* blocks, a Polish Expression contains *n* operands (blocks) and *n-1* operators (H, V).
- However, for a given slicing floorplan, the corresponding slicing tree (and hence polish expression) is not unique. Therefore, there is some redundancy in the representation.

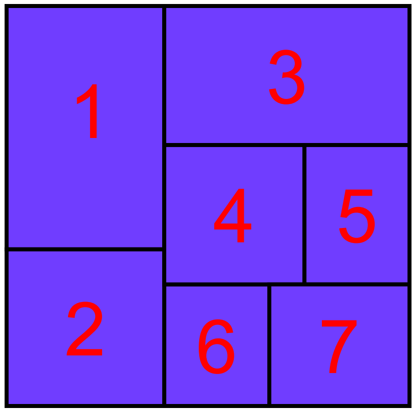




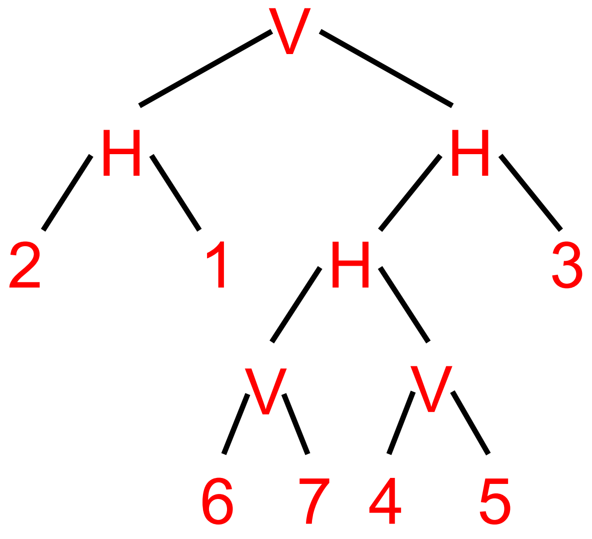
Skewed ST and Normalized PE

- Skewed Slicing Tree:
  - no node and its right son are the same.
- Normalized Polish Expression:
  - no consecutive H's or V's.

Slicing Floorplan



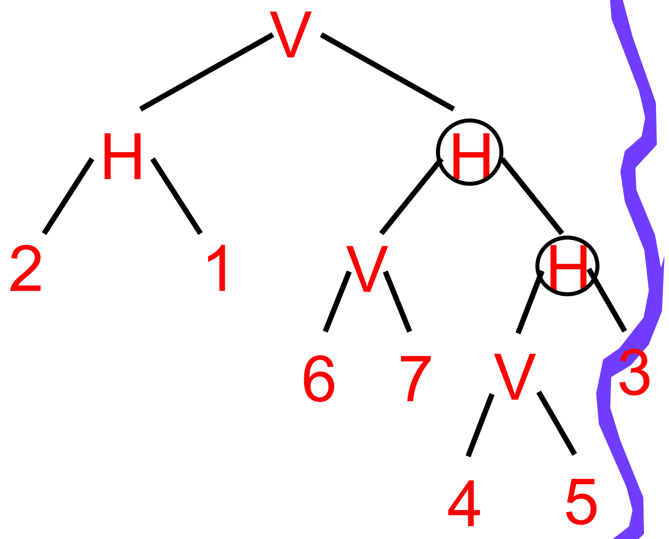
Slicing Tree (Skewed)



$\boxed{21H67V45VH3HV}$

Polish Expression

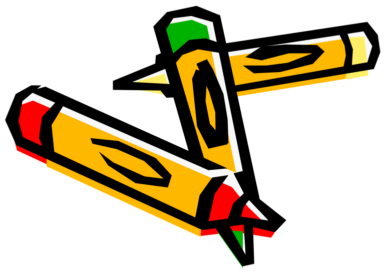
Slicing Tree



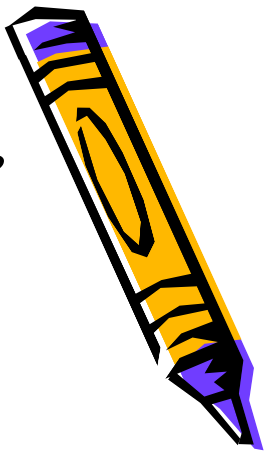
$\boxed{21H67V45V3\underline{HH}V}$

Normalized Polish Expression

- There is a 1-1 correspondence between Slicing Floorplan, Skewed Slicing Tree, and Normalized Polish Expression.
- Will use Normalized Polish Expression to represent slicing floorplans.
  - What is a valid NPE?
- Can be formulated as a state space search problem.



Neighborhood Structure



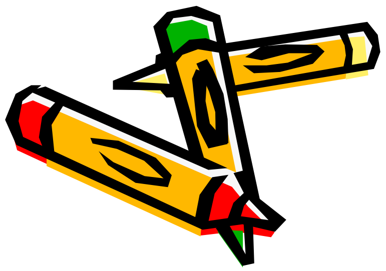
- Chain: HVHVH.... or VHVHV....

$\boxed{\text{16H35V2HV74HV}}$

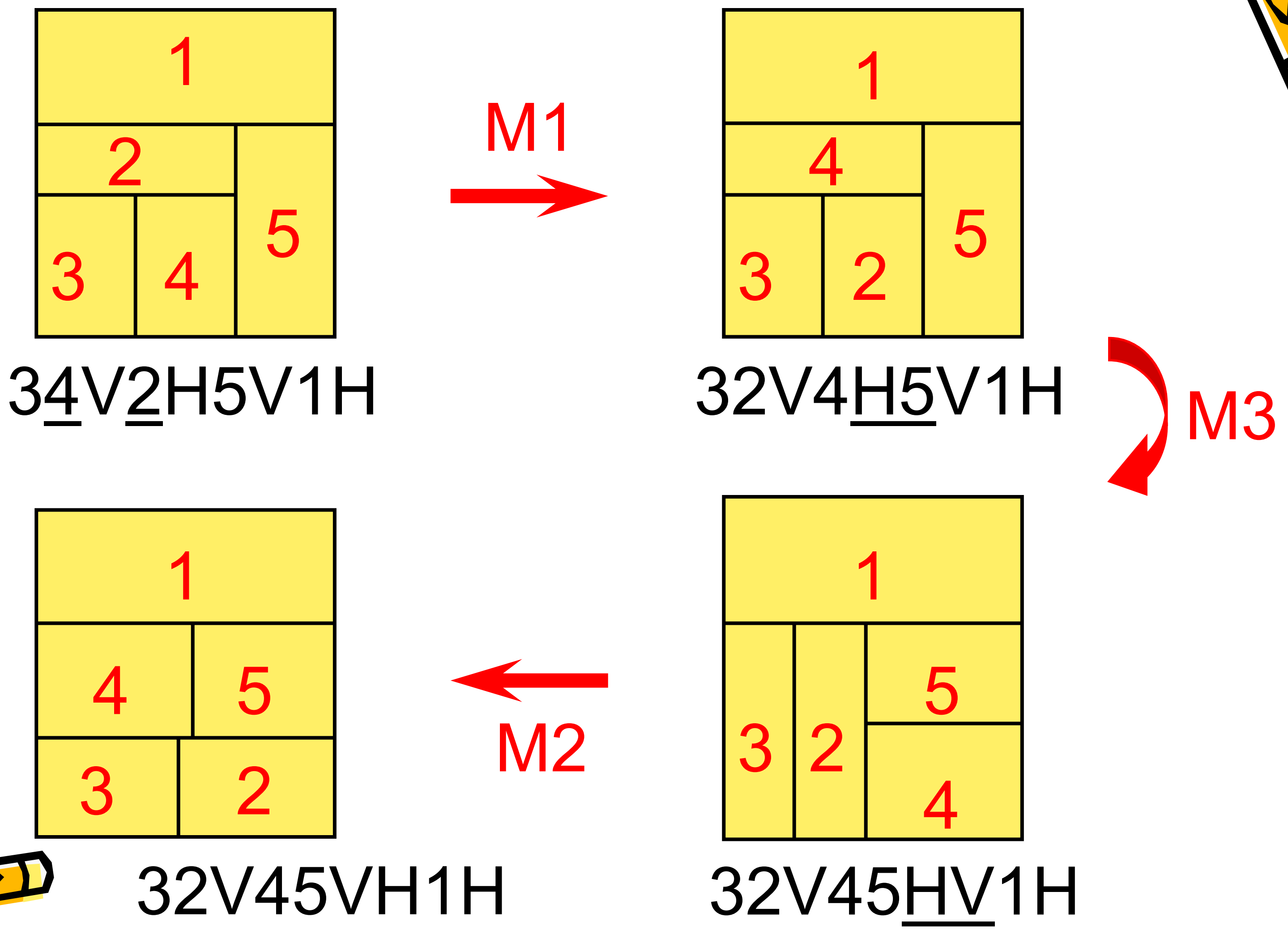
↔ ↔ ⟷ ⟷

Chains

- The moves:
  M1: Swap adjacent operands (ignoring chains)
  M2: Complement some chain
  M3: Swap 2 adjacent operand and operator
  (Note that M3 can give you some invalid NPE.
  So checking for validity after M3 is needed.)

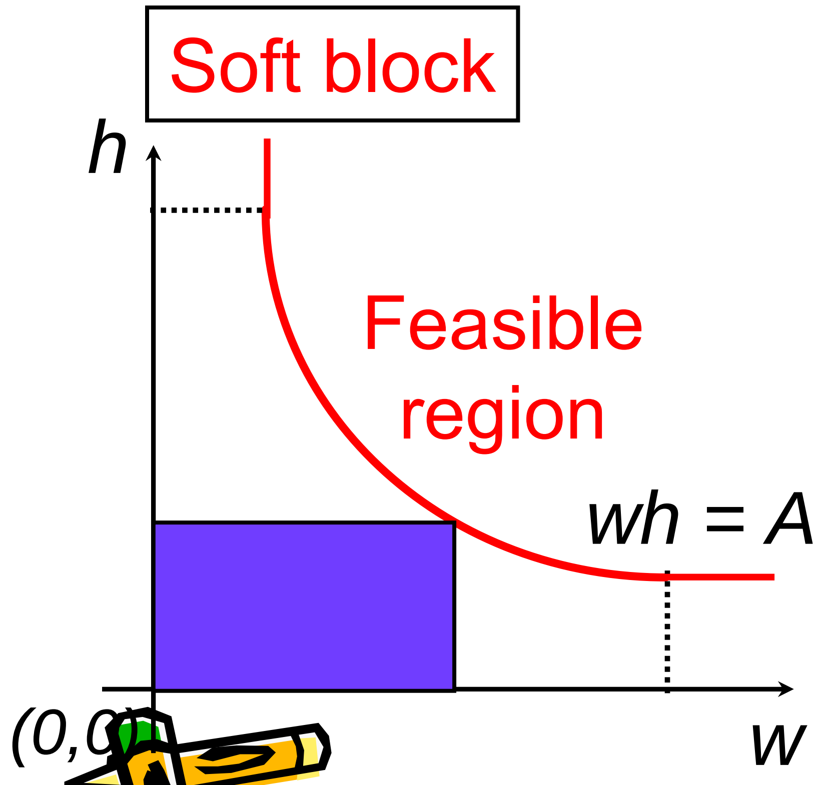


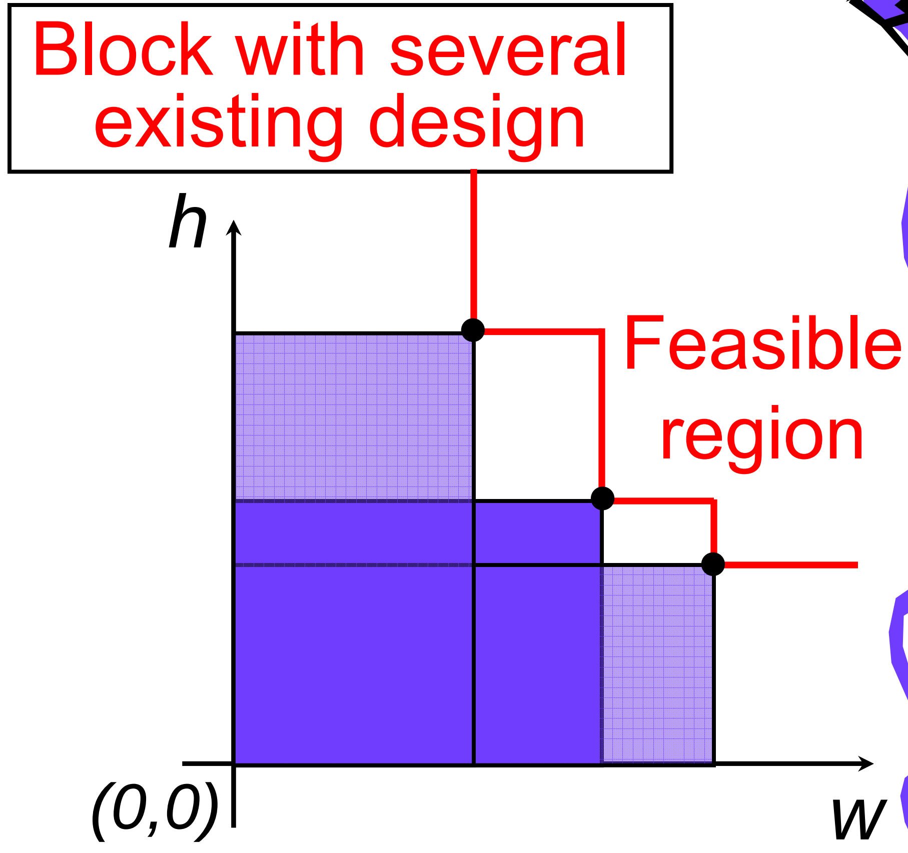
Example of Moves



Shape Curve

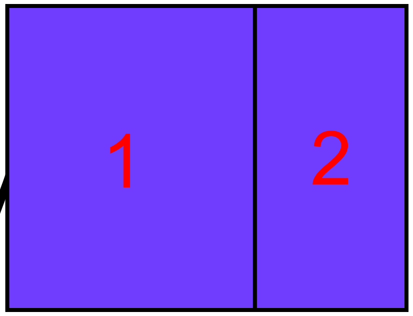
- To represent the possible shapes of a block.

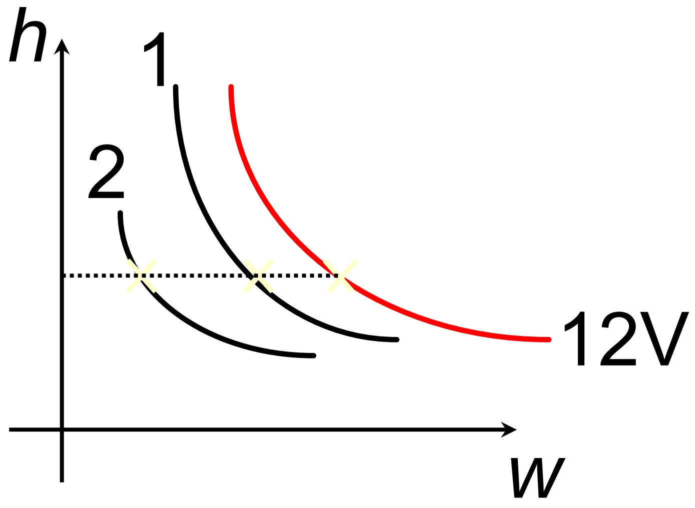




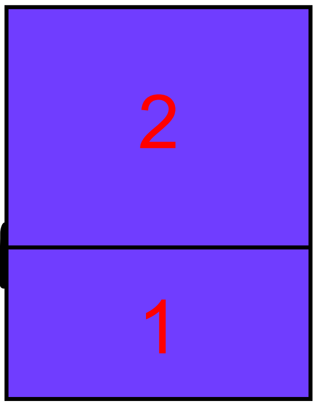
Combining Shape Curves

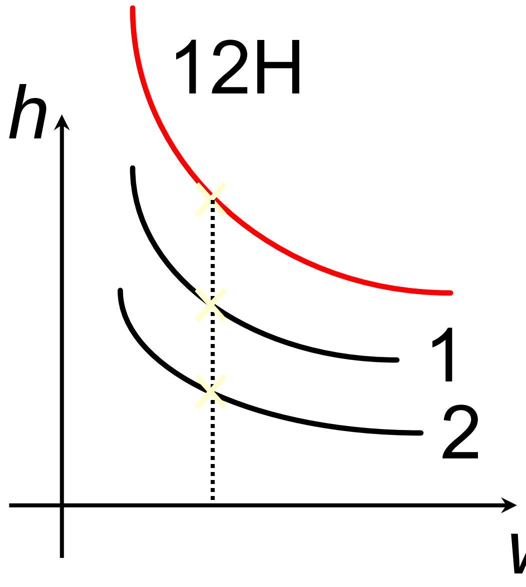
- 12V





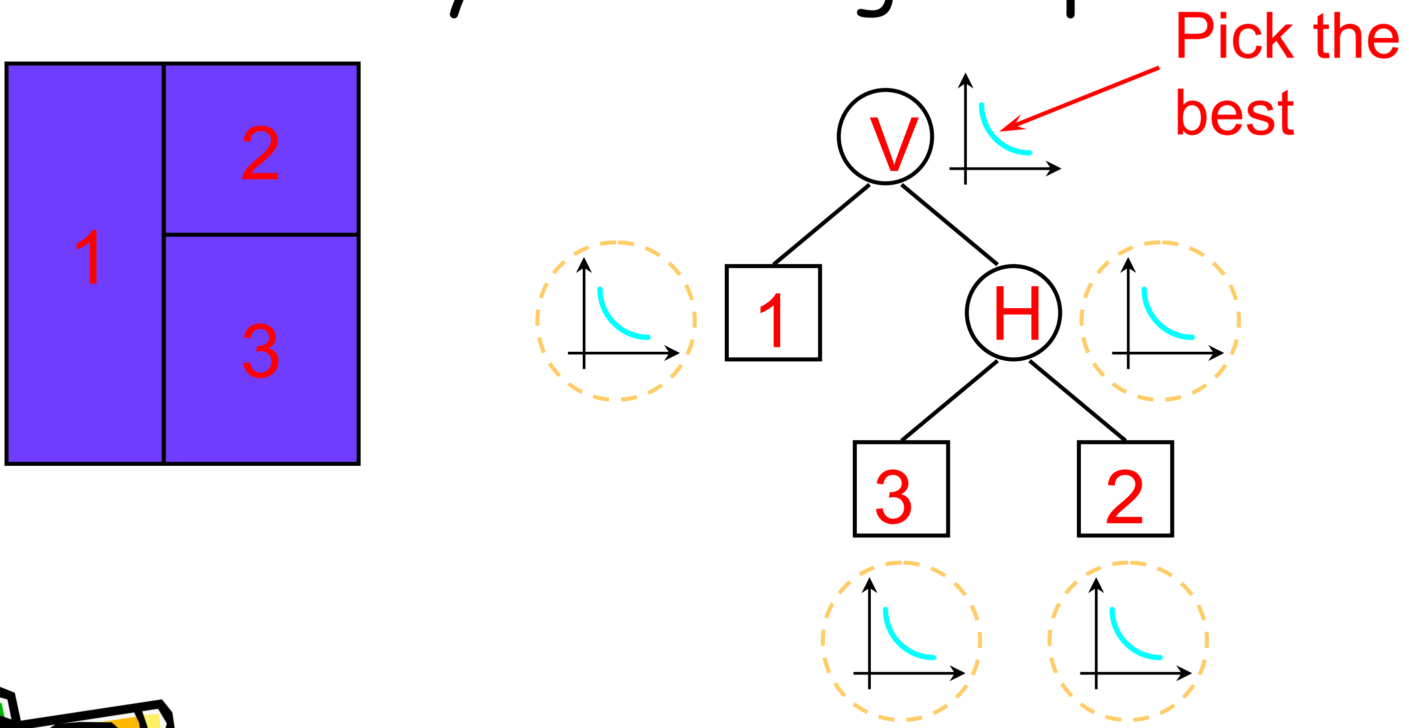
- 12H



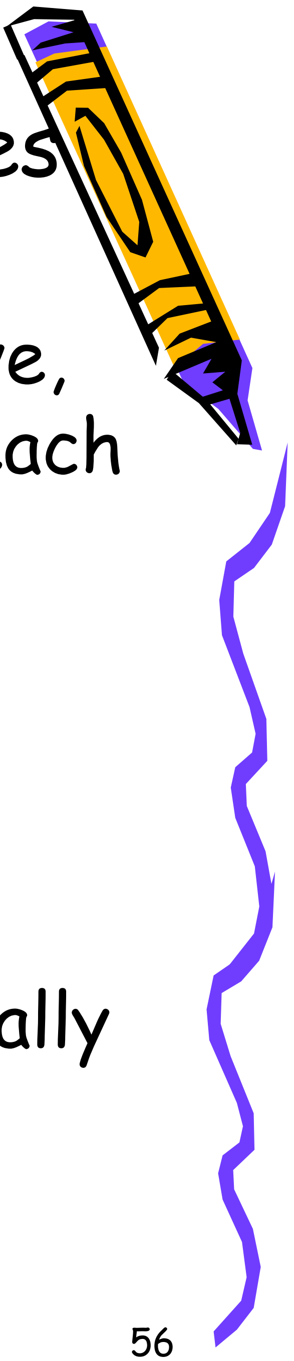


Find the Best Area for a NPE

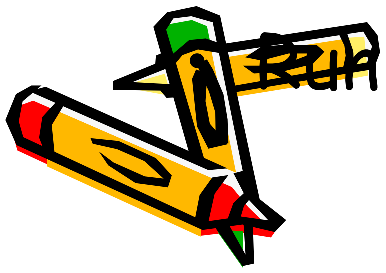
- Recursively combining shape curves.



Updating Shape Curves after Moves

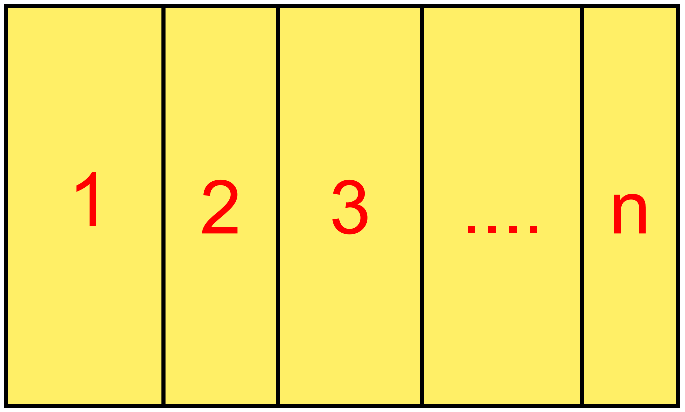


- If keeping k points for each shape curve, time for shape curve computation for each NPE is O(kn).
- After each move, there is only small change in the floorplan. So there is no need to start shape curve computation from scratch.
- We can update shape curves incrementally after each move.
- Run time is about O(k log n).



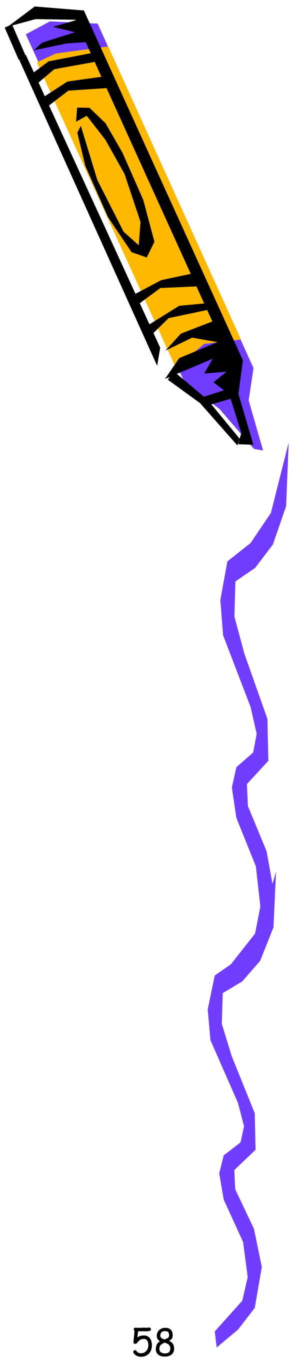
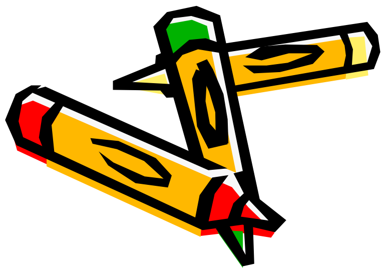
Initial Solution

- 12V3V4V…nV



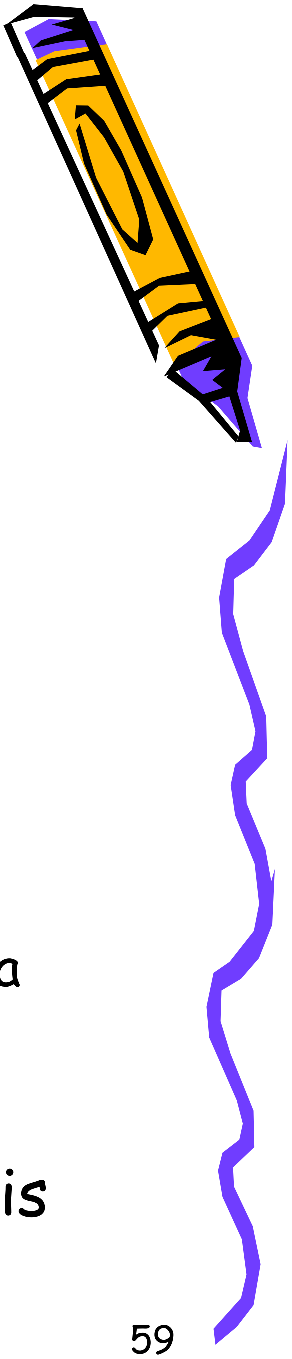
Annealing Schedule

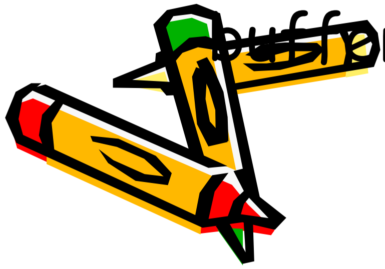
- $T_i = \alpha T_{i-1}$ where $\alpha=0.85$
- At each temperature, try k x n moves (k is around 5 to 10)
- Terminate the annealing process if
  - either # of accepted moves < 5%
  - or the temperate is low enough

Problem formulation

- Input:
  - Blocks (standard cells and macros) $B_1, \ldots, B_n$
  - Shapes and Pin Positions for each block B_i
  - Nets $N_1, \ldots, N_m$
- Output:
  - Coordinates (x_i, y_i) for block B_i.
  - No overlaps between blocks
  - The total wire length is minimized
  - The area of the resulting block is minimized or given a fixed die
- Other consideration: timing, routability, clock, buffering and interaction with physical synthesis

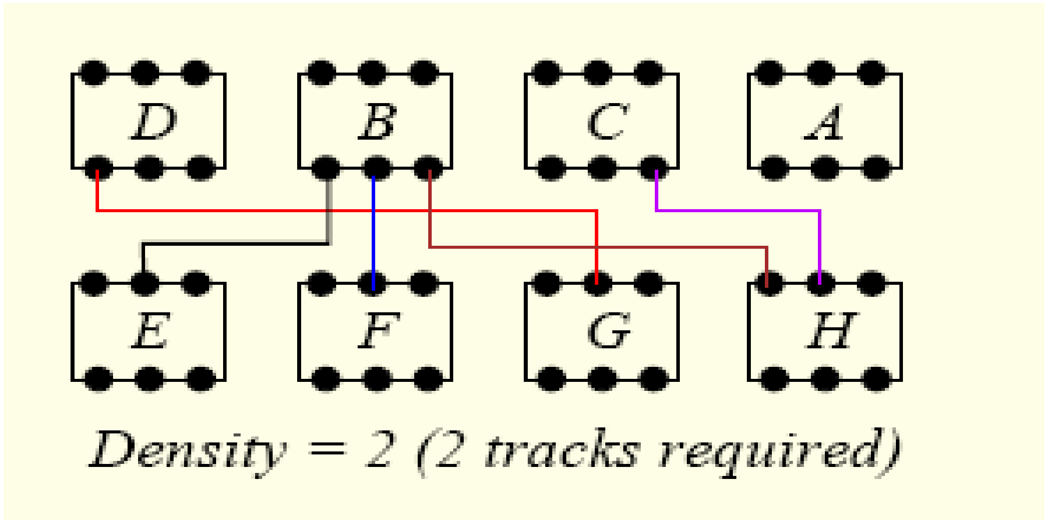




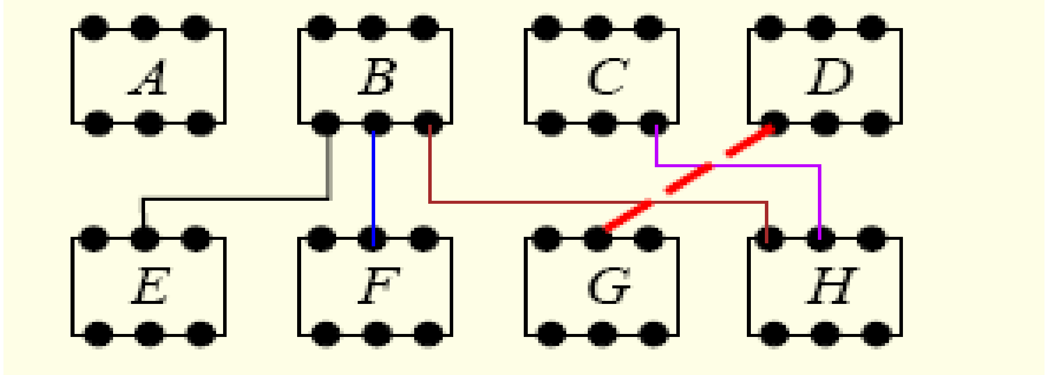
Importance of Placement

- Placement is a key step in physical design
- Poor placement consumes large area, leads to difficult/ impossible routing task
- Ill placed layout cannot be improved by high quality routing
- Quality of placement:
  - Layout area
  - Routability
  - Performance (usually timing, measured by delay of critical/ longest net)

Placement affects chip area

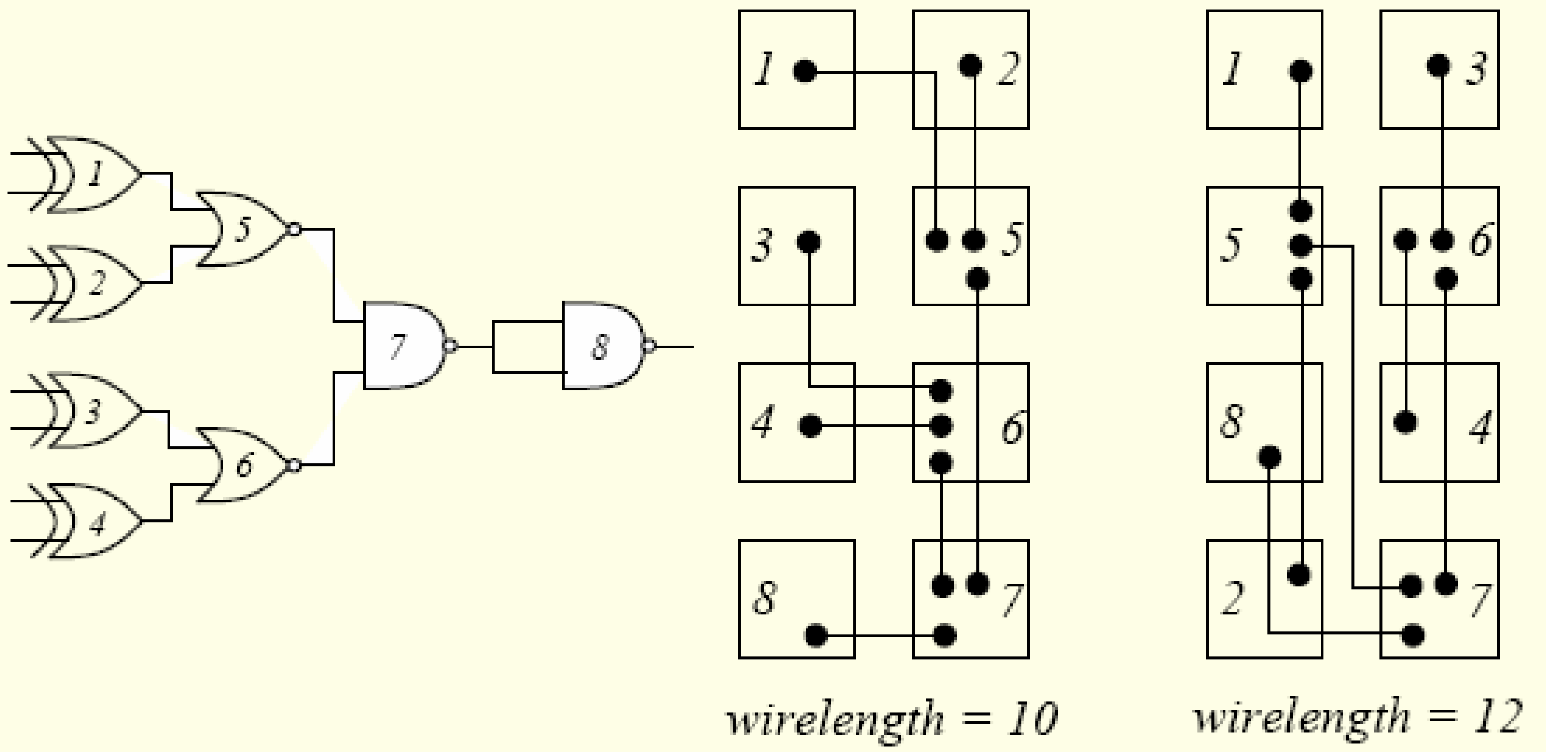


Density = 2 (2 tracks required)

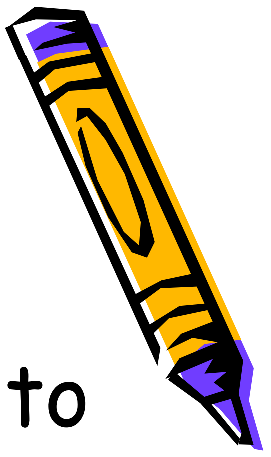


Shorter wirelength, 3 tracks required.

…And also Wire Length

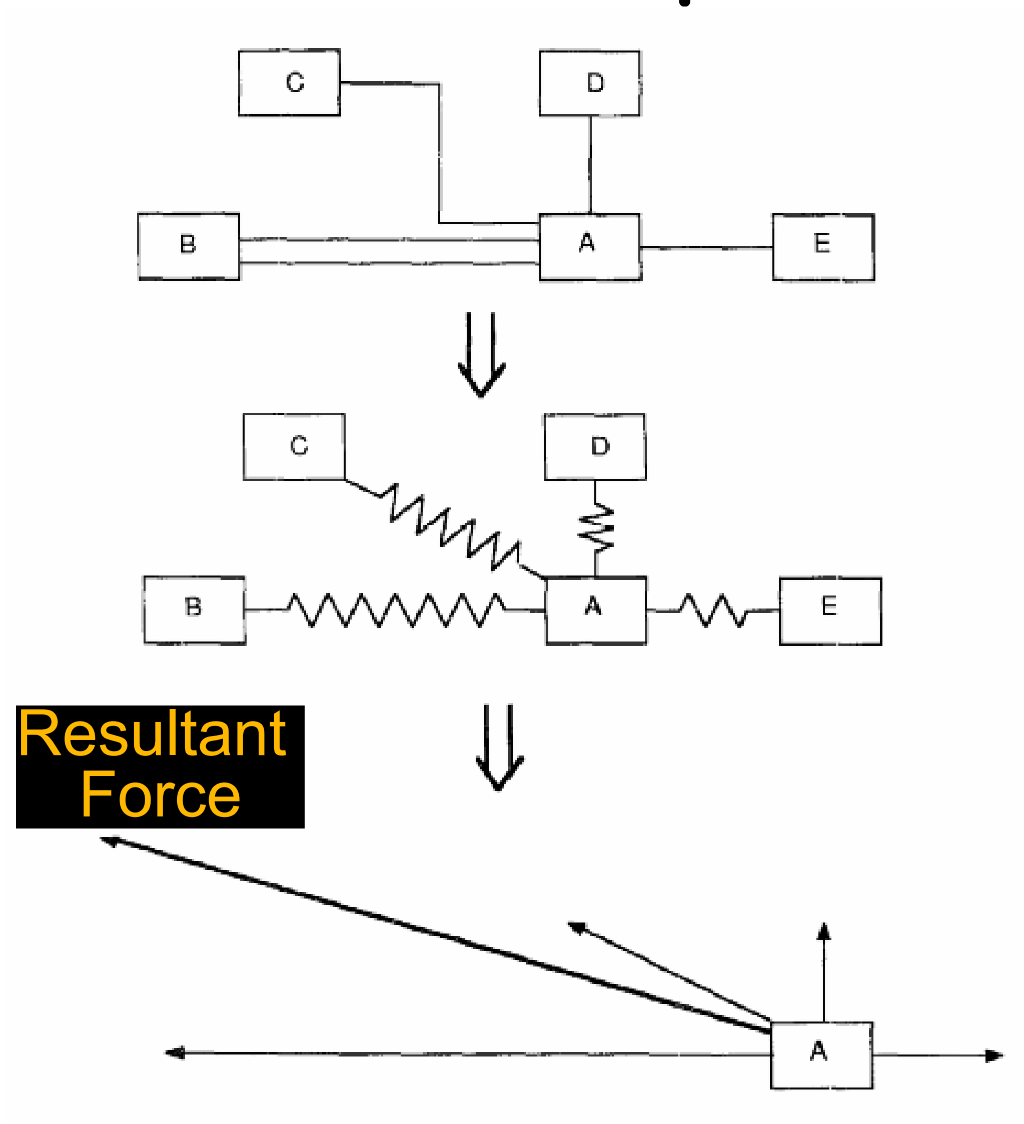


Force Directed Approach



- Transform the placement problem to the classical mechanics problem of a system of objects attached to springs
- Analogies:
  - Module (Block/Cell/Gate) = Object
  - Net = Spring
  - Net weight = Spring constant
  - Optimal placement = Equilibrium configuration

An Example



Force Calculation

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

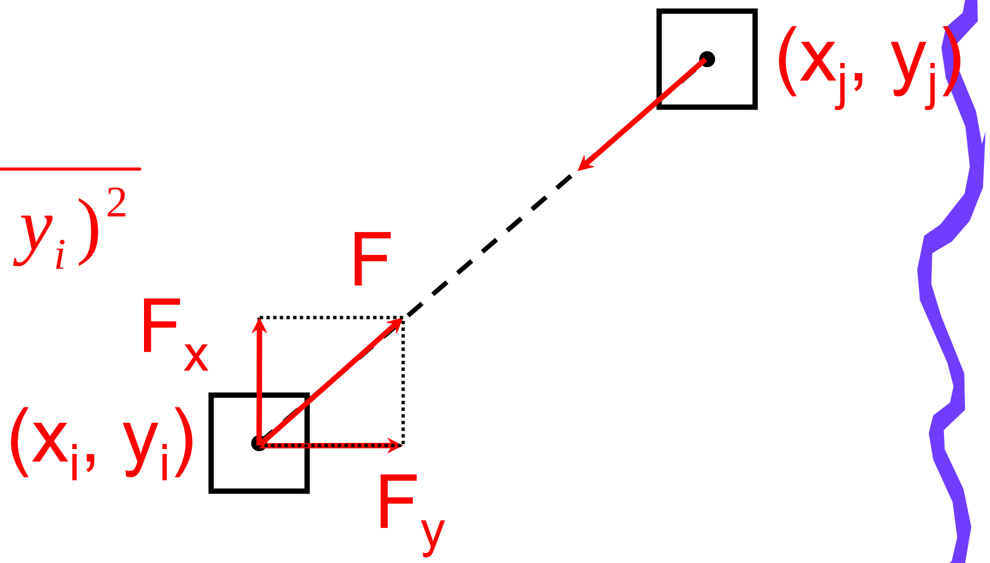
$$\text{Distance } d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

$$\text{Net Cost } c_{ij}$$

$$F = c_{ij}\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

$$F_x = c_{ij}(x_j - x_i)$$

$$F_y = c_{ij}(y_j - y_i)$$



Problem Formulation

- Equilibrium: $\Sigma_j\ c_{ij}\ (x_j - x_i) = 0$ for all module i
- However, trivial solution: $x_j = x_i$ for all i, j. Everything placed on the same position!
- Need to have some way to avoid overlapping
- A method to avoid overlapping:
  - Add some repulsive force which is inversely proportional to distance (or distance squared)
- Solution of force equations correspond to the minimum potential energy of system
  -

$$PE = \sum_{i=1}^{n} [(F_x^i)^2 + (F_y^i)^2]$$

Comments on Force-Directed Placement

✓ Use directions of forces to guide the search

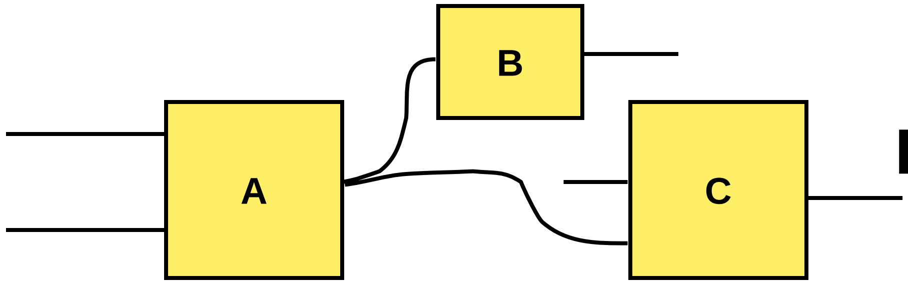
✓ Usually much faster than simulated annealing

x Focus on connections, not shapes of blocks

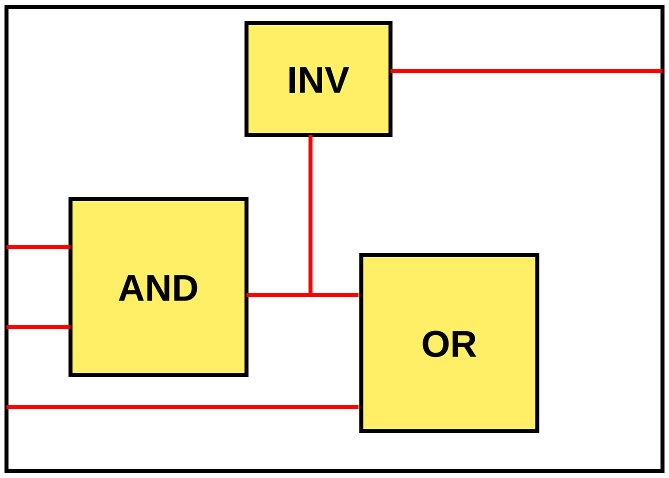
x Only a heuristic; an equilibrium configuration does not necessarily give a good placement

? Successful or not depends on the way to eliminate overlapping

Routing in design flow



Post Placed Netlist



Floorplan/Placement

Routing

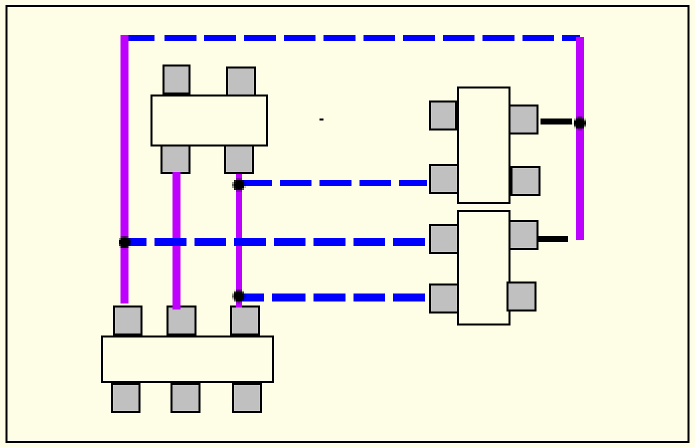
Process of finding geometric layouts of the net

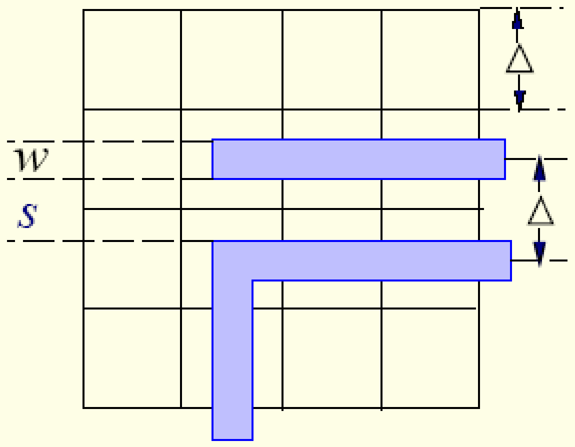
The Routing Problem

- Apply it after Placement
- Input:
  - Netlist
  - Timing budget for, typically, critical nets
  - Locations of blocks and locations of pins
- Output:
  - Geometric layouts of all nets
- Objective:
  - Minimize the total wire length, the number of vias, or just completing all connections without increasing the chip area.
  - Each net meets its timing budget.

The Routing Constraints

- Examples:
  - Placement constraint
  - Number of routing layers
  - Delay constraint
  - Meet all geometrical constraints (design rules)
  - Physical/Electrical/Manufacturing constraints:
    - Crosstalk


Two−layer routing



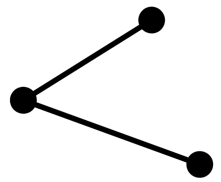
Geometrical constraint

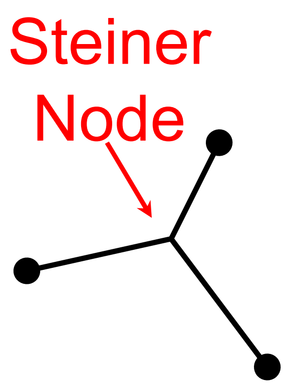
Steiner Tree

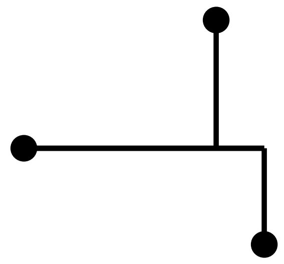
- For a multi-terminal net, we can construct a spanning tree to connect all the terminals together.
- But the wire length will be large.
- Better use Steiner Tree:

  A tree connecting all terminals and some additional nodes (Steiner nodes).
- Rectilinear Steiner Tree:

  Steiner tree in which all the edges run horizontally and vertically.

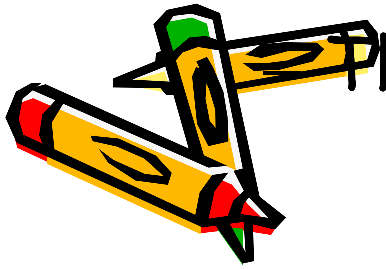


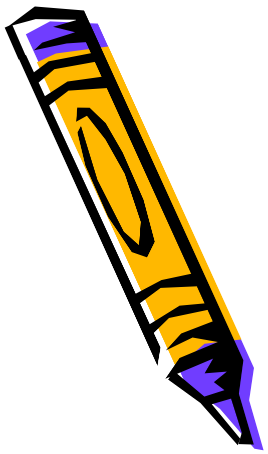




Routing Problem is Very Hard

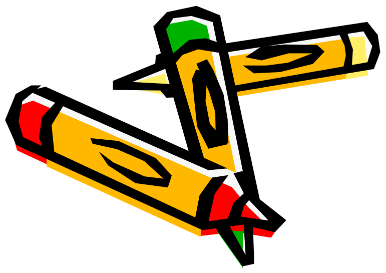
- Minimum Steiner Tree Problem:
  - Given a net, find the Steiner tree with the minimum length.
  - Input :An edge weighted graph G=(V,E) and a subset D (demand points)
  - Output: A subset of vertices V'(such that D is covered) and induces a tree of minimum cost over all such trees
  - This problem is NP-Complete!





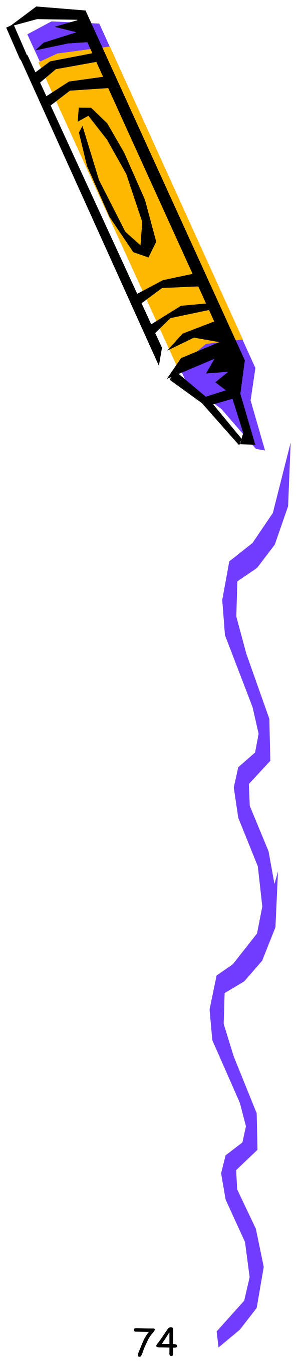
Heuristic Algorithms

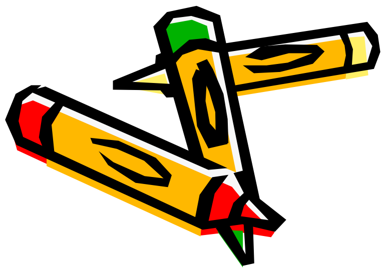
- Use MST (minimum spanning tree) algorithms to start with
  - $Cost_{MST}/Cost_{RMST} \leq 3/2$
  - Heuristics can guarantee that the weight of RST is at most 3/2 of the weight of the optimal tree
- Apply local modifications to reach a RMST (rectilinear minimum steiner tree)



Kinds of Routing

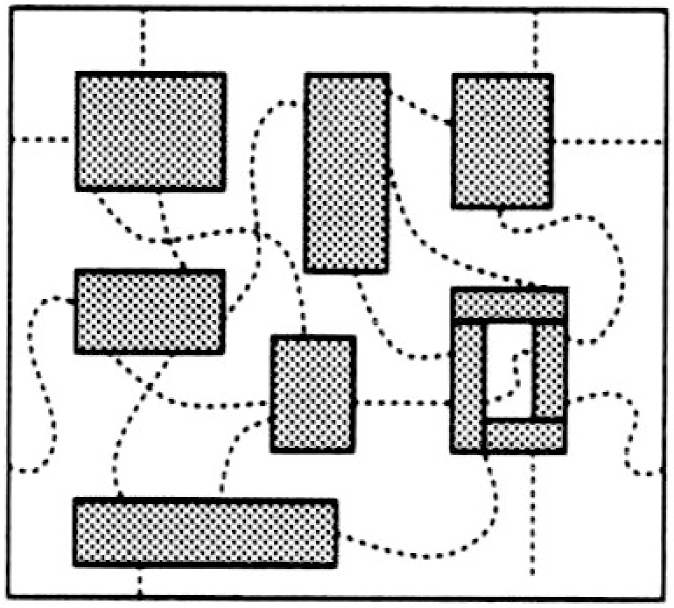
- Global Routing
- Detailed Routing
  - Channel
  - Switchbox
- Others:
  - Maze routing
  - Over the cell routing
  - Clock routing



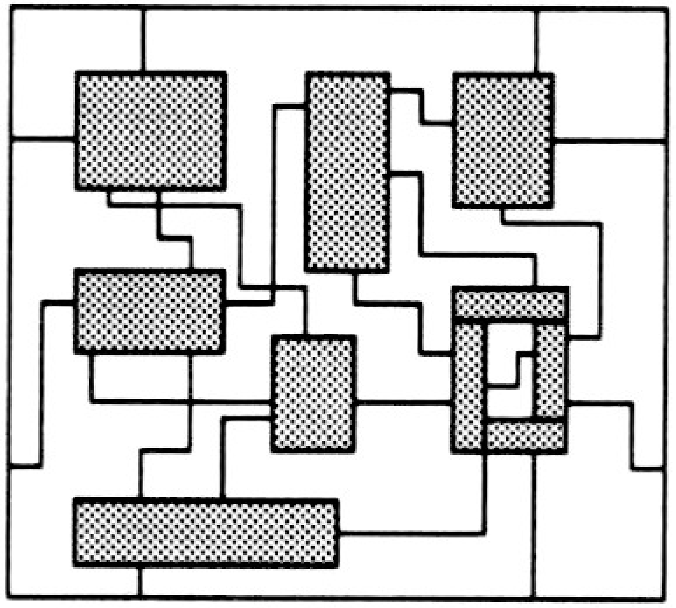


General Routing Paradigm

Two phases:



Global Routing

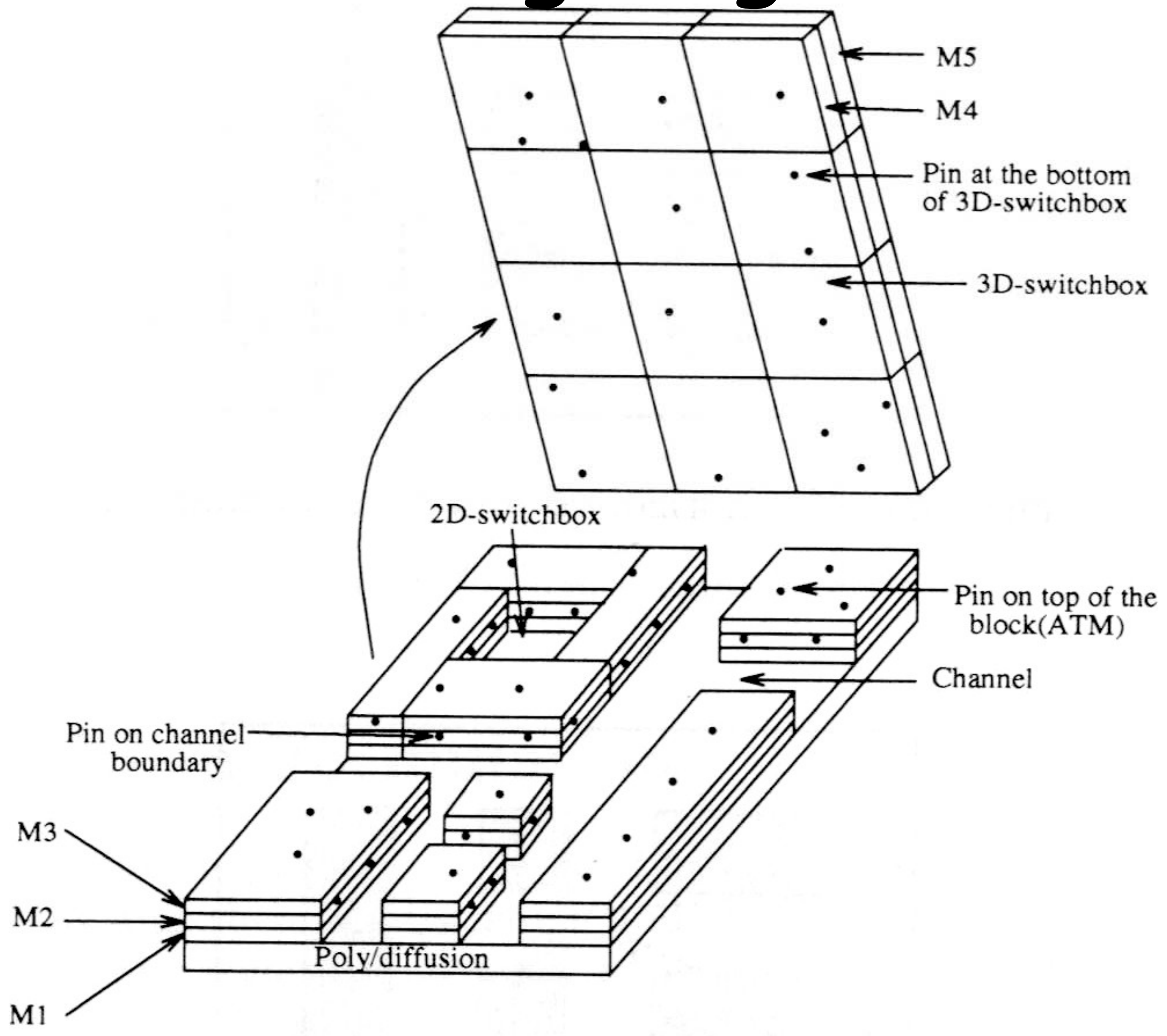


Detailed Routing

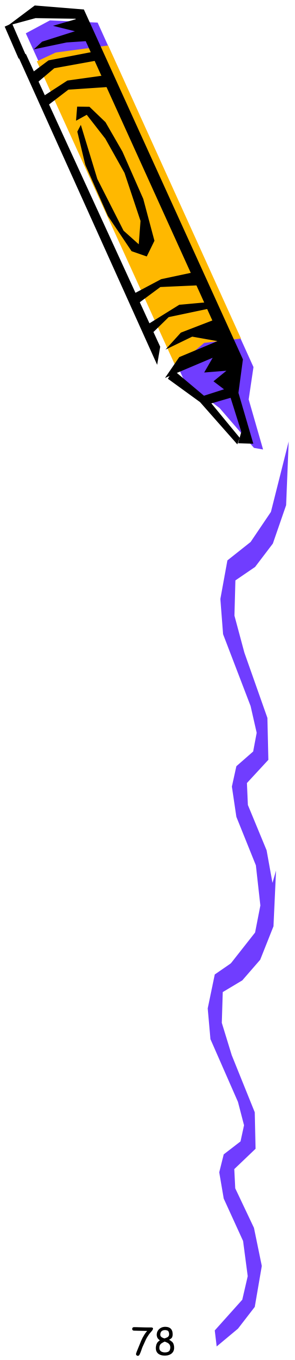
Extraction and Timing Analysis

- After global routing and detailed routing, information of the nets can be extracted and delays can be analyzed.
- If some nets fail to meet their timing budget, detailed routing and/or global routing needs to be repeated.

Routing Regions



Global Routing

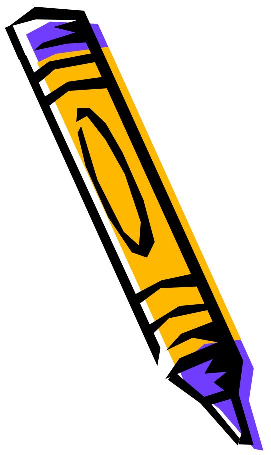


Global routing is divided into 3 phases:

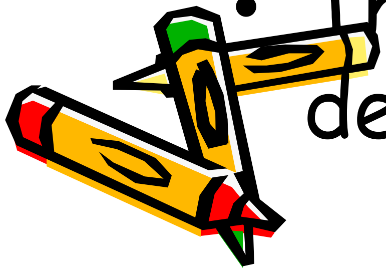
1. Region definition
2. Region assignment
3. Pin assignment to routing regions

Maze Routing

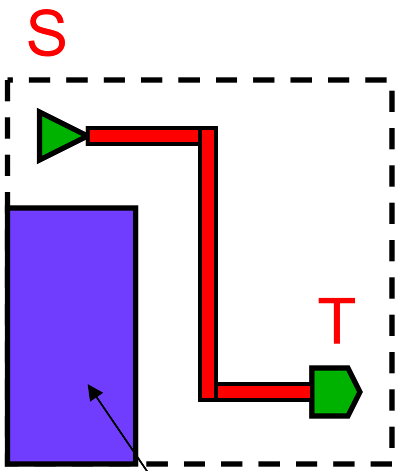
Maze Routing Problem



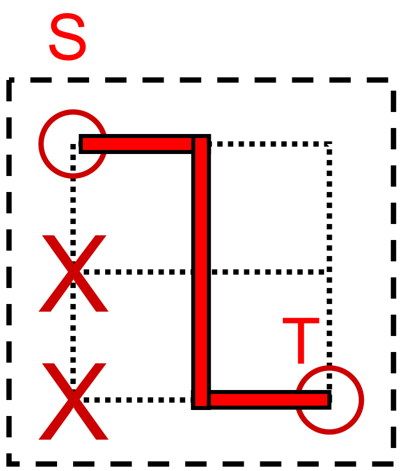
- Given:
  - A planar rectangular grid graph.
  - Two points S and T on the graph.
  - Obstacles modeled as blocked vertices.
- Objective:
  - Find the shortest path connecting S and T.
- This technique can be used in global or detailed routing (switchbox) problems.



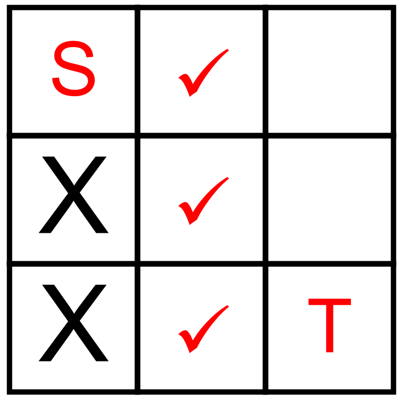
Grid Graph



Area Routing



Grid Graph (Maze)

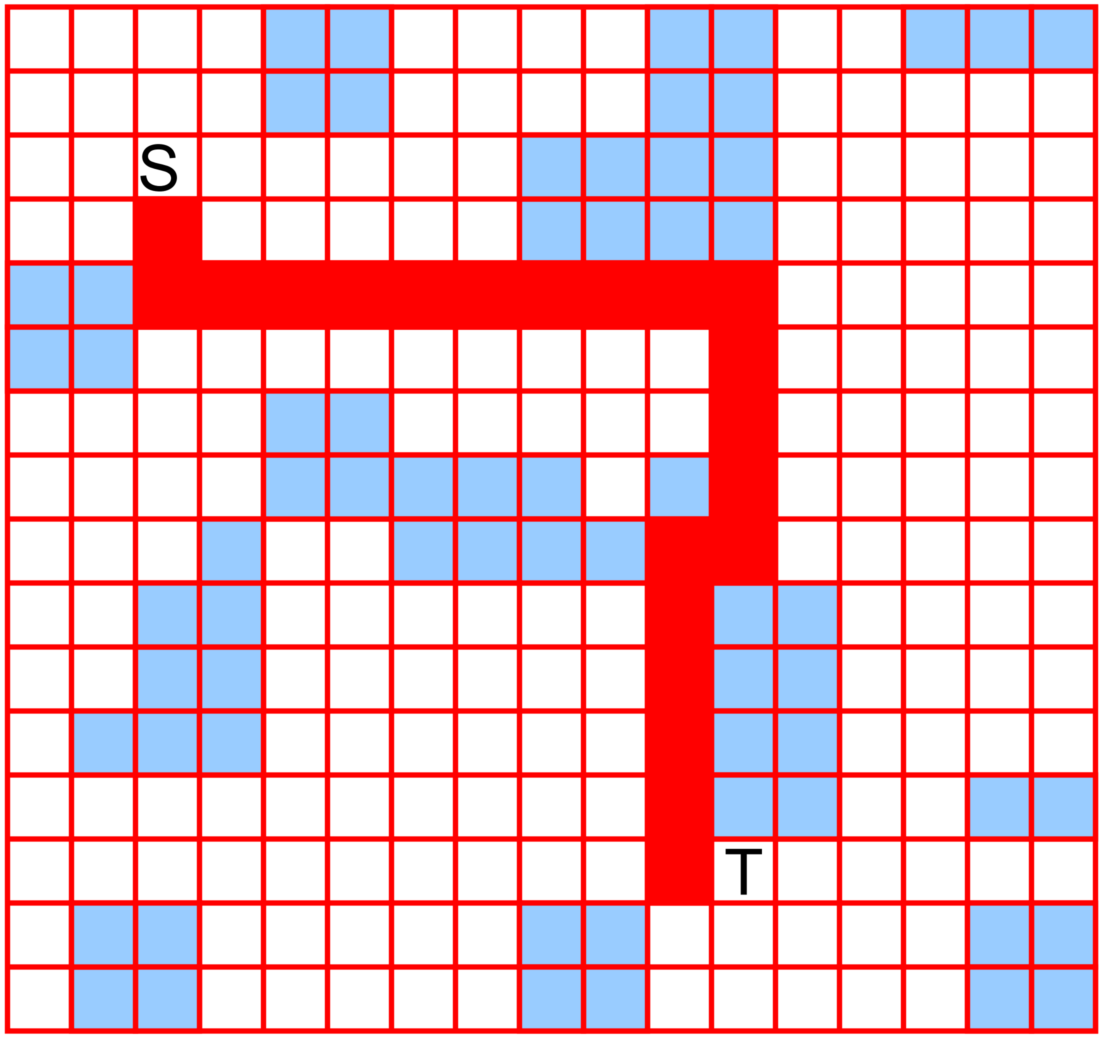


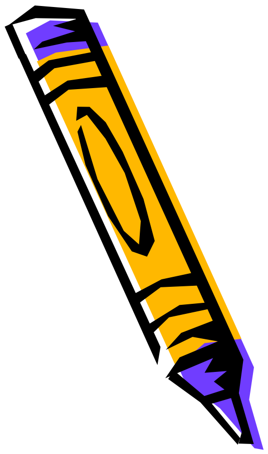
S	✓	
X	✓	
X	✓	T

Simplified Representation

Blocked cells

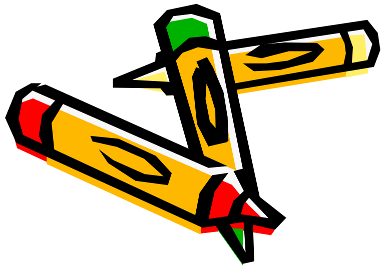
Maze Routing





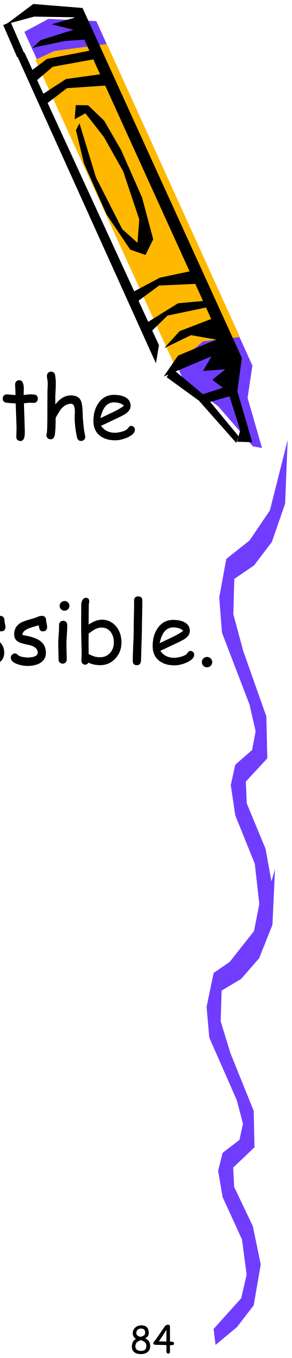
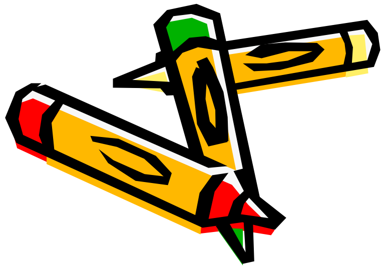
Lee’s Algorithm

“An Algorithm for Path Connection and its Application”, C.Y. Lee, IRE Transactions on Electronic Computers, 1961.



Basic Idea

- A Breadth-First Search (BFS) of the grid graph.
- Always find the shortest path possible.
- Consists of two phases:
  - Wave Propagation
  - Retrace

An Illustration

S 0	1	2	3
1	2	3	▨
▨	3	4	5
5	4	5	T 6

Wave Propagation

- At step k, all vertices at Manhattan-distance k from S are labeled with k.
- A Propagation List (FIFO) is used to keep track of the vertices to be considered next.

S 0			
			🟦
🟦			
			T

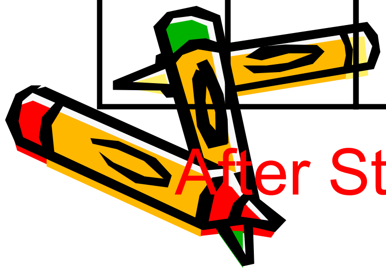
After Step 0

S 0	1	2	3
1	2	3	🟦
🟦	3		
			T

After Step 3

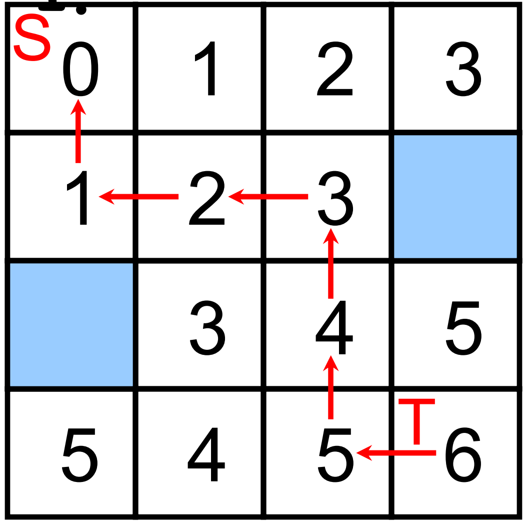
S 0	1	2	3
1	2	3	🟦
🟦	3	4	5
5	4	5	T 6

After Step 6



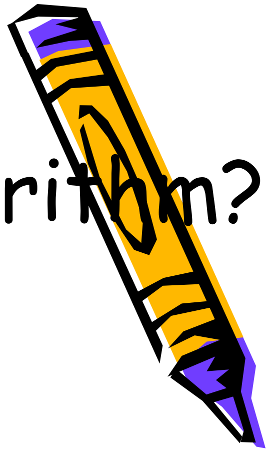
Retrace

- Trace back the actual route.
- Starting from *T*.
- At *vertex with k, go to any vertex with label k-1.*

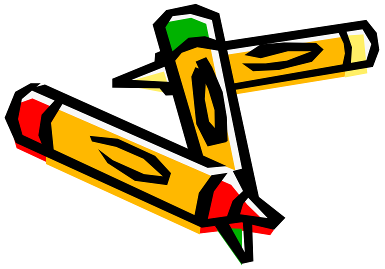


Final labeling

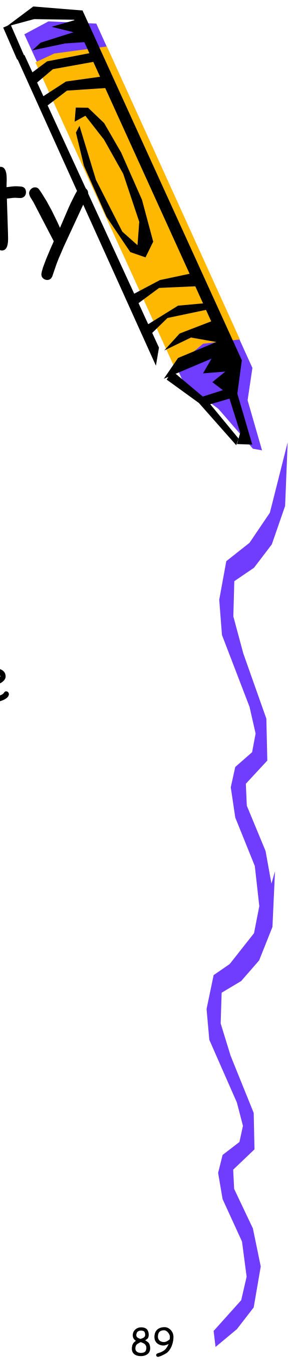
How many grids visited using Lee's algorithm?



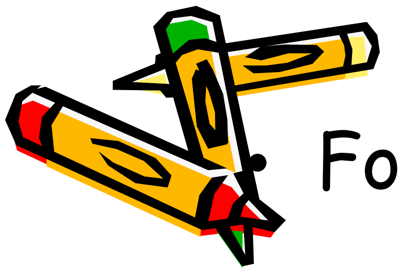
13	12	11	10	🟦	🟦	7	6	7	7	🟦	🟦	9	10	🟦	🟦	🟦
12	11	10	9	🟦	🟦	6	5	6	7	🟦	🟦	8	9	10	11	12
11	10	9	8	7	6	5	4	🟦	🟦	🟦	🟦	7	8	9	10	11
10	9	8	7	6	5	4	3	🟦	🟦	🟦	🟦	6	7	8	9	10
🟦	🟦	7	6	5	4	3	2	1	2	3	4	5	6	7	8	9
🟦	🟦	6	5	4	3	2	1	S	1	2	3	4	5	6	7	8
9	8	7	6	🟦	🟦	3	2	1	2	3	4	5	6	7	8	9
10	9	8	7	🟦	🟦	🟦	🟦	🟦	3	🟦	5	6	7	8	9	10
11	10	9	8	9	10	🟦	🟦	🟦	🟦	7	6	7	8	9	10	11
12	11	🟦	🟦	10	11	12	11	10	9	8	🟦	🟦	9	10	11	12
13	12	🟦	🟦	11	12	13	12	11	10	9	🟦	🟦	10	11	12	13
	🟦	🟦	🟦	12	13		13	12	11	10	🟦	🟦	11	12	13	
				13				13	12	11	🟦	🟦	12	13	🟦	🟦
									13	12	T		13			
	🟦	🟦						🟦	🟦	13					🟦	🟦
	🟦	🟦						🟦	🟦						🟦	🟦



Time and Space Complexity

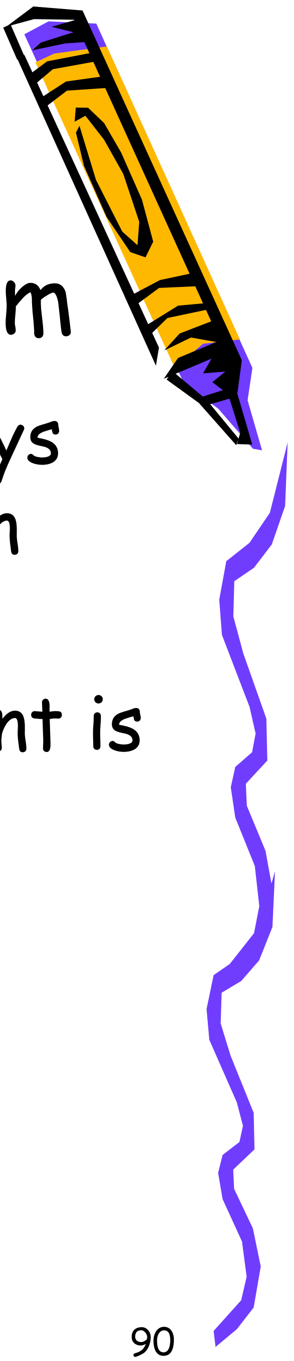


- For a grid structure of size $w \times h$:
  - Time per net = $O(wh)$
  - Space = $O(wh \log wh)$ ($O(\log wh)$ bits are needed during exploration phase + one additional bit to indicate blocked or not)
- For a 2000×2000 grid structure:
  - 12 bits per label
  - Total 6 Mbytes of memory!

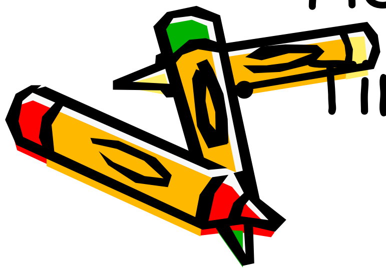


- For 4000 x 4000, 48 M bytes!

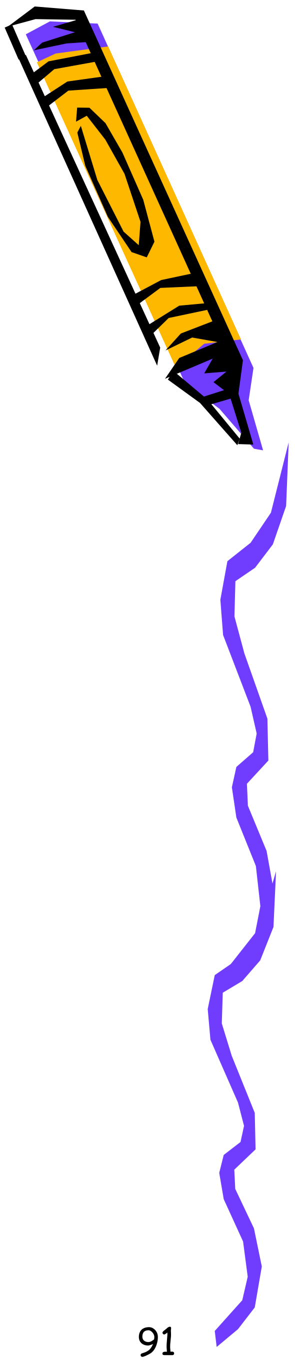
Acker's coding : Improvement to Lee's Algorithm

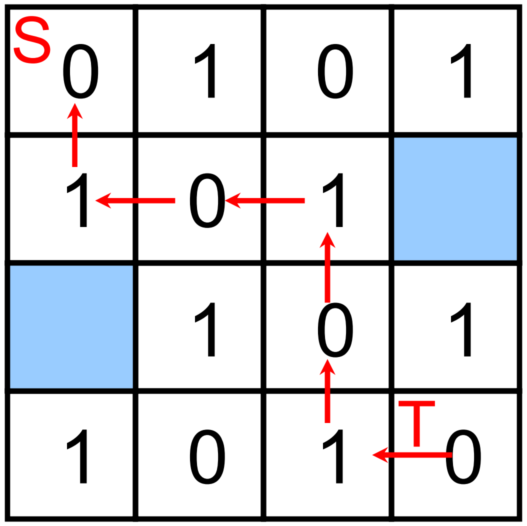


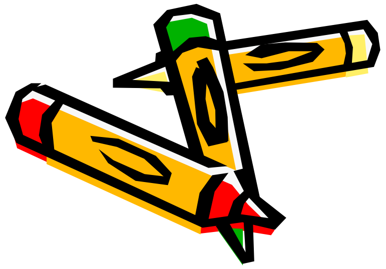
- The vertices in wave-front L are always adjacent to the vertices L-1 and L+1 in the wavefront
- Soln: the predecessor of any wavefront is labeled different from its successor
- 0,0,1,1,0,….
- Need to indicate blocked or not
- Hence can do away with 2 bits
- Time complexity is not improved



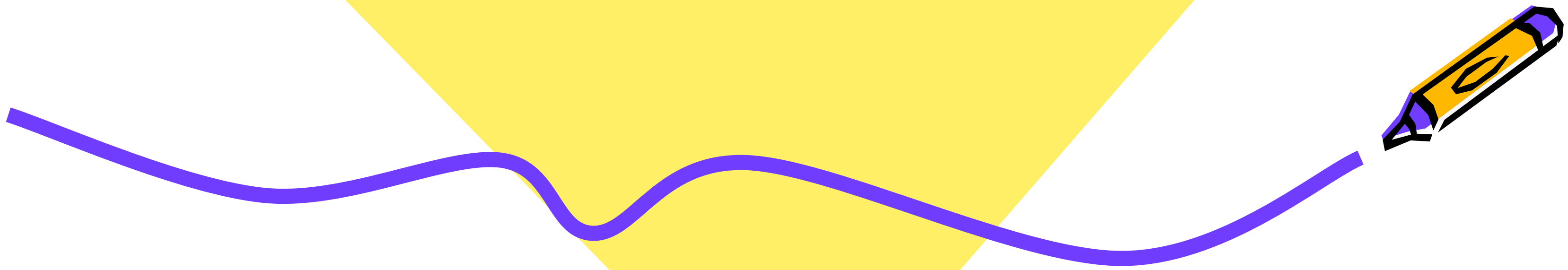
Acker's Technique





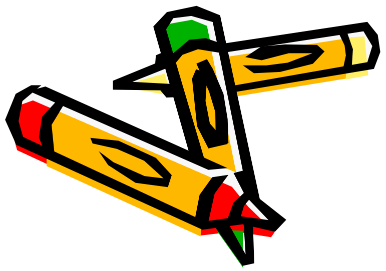


Detailed Routing

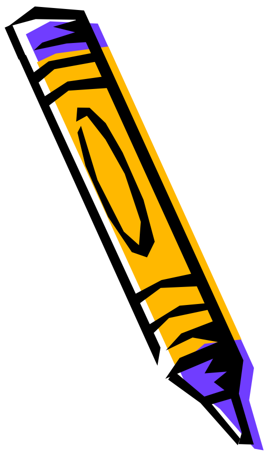


Detailed routing

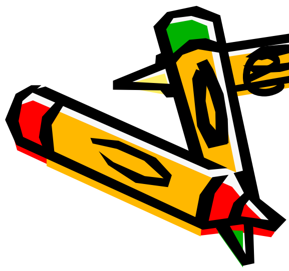
- Global routing do not define wires
- They define routing regions
- Detailed router places actual wires within regions, indicated by the global router
- We consider the channel routing problem here…



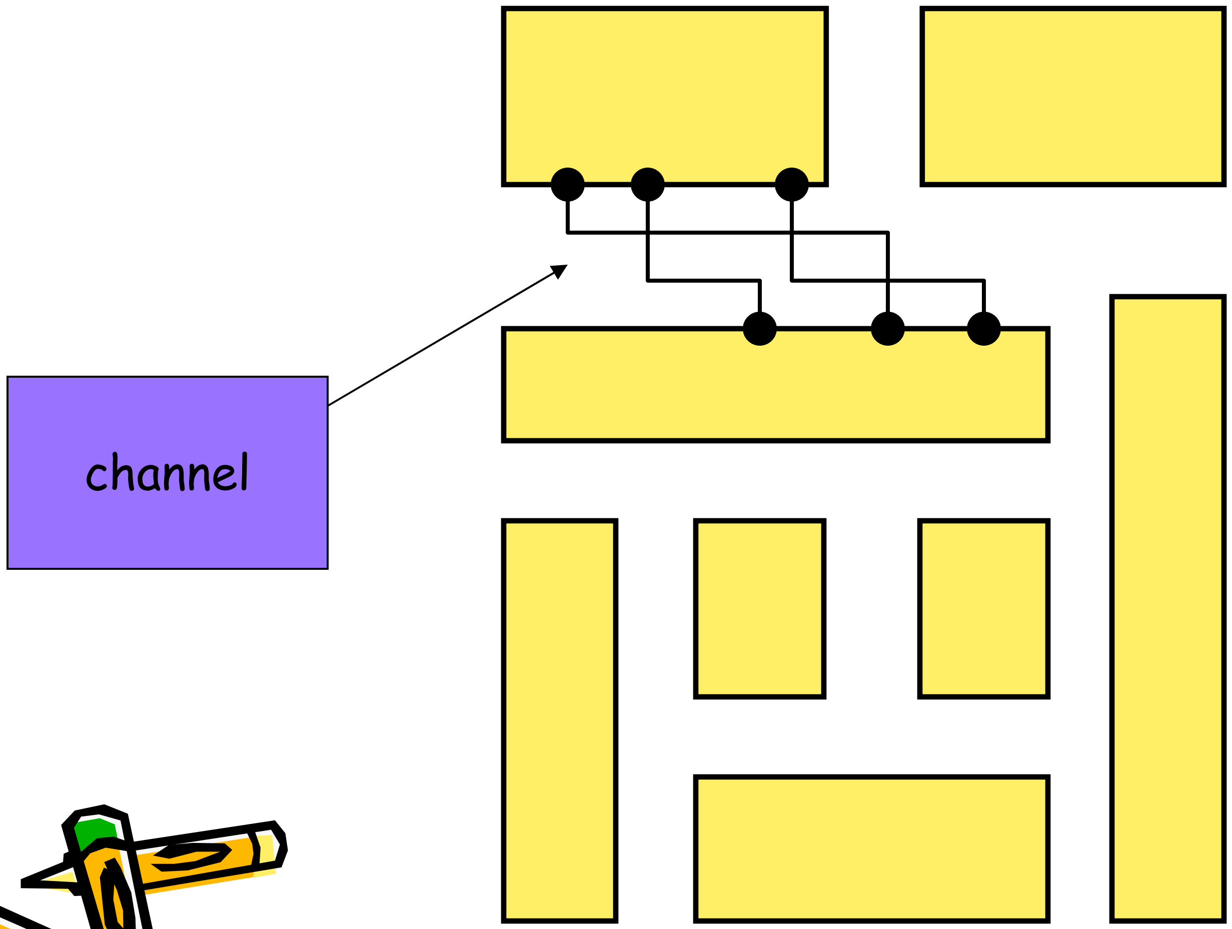
Channel Routing



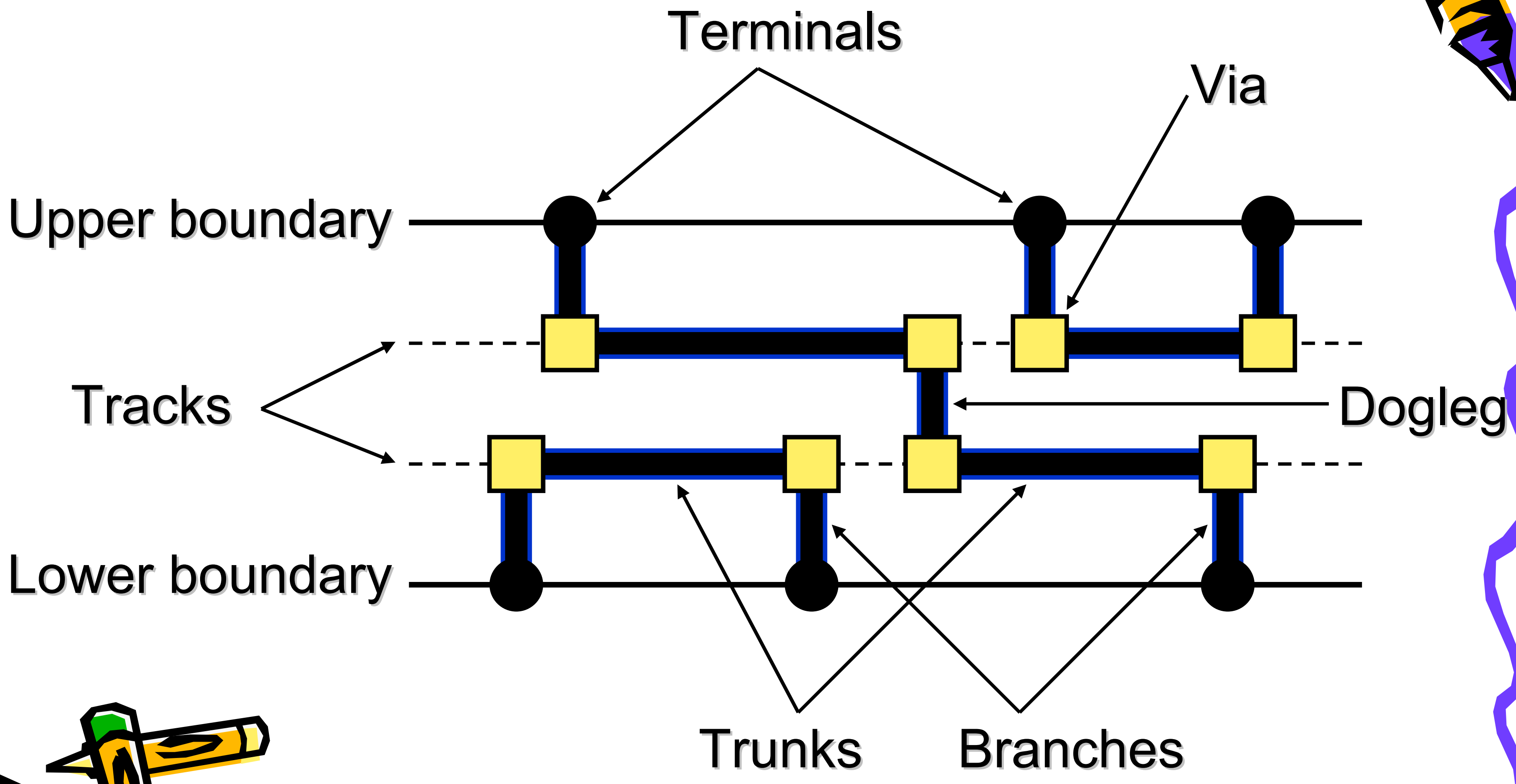
- A channel is the routing region bounded by two parallel rows of terminals
- Assume top and bottom boundary
- Each terminal is assigned a number to indicate which net it belongs to
- 0 indicates : does not require an electrical connection



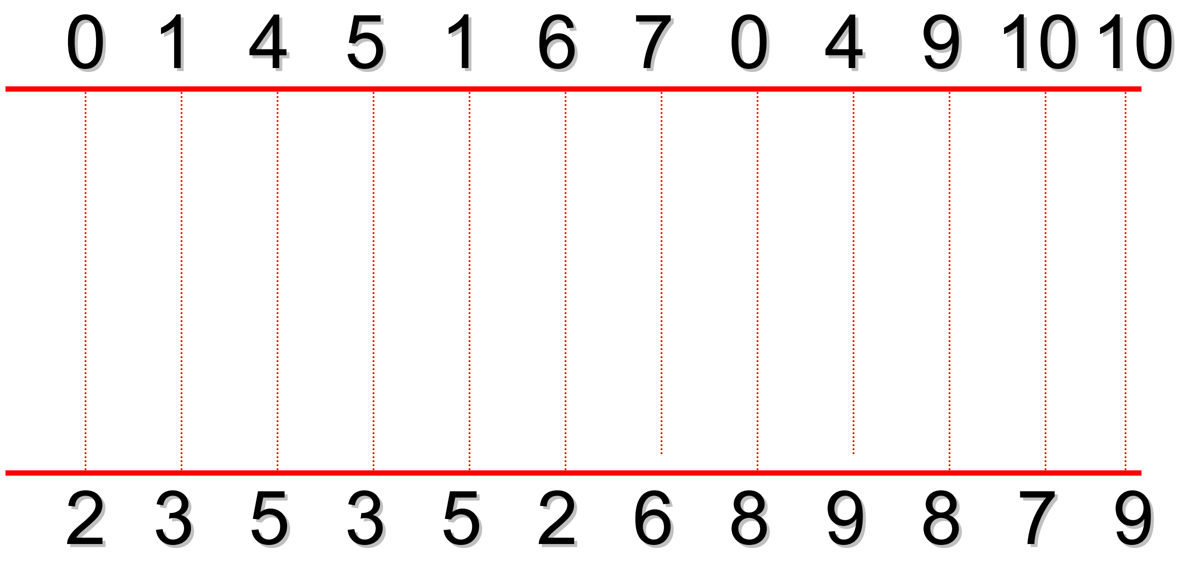
Channel Routing



Channel Routing

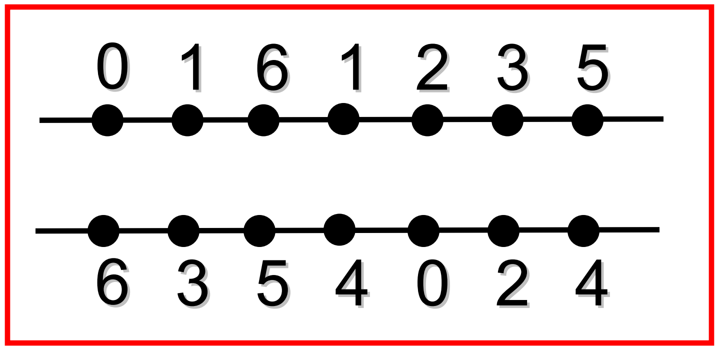


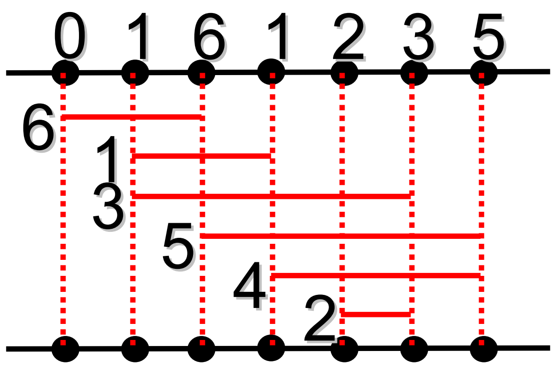
Channel Routing

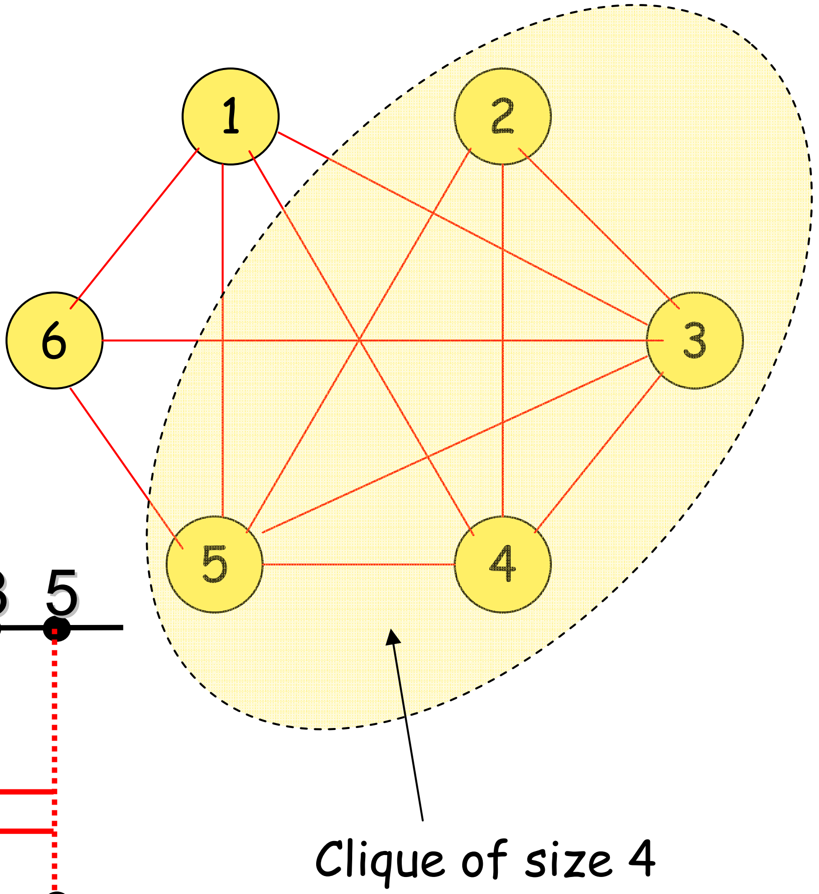


How to connect all the points with the same label with the smallest no. of tracks (to minimize the channel height)?

Horizontal Constraint Graph (HCV)



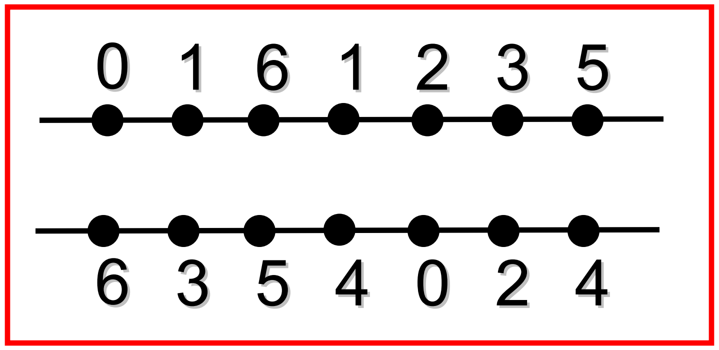




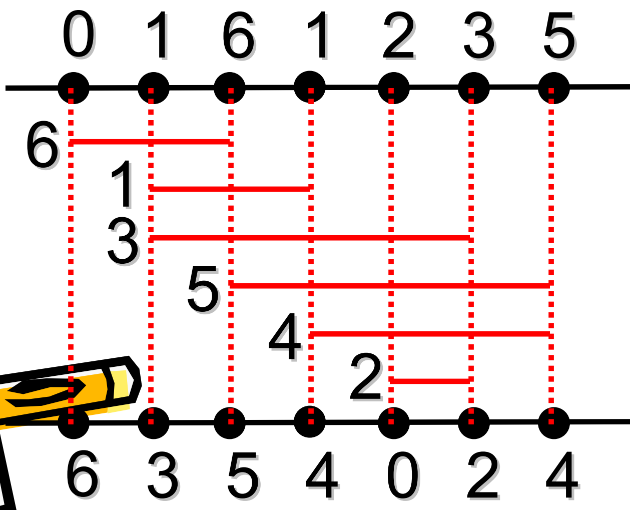
Left-Edge Algorithm

1. Sort the horizontal segments of the nets in increasing order of their <u>left</u> end points.
2. Place them one by one **greedily** on the bottommost available track.

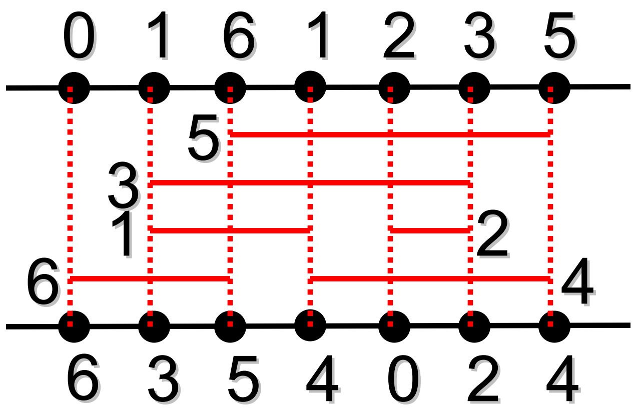
Left-Edge Algorithm



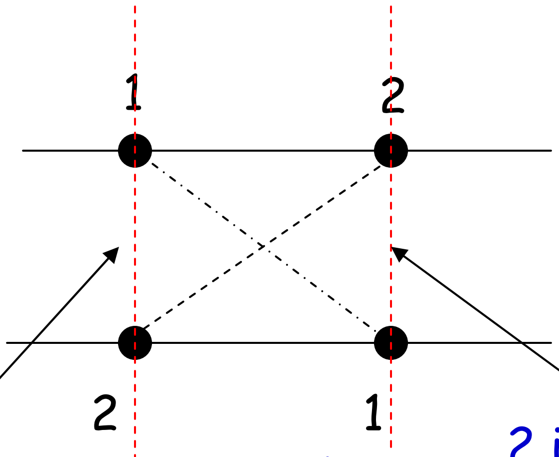
1. Sort by left end points.



2. Place nets greedily.

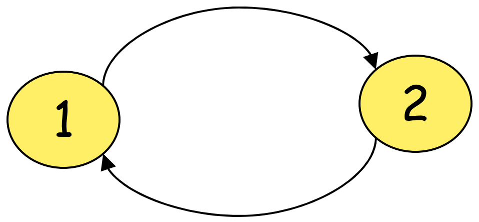


Vertical Constraint Graph and Doglegs

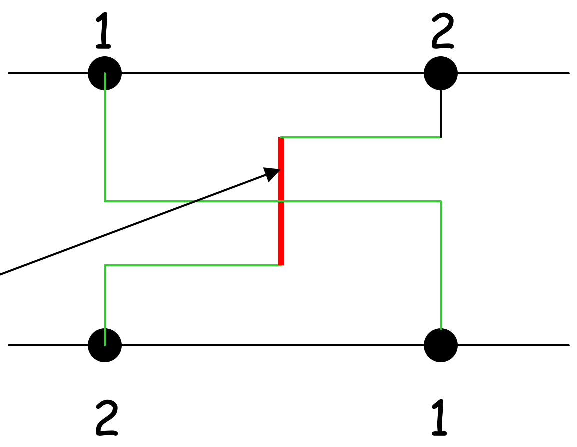


1 imposes a vertical constraint on 2, as top terminal belongs to 1 and bottom terminal belongs to 2

2 imposes a vertical constraint on 1



VCG : Cycle

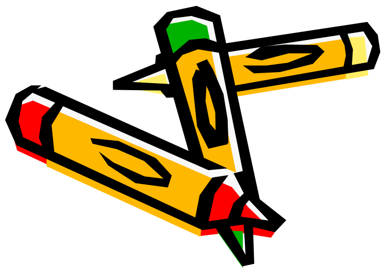


Dogleg

The Cadence Tutorial

Silicon Ensemble (Cadence)

- LEF: Cell boundaries, pins, routing layer (metal) spacing and connect rules.
- DEF: Contains netlist information, cell placement, cell orientation, physical connectivity.
- GCF: Top-level timing constraints handed down by the front end designer are handed to the SE, using PEARL.

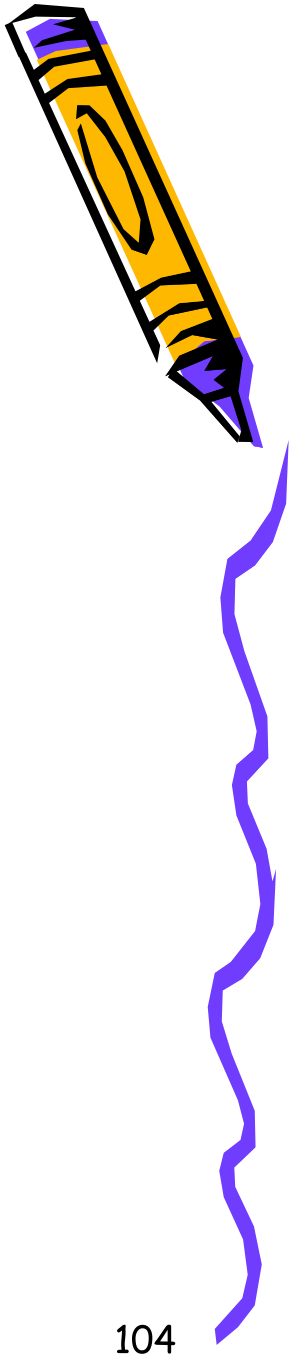


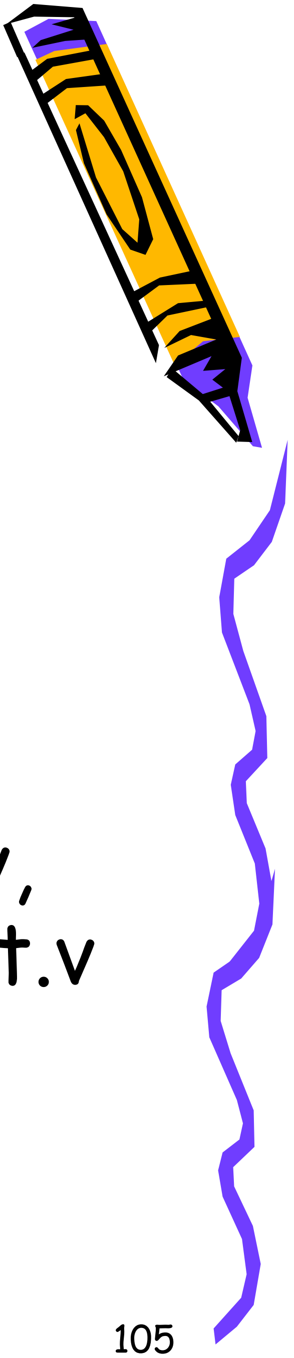
The files required

- Pre-running file:
- se.ini- initialization file for SE.

- Create the following directories:
- lef, def, verilog (netlist) , gcf.

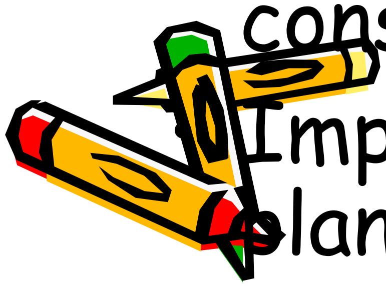
- Type seultra –m=300 &, opens SE in graphical mode.





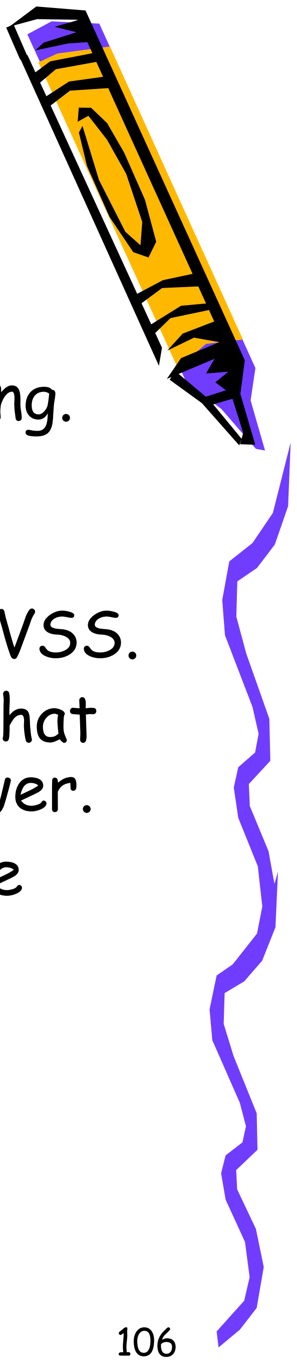
Importing required files

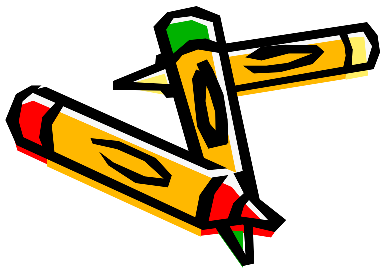
- Import LEF (in the order given):
- header.lef, xlitecore.lef, c8d_40m_dio_00.lef
- Import gcf file:
- Import verilog netlist, xlite_core.v, c8d_40m_dio_00.v, padded_netlist.v
- Import the gcf file as system constraints file.
- Import the .def file for the floor-planning



Structure of a Die

- A Silicon die is mounted inside a chip package.
- A die consists of a logic core inside a power ring.
- Pad-limited die uses tall and thin pads which maximises the pads used.
- Special power pads are used for the VDD and VSS.
- One set of power pads supply one power ring that supplies power to the I/O pads only: Dirty Power.
- Another set of power pads supply power to the logic core: Clean Power.



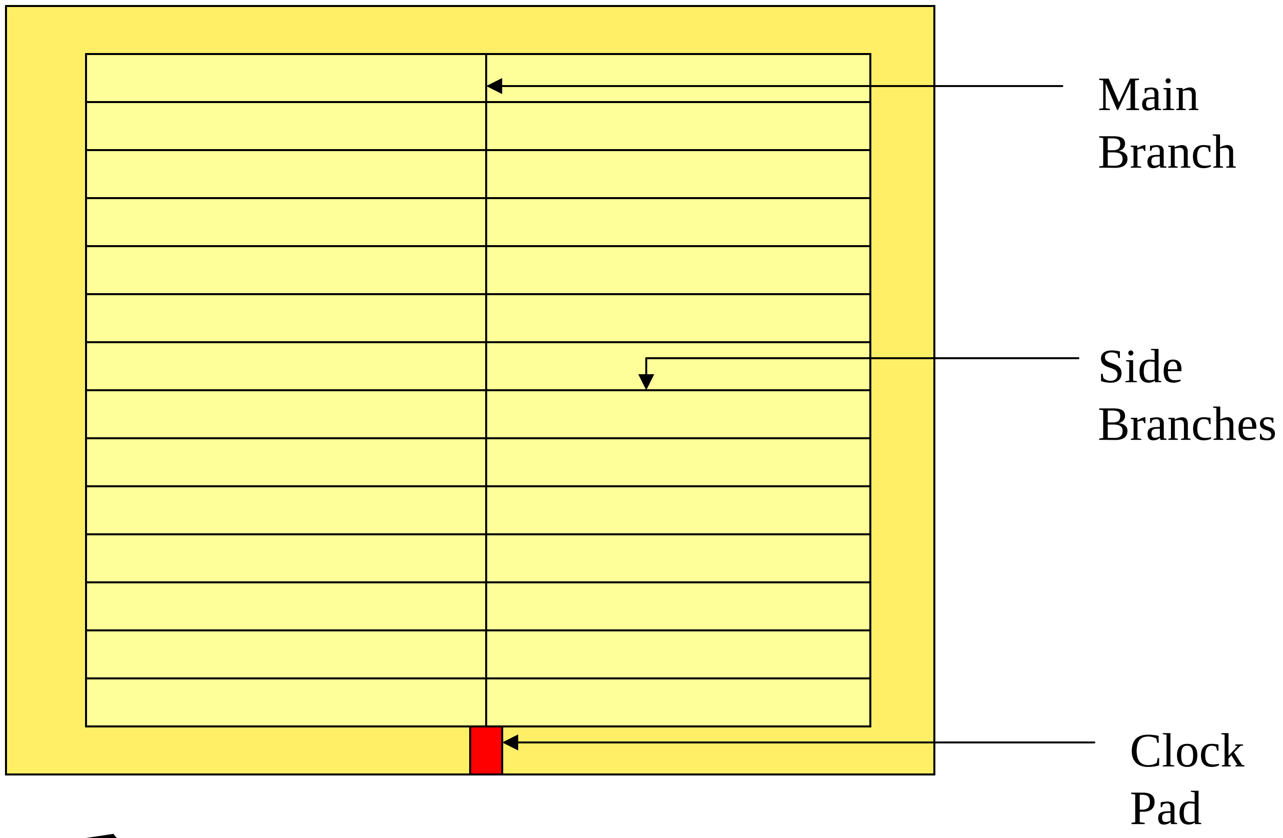


- Dirty Power: Supply large transient current to the output transistor.
- Avoids injecting noise into the internal logic circuitry.
- I/O Pads can protect against ESD as it has special circuit to protect against very short high voltage pulses.

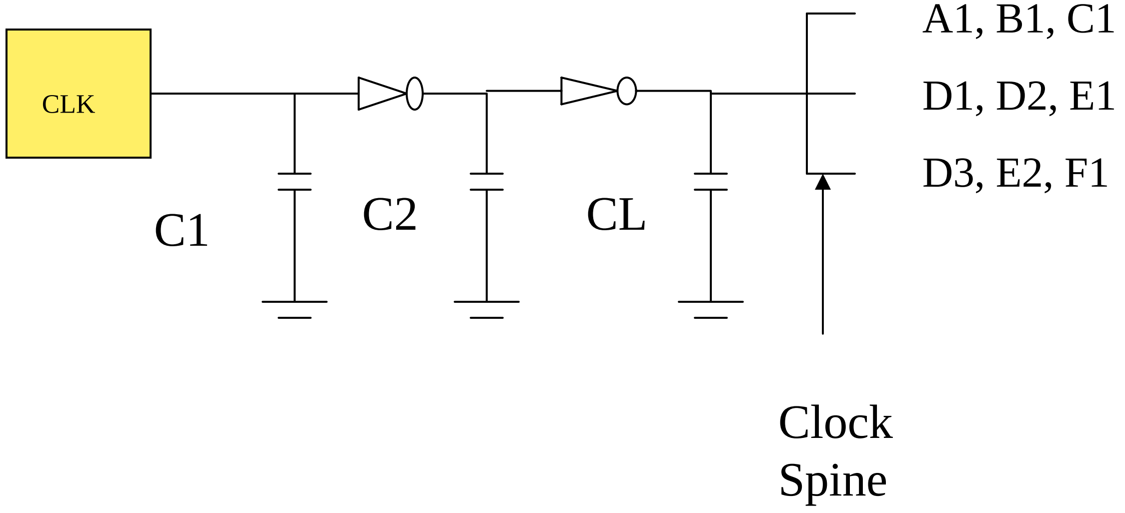
Design Styles

- PAD limited design: The number of PADS around the outer edge of the die determines the die size , not the number of gates.
- Opposite to that we have a core-limited design.

Concept of clock Tree



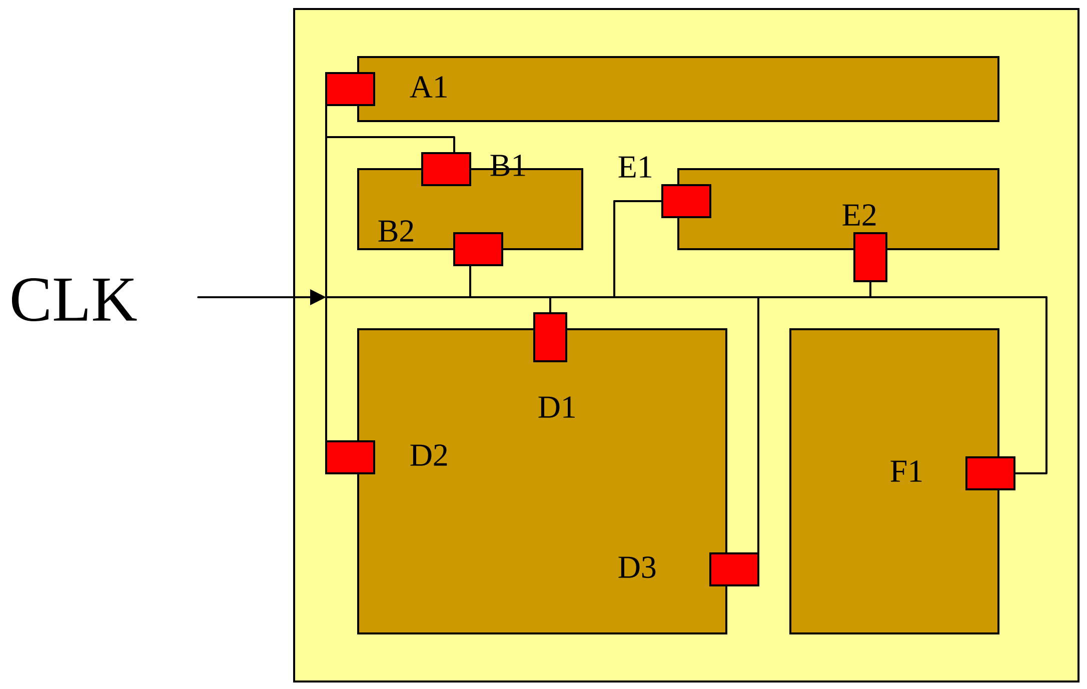
CLOCK DRIVER



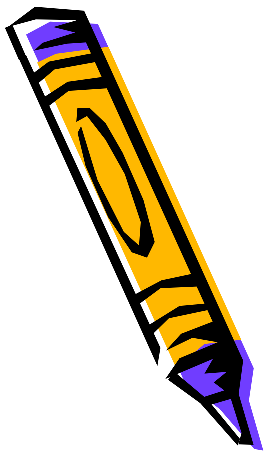
An important result:

The delay through a chain of CMOS gates is minimized when the ratio between the input capacitance C1 and the load C2 is about 3.

Clock and the cells

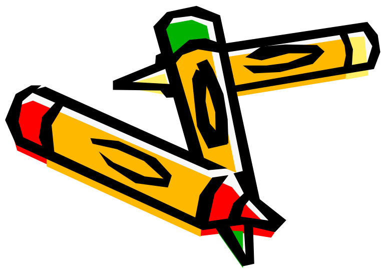


- All clocked elements are driven from one net with a clock spine, **skew** is caused by differing interconnect delays and loads (fanouts ?).
- If the clock driver delay is much larger than the inter-connect delay, a clock spline achieves minimum skew but with **latency**.
- Spread the power dissipation through the chip.
- Balance the rise and the fall time.



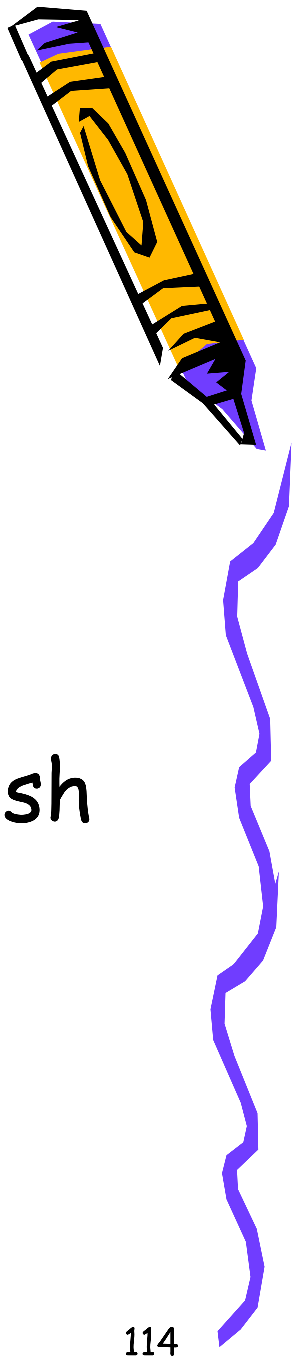
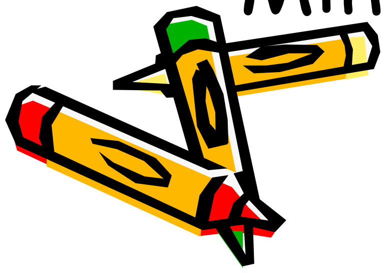
Placement

- Row based ASICS.
- Interconnects run in horizontal and vertical directions.
- Channel Capacity: Maximum number of horizontal connections.
- Row Utilization

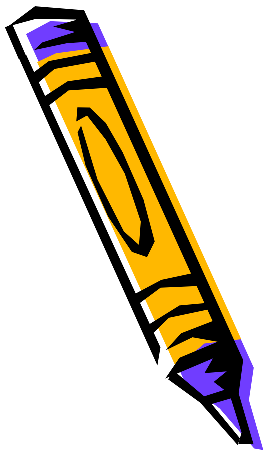


Routing

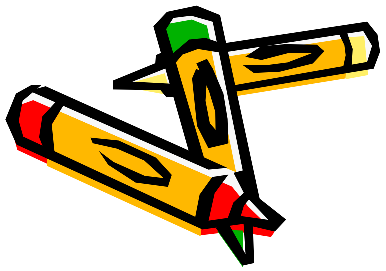
- Minimize the interconnect length.
- Maximize the probability that the detailed router can completely finish the job.
- Minimize the critical path delay.

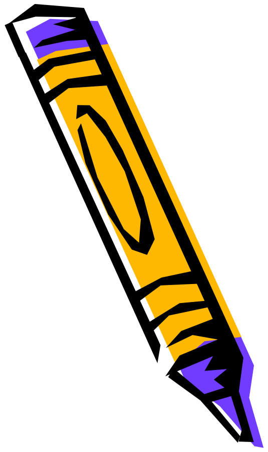
Conclusion: Our backend flow



1. Loading initial data.
2. Floor-planning
3. I/O Placing
4. Planning the power routing : Adding Power rings , stripes
5. Placing cells
6. Placing the clock tree.
7. Adding filler cells.
8. Power routing : Connect the rings to the follow pins of the cells.
9. Routing (Global and final routing)
10. Verify Connectivity, geometry and antenna violations.
11. Physical verification (DRC and LVS check using Hercules).



Thank You



Main references

- Algorithms for VLSI Physical Design Automation (Hardcover) by Naveed A. Sherwani
- Application-Specific Integrated Circuits, M. J. Sebastian Smith
- Silicon-Ensemble Tool, Cadence®