Partitioning in VLSI Physical Design

Parthasarathi Dasgupta

MIS Group

Indian Institute of Management Calcutta

Why Partitioning?

Partitioning - The process of decomposition of a large system into independent manageable subsystems.

Why partition a circuit ?

- Complex system with a large number of components
- Efficient design by breaking into smaller subsystems
- Each subsystem can be designed independently and concurrently
- Decompose the system so as to preserve the original functionality
- Minimize interface across the partition
- Time for decomposition should be small

References and Copyright

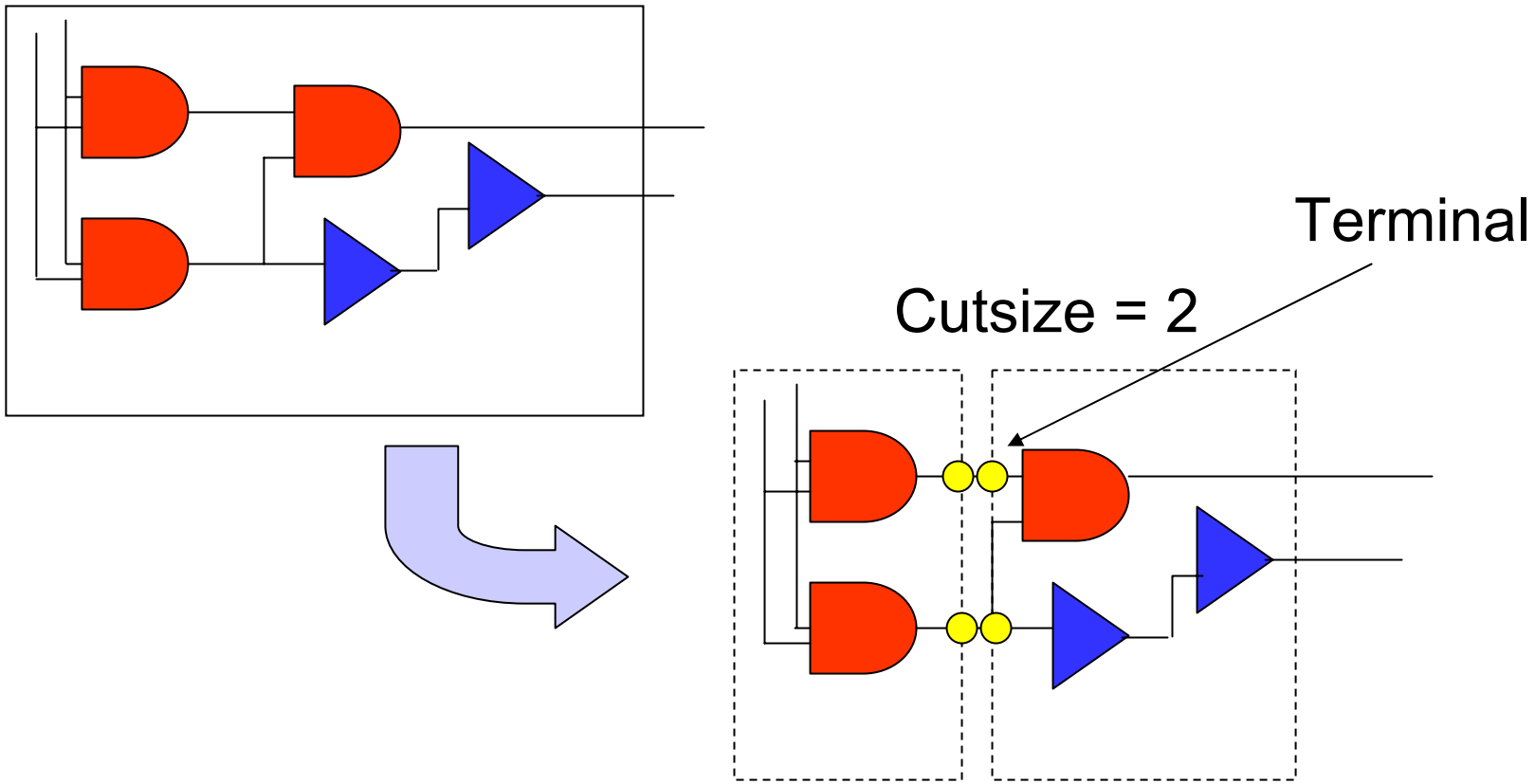
Some of the Slides used (and suitably modified):

– [©Bazargan] © Kia Bazargan;
Department of EE & CS, Univ of Minnesota

Partitioning

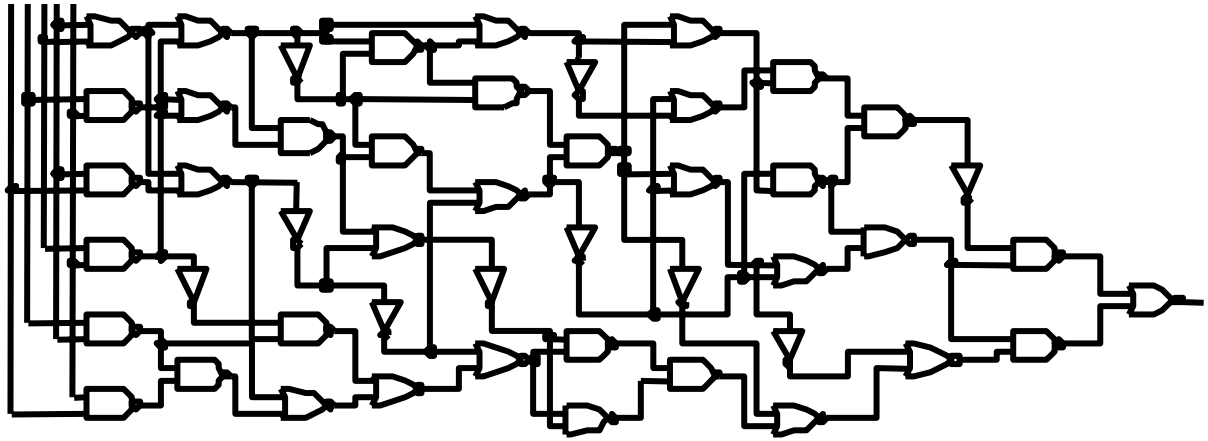
- Decomposition of a complex system into smaller subsystems
  - Done hierarchically
  - Partitioning done until each subsystem has manageable size
  - Each subsystem can be designed independently
- Interconnections between partitions minimized
  - Less hassle interfacing the subsystems
  - Communication between subsystems usually costly

Partitioning of a Circuit: A Simple Example



Partitioning of a Circuit: Realistic Example

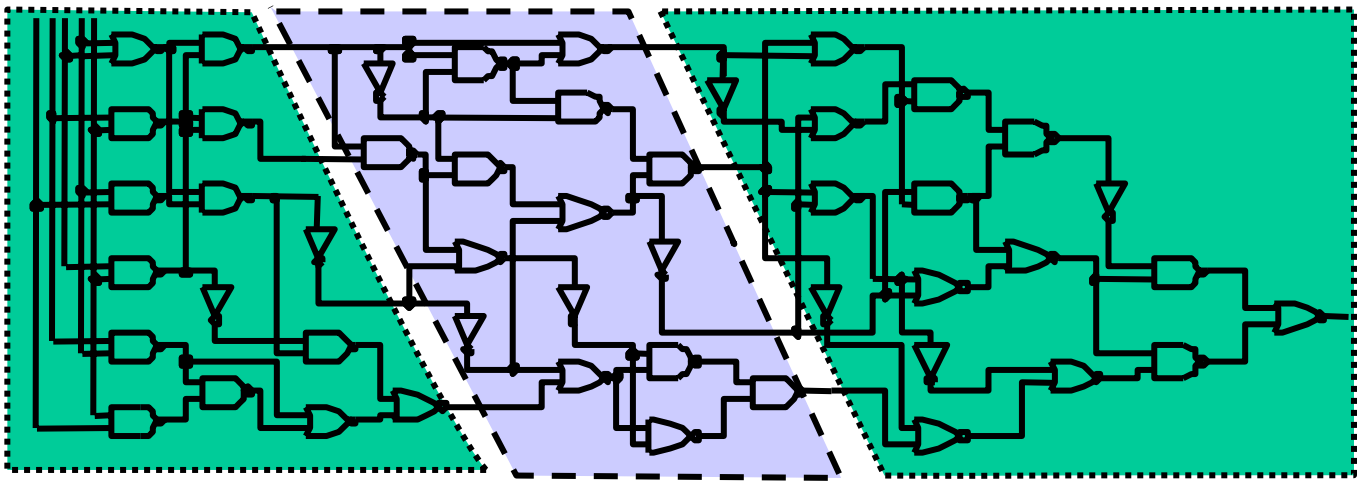
Problem Input size: 48



Cut 1=4
Size 1=15

Cut 2=4
Size 2=16

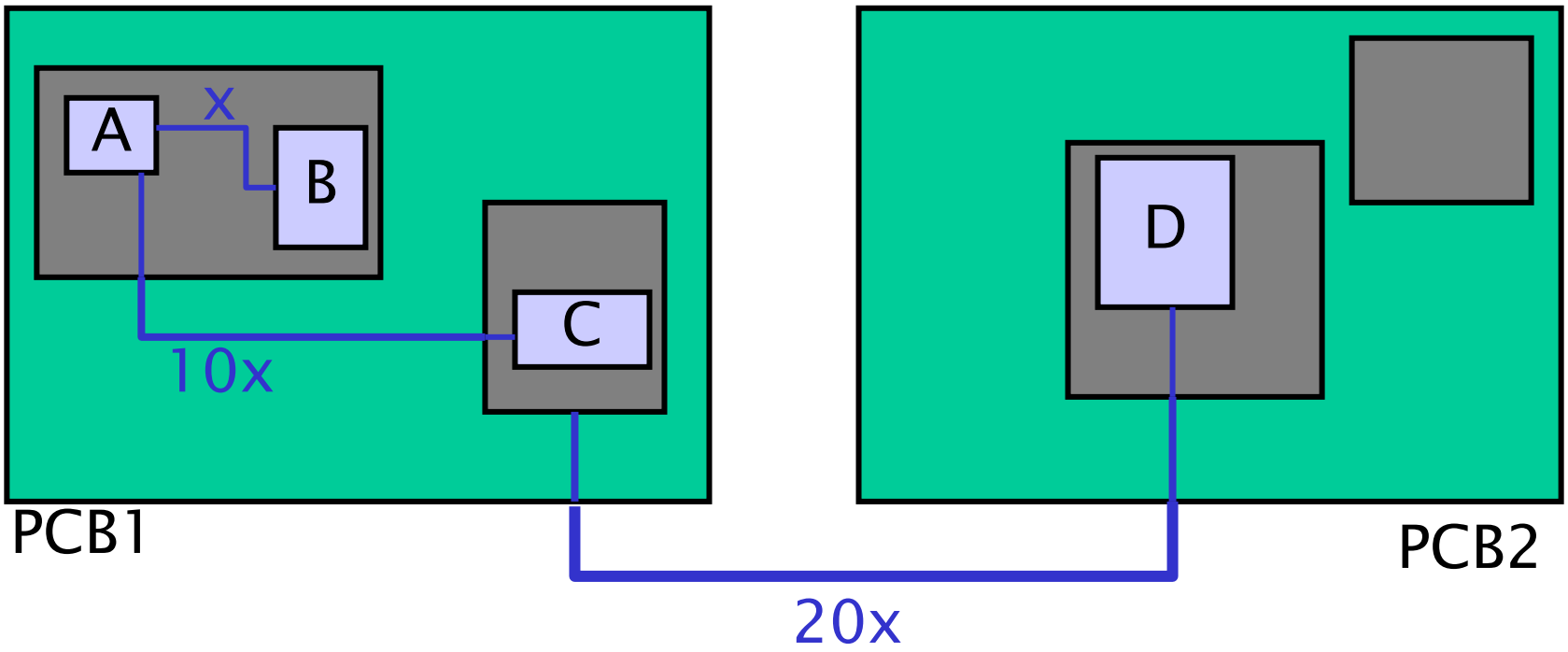
Size 3=17



Hierarchical Partitioning

- Levels of partitioning:
  - System-level partitioning:
    Each sub-system can be designed as a single PCB
  - Board-level partitioning:
    Circuit assigned to a PCB is partitioned into sub-circuits each fabricated as a VLSI chip
  - Chip-level partitioning:
    Circuit assigned to the chip is divided into manageable sub-circuits
    NOTE: physically not necessary

Delay at Different Levels of Partitions



Partitioning: Formal Definition

- Input
  - Graph or hypergraph
  - Usually with vertex weights (sizes)
  - Usually weighted edges
- Constraints
  - Number of partitions (K-way partitioning)
  - Maximum capacity of each partition
    OR
    maximum allowable difference between partitions
- Objective
  - Assign nodes to partitions subject to constraints
    s.t. the cutsize is minimized
- Tractability
  - Is NP-complete

Bipartitioning Formulations: Notations

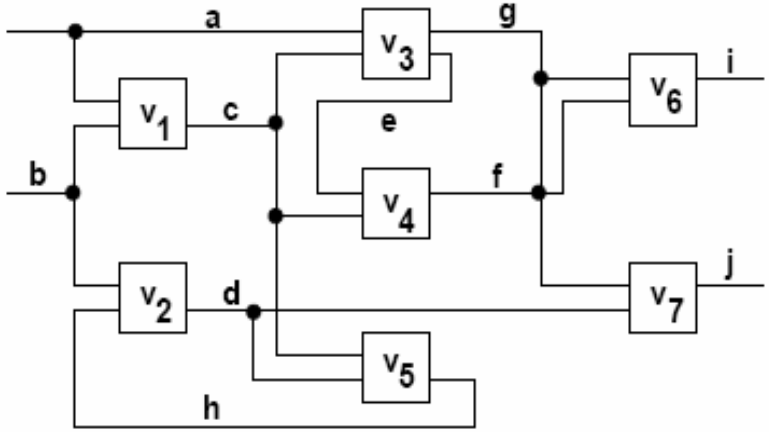
- A net n is cut by a cluster C if at least one, but not all, pins of n is in C.
- We use E(C) to denote the set of nets cut by a cluster C.
- We use w(C) to denote the no. of cells assigned to a cluster C.

Some Bipartitioning Formulations

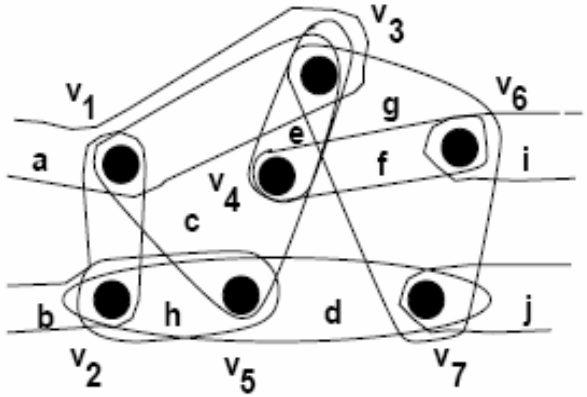
- Min-Cut Bi-partitioning:
  - Objective : Minimize $F(P_2) = |E(C_1)| = |E(C_2)|$
- Min-Cut Bisection:
  - Objective : Minimize $F(P_2) = |E(C_1)| = |E(C_2)|$
  - Constraint : $|w(C_1) - w(C_2)| \leq \varepsilon$
- Size-Constrained Min-Cut Bipartitioning:
  - Objective : Minimize $F(P_2) = |E(C_1)| = |E(C_2)|$
  - Constraint: $L \leq w(C_1), w(C_2) \leq U$
- Minimum Ratio Cut Bipartitioning:
  - Objective : Minimize $F(P_2) = |E(C_1)| / (w(C_1) \times w(C_2))$

Hyper-Graph Representations

Definition: A *hyper-edge* is an interconnection of a set of (> 2) distinct vertices.



(a)

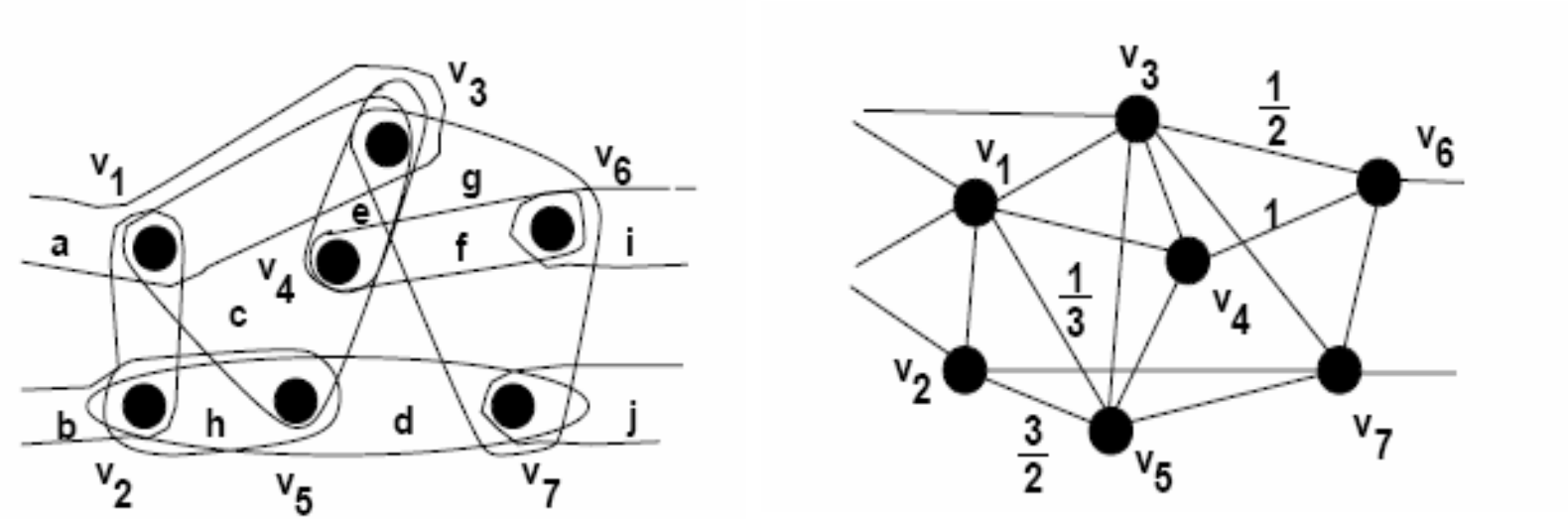


(b)

A circuit with 7 modules and 10 signal nets:

(a) circuit diagram with all inputs on the left side of the modules and all outputs on the right side
(b) the hyper-graph representation

Hyper-Graph to Graph Representations



(a) the hyper-graph representation
(b) the weighted graph representation using the standard clique net model with uniform edge weight

Some Partitioning Methods

- Move-based approaches
- Min-cut / max-flow algorithms
  - Ford-Fulkerson – for unconstrained partitions
- Clustering approaches
- Simulated annealing
- Genetic algorithm

Move-based Algorithm

- Iteratively explores the space of feasible solutions according to a *neighborhood operator*

- Methods include
  - *greedy*
  - *iterative exchange*
  - *simulated annealing*
  - *evolutionary algorithms*

An Iterative Exchange Scheme

Input

a) An initial partition

b) A set of nets (Netlist)

Output

Improved partition with an improved netlist

Method

Starting with the initial partition, iterate continuously, modifying the partitions gradually until the partitions can not be improved further

Group-Migration Algorithms

- Start with an initial partition, usually generated randomly.
- Move components between partitions to improve.
- Deterministic method which is often trapped at a local minimum.

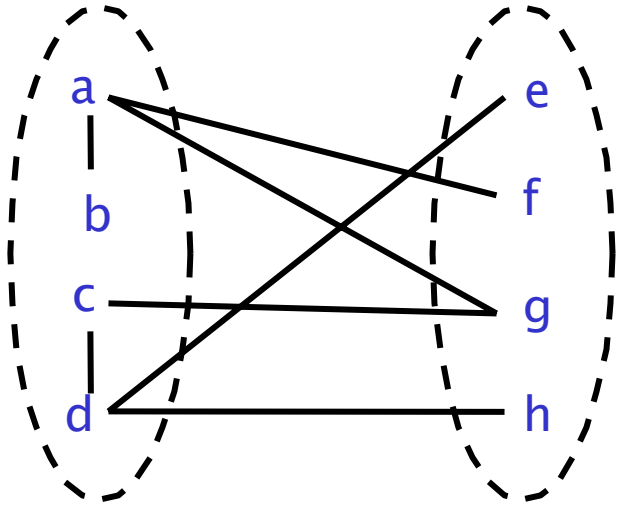
Kernighan-Lin (KL) Algorithm

- On non-weighted graphs
- An iterative improvement technique
- A two-way (bisection) partitioning algorithm
- The partitions must be balanced (of equal size)
- Iterate as long as the cutsize improves:
  - Find a pair of vertices that result in the largest decrease in cutsize if exchanged
  - Exchange the two vertices (potential move)
  - “Lock” the vertices
  - If no improvement possible, and still some vertices unlocked, then exchange vertices that result in smallest increase in cutsize

Kernighan-Lin (KL) Algorithm

- Initialize
  - Bipartition G into V_1 and V_2, s.t., $|V_1| = |V_2| \pm 1$
  - $n = |V|$
- Repeat
  - for i=1 to n/2
    - Find a pair of unlocked vertices $v_{ai} \in V_1$ and $v_{bi} \in V_2$ whose exchange makes the *largest decrease or smallest increase in cut-cost (gain)*
    - Mark v_{ai} and v_{bi} as locked
    - Store the gain g_i.
  - Find k, s.t. $\sum_{i=1..k} g_i = Gain_k$ is maximized
  - If $Gain_k > 0$ then
    move $v_{a1},...,v_{ak}$ from V_1 to V_2 and
    $v_{b1},...,v_{bk}$ from V_2 to V_1.
- Until $Gain_k \leq 0$

Kernighan-Lin (KL) Example

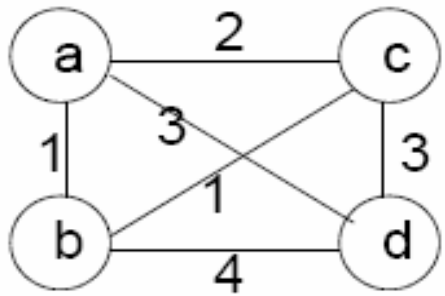


Step No.	Vertex Pair	Gain	Cut-cost
0	--	0	5
1	{ d, g }	3	2
2	{ c, f }	1	1
3	{ b, h }	-2	3
4	{ a, e }	-2	5

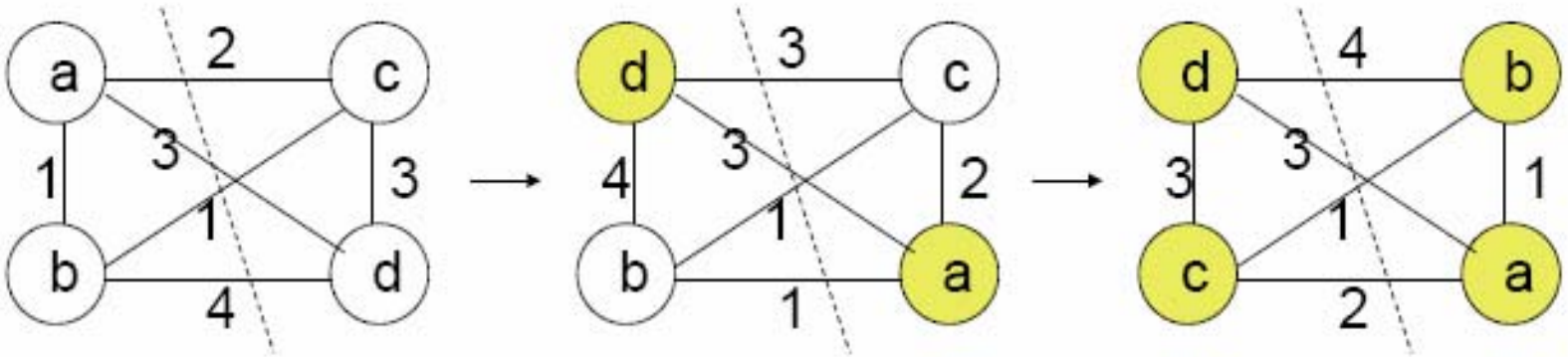
Kernighan-Lin (KL) Example

Example with Edge Weights

	a	b	c	d
a	0	1	2	3
b	1	0	1	4
c	2	1	0	3
d	3	4	3	0



Kernighan-Lin (KL) Example- Pass One

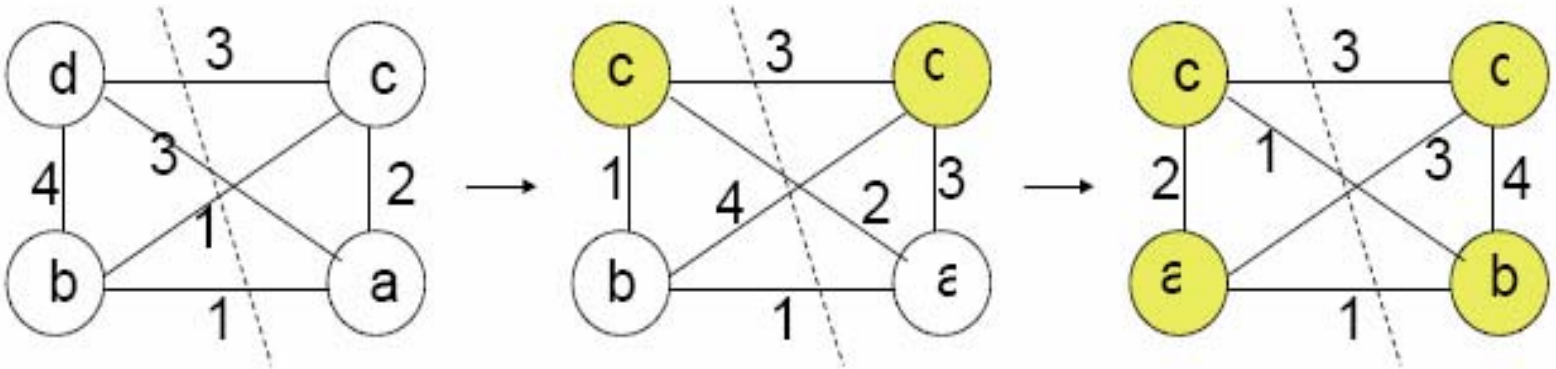


g(a,c) = -1+3-3+1 = 0
g(a,d) = -1+2-3+4 = 2
g(b,c) = -1+4-3+2 = 2
g(b,d) = -1+1-3+3 = 0
$g_1 = 2$

g(b,c) = -4+1-2+3 = -2
$g_2 = -2$

$\therefore G = g_1 = 2 \quad (k = 1)$

Kernighan-Lin (KL) Example- Pass Two



g(a,b) = -2+3-4+1 = -2
g(a,d) = -2+1-4+3 = -2
g(c,b) = -2+3-4+1 = -2
g(c,d) = -2+1-4+3 = -2
$g_1 = -2$

g(a,b) = -3+2-1+4 = 2
$g_2 = 2$

$G = g_1 + g_2 = 0 \quad (k = 2)$

STOP!

Kernighan-Lin (KL) : Analysis

- Time complexity?
  - Inner (for) loop
    - Iterates n/2 times
    - Iteration 1: (n/2) x (n/2)
    - Iteration i: $(n/2 - i + 1)^2$.
  - Passes? Usually independent of n
  - $O(n^3)$

Kernighan-Lin (KL) : Drawbacks?

- Local optimum
- Balanced partitions only
- No weight for the vertices
- High time complexity
- Hyper-edges? Weighted edges?

Fiduccia-Mattheyses (FM) Algorithm

- Modified version of KL
- A single vertex is moved across the cut in a single move
  - Unbalanced partitions
- Vertices are weighted
- Concept of cutsize extended to hypergraphs
- Special data structure to improve time complexity to $O(n^2)$
  - (Main feature)
- Can be extended to multi-way partitioning

C. M. Fiduccia and R. M. Mattheyses, 19th DAC, 1982.

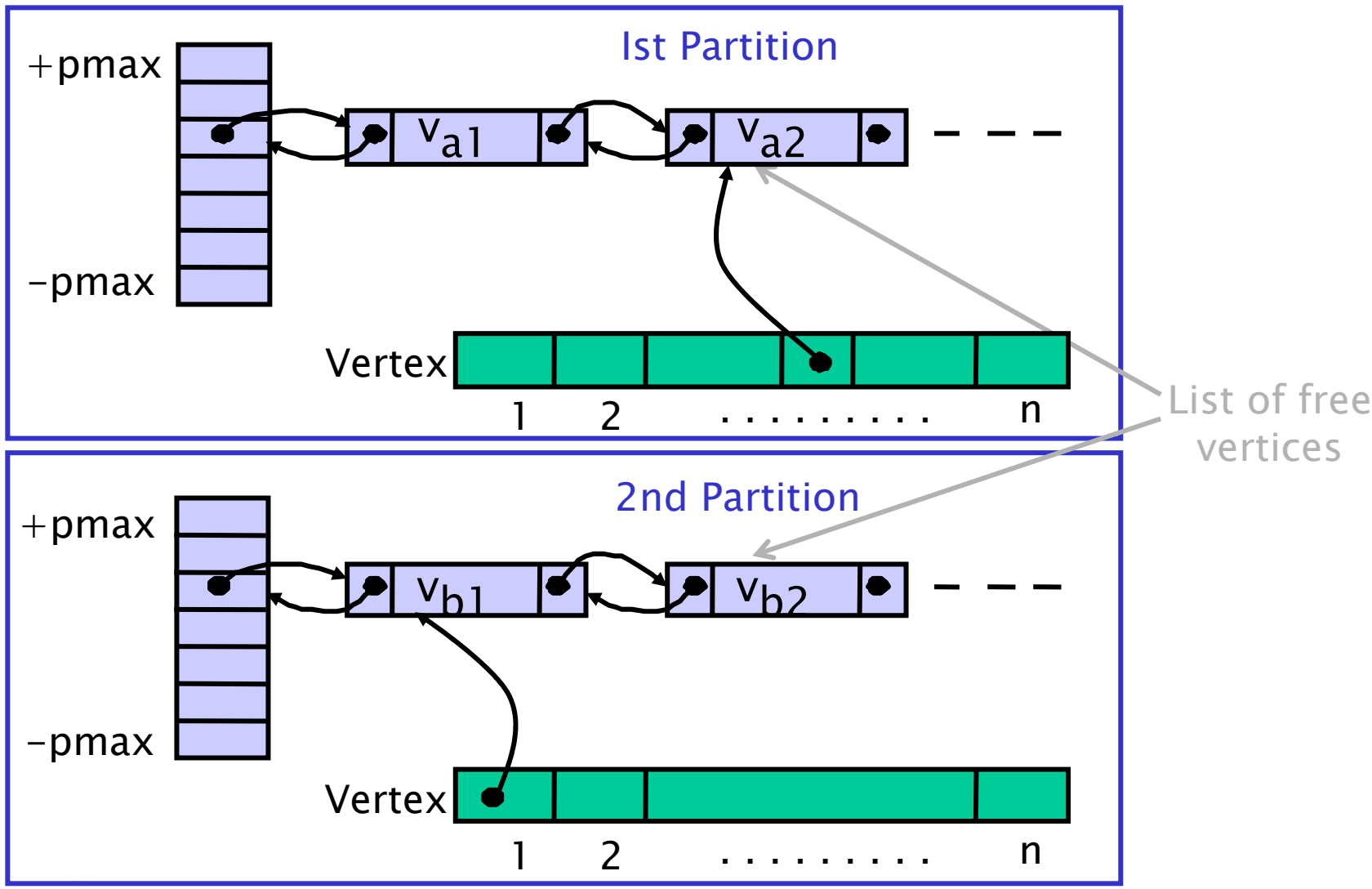
The FM Algorithm: Data Structure

- Pmax
  - Maximum gain
  - $p_{max} = d_{max} . w_{max}$, where
    d_{max} = max degree of a vertex (# edges incident to it)
    w_{max} is the maximum edge weight
  - What does it mean intuitively?

- -Pmax .. Pmax array
  - Index *i* is a pointer to the list of unlocked vertices with gain *i*.

- Limit on size of partition
  - A maximum defined for the sum of vertex weights in a partition (alternatively, the maximum ratio of partition sizes might be defined)

The FM Algorithm: Data Structure



The FM Algorithm: Balanced Partition

- Definition. A maximum vertex weight

  $W \geq w(V)/2 + \max_{v \in V}\{w(v)\}$, where w(v) = weight of vertex v, w(v) = sum of the weights of all vertices
- Balanced partition {A|B} is one having a total vertex weight of at most W, i.e., w(A), w(B) $\leq$ W.

Differences from KL:

Move only one cell each time.

Cells can have different sizes.

Nets can be multi-terminal.

Maintain a balanced partition after every move.

The FM Algorithm: Overview

- Initialize
  - Start with a balanced partition A, B of G
    (can be done by sorting vertex weights in decreasing order, placing them in A and B alternatively)
- Iterations
  - Similar to KL
  - A vertex cannot move if it violates the balance condition
  - Choosing the node to move:
    pick the max gain in the partitions
  - Moves are tentative (similar to KL)
  - When no moves possible or no more unlocked vertices available, the pass ends
  - When no move can be made in a pass, the algorithm terminates

The FM Algorithm

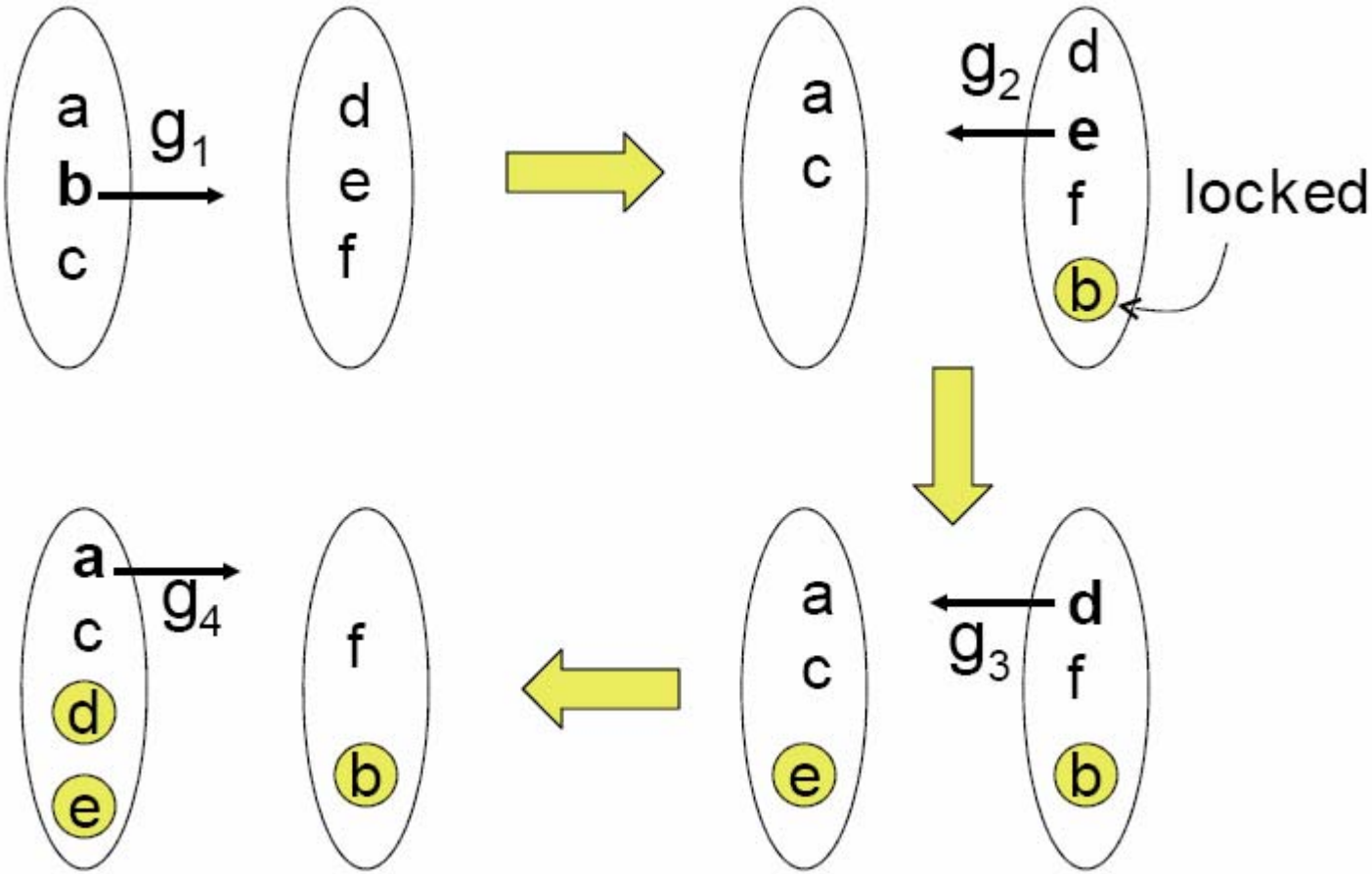
Start with a balanced partition P = {X,Y}.

Repeat

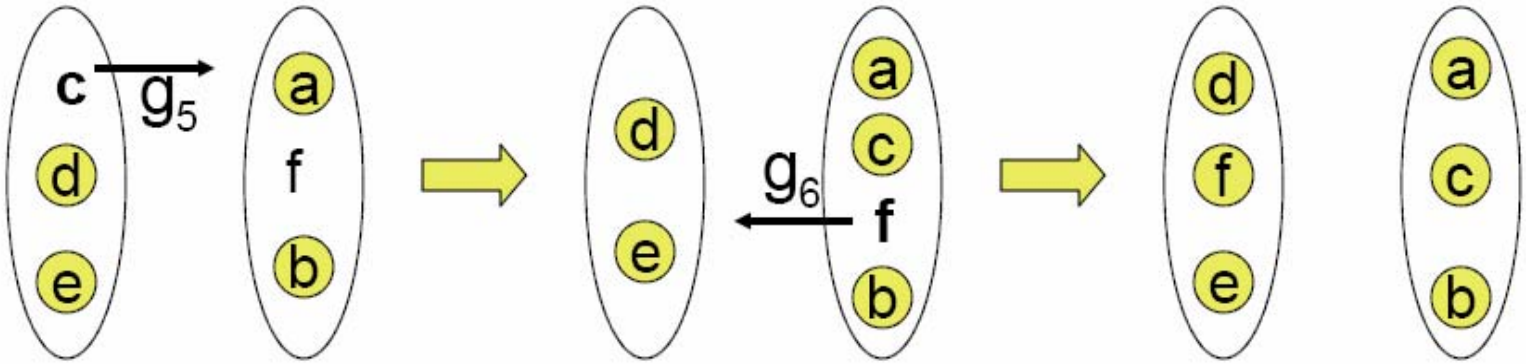
- For i = 1 to n:
  - Choose a free cell b $\in$ X$\cup$Y s.t. moving b to the other side gives the highest gain, gain(b), and moving b preserves balance in P.
  - Move and lock b.
  - Let gi = gain(b).
- Find k s.t. G = g1 + g2 + ….. + gk is maximized and shuffle the cells up to this kth step.

Until G = 0.

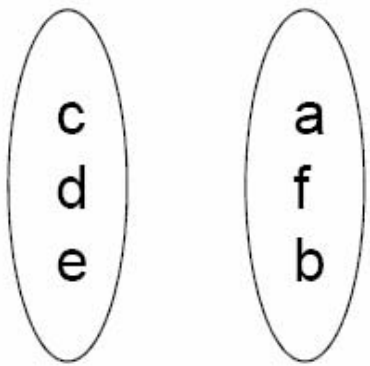
The FM Algorithm: Example



The FM Algorithm: Example

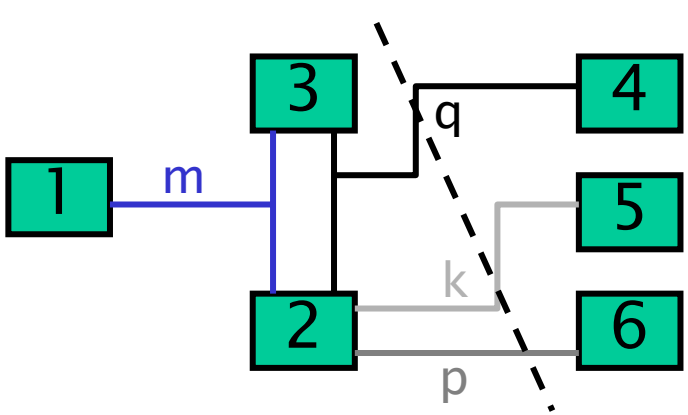


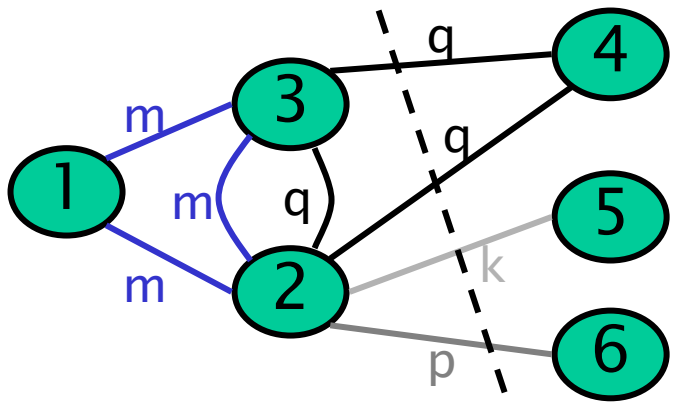
If $G = g_1 + g_2 + g_3 + g_4$ is the largest partial sum, the partition after this pass is:



Why Hyperedges?

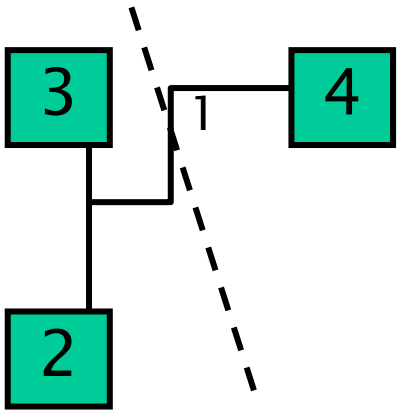
- For multi terminal nets, K-L may decompose them into many 2-terminal nets, but not efficient!
- Consider this example:
- If A = {1, 2, 3} B = {4, 5, 6}, graph model shows the cutsize = 4 but in the real circuit, only 3 wires cut
- Reducing the number of nets cut is more realistic than reducing the number of edges cut



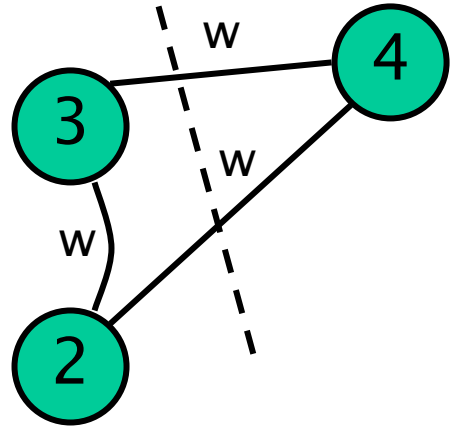


Hyperedge to Edge Conversion

- A hyperedge can be converted to a “clique”.



“Real” cut=1



“net” cut=2w

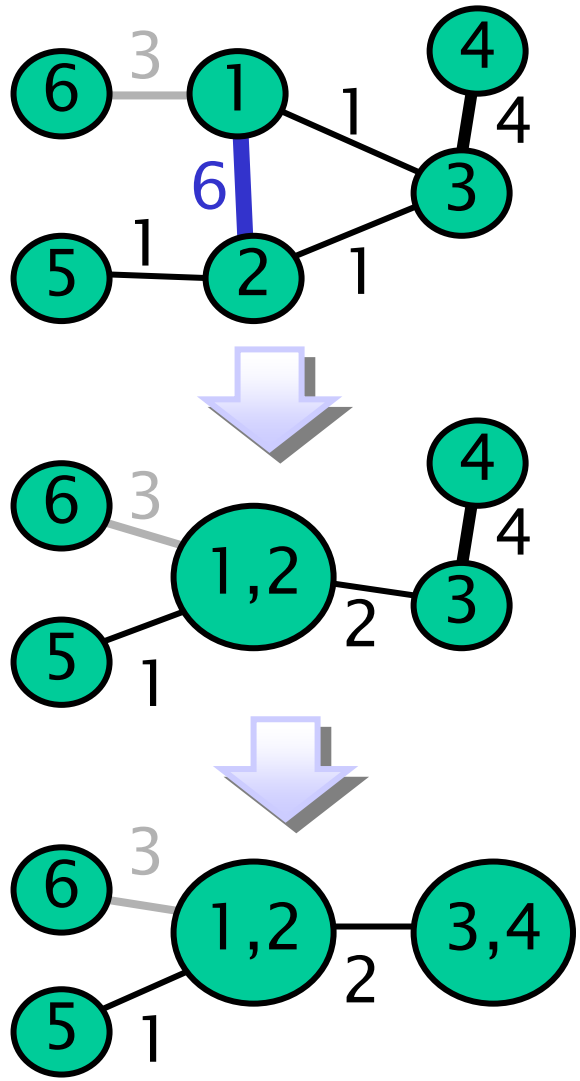
- w=?
  - w=2/(n-1) has been used, also w=2/n
  - Best: w=4/(n^2 – mod(n,2))
    for n=3, w=4/(9-1)=0.5
- Always necessary to convert hyper-edge to edge?

Helpful-Set: A Generalization of FM

- As in FM heuristic, based on local rearrangements.
- Searches for two sets of equal size, one in each part, which will improve cut size if they both change the parts.
- It considers not only single vertices, but also whole sets, to take part in the exchange.

Clustering

- Clustering
  - Bottom-up process
  - Merge heavily connected components into clusters
  - Each cluster will be a new “node”
  - “Hide” internal connections (i.e., connecting nodes within a cluster)
  - “Merge” two edges incident to an external vertex, connecting it to two nodes in a cluster
- Can be a preprocessing step before partitioning
  - Each cluster treated as a single node



Simulated Annealing (SA) in Partitioning

Algorithm SA

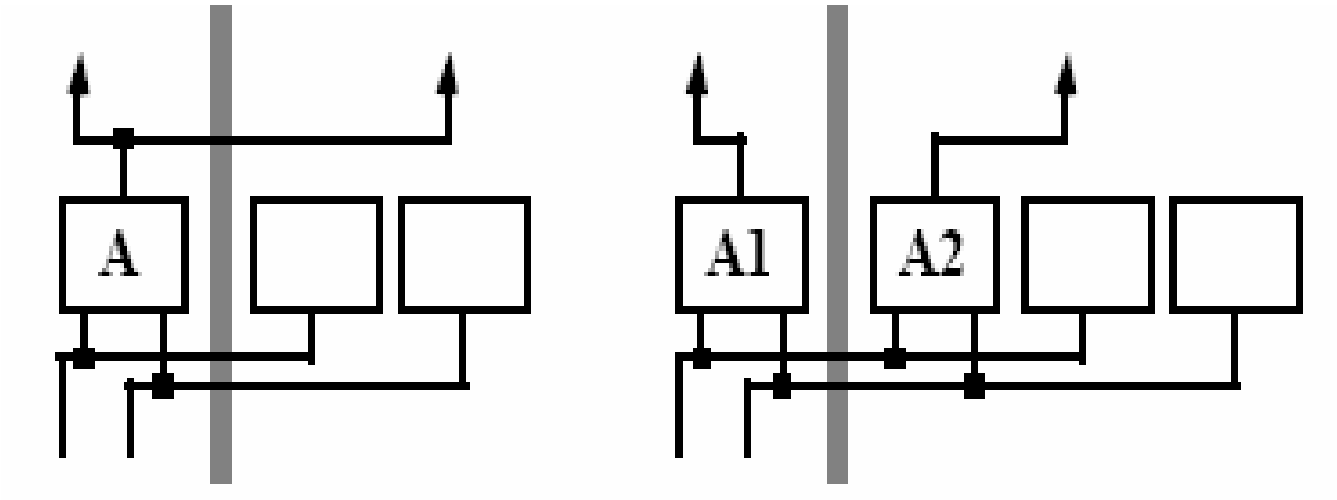
```
begin
  t := t0
  cur_part := init_part;
  cur_score :=  SCORE(cur_part);
  repeat
     repeat
       comp1 = SELECT(part1)
       comp2 = SELECT(part2)
        trial_part = XCHANGE(comp1,comp2,cur_part)
        trial_score :=  SCORE(trial_part)
       δs = trial_score - cur_score
```

SA-based Partitioning (*Contd.*)

```
if (ds < 0) then
        cur_score = trial_score
        cur_part = MOVE(comp1,comp2)
      else
        r = RANDOM(0,1);
        if (r < e^-ds/t) then
          cur_score = trial_score
          cur_part = MOVE(comp1,comp2)
  until(equilibrium at t is reached)
  t = at  (* 0 < a < 1 *)
until (freezing point is reached)
```

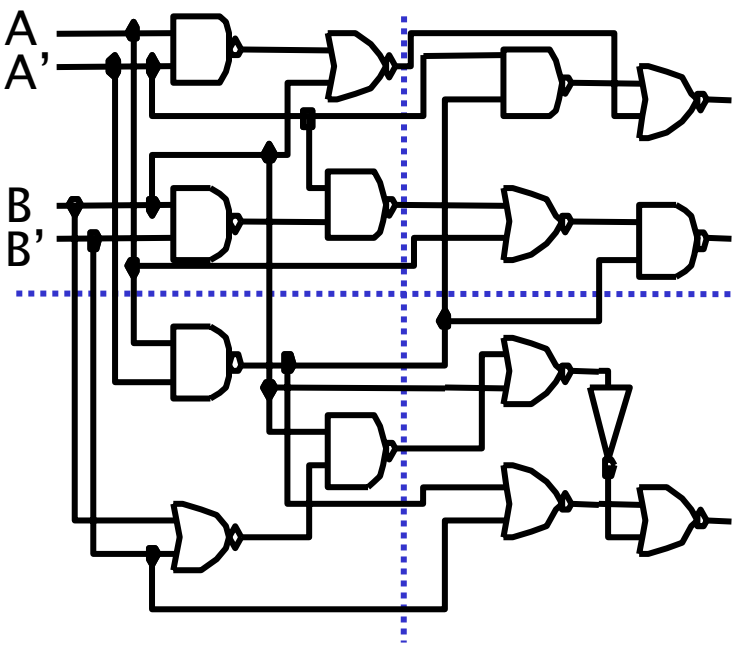
Subgraph Replication to Reduce Cutsize

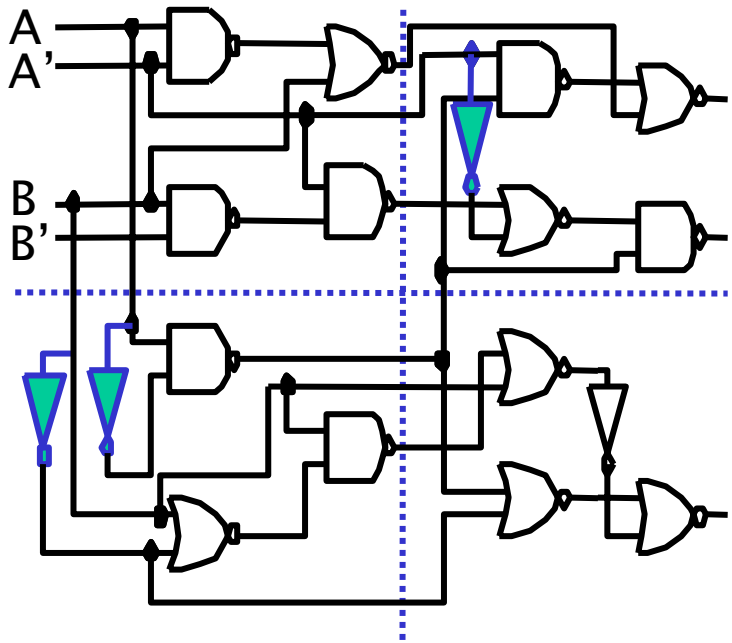
- Vertices are replicated to improve cutsize
- Good results if limited number of components replicated



Node **A** is replicated into **A1** and **A2**, yielding a gain of 1

Subgraph Replication: An Example





Partitioning and Performance

The hypergraph partitioning problem is to divide the nodes of a hypergraph into roughly equal parts; the traditional objective is to minimize cutsize.

In performance-driven partitioning, we also seek to minimize path delay on timing paths.

Motivating Questions

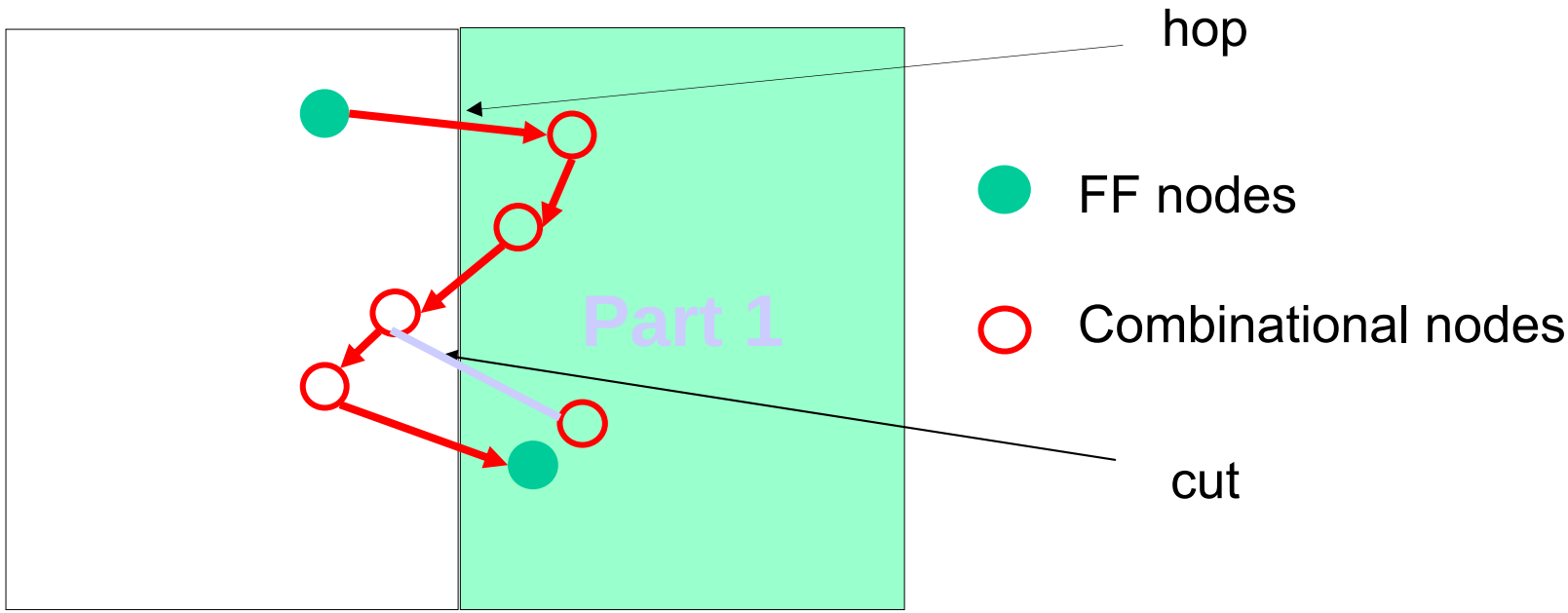
- Can we avoid global timing analysis?
  - Global timing analysis is extremely time-consuming
- Can we improve path delay without significant degrading of cutsize?
  - Need smooth tradeoff between delay and cutsize
- Can we reduce implementation overheads?
  - Previous methods store thousands of critical paths and continuously update them

Why Performance Driven Partitioning?

- Achieving timing closure becomes increasingly difficult in deep-submicron technologies due to non-ideal scaling of interconnect delay.
- Routing alone can no longer solve timing problem, even with aggressive optimizations (buffer insertion, buffer/wire sizing,…).
- Timing needs to be addressed at all design stages.
- Partitioning is a critical step in defining interconnect timing properties, but is traditionally driven by cutsize objective.

Delay Model

Delay = hop_delay + node_delay



hop_delay=Elmore delay
node_delay=constant

Performance Driven Bipartition Problem

Given:

- Hypergraph H=(V,E)
- Area Balance tolerance s (0<s<1), a parameter to control allowable slack in the area constraint
- a, a given parameter which captures tradeoff between cutsize and path delay (hopcount)

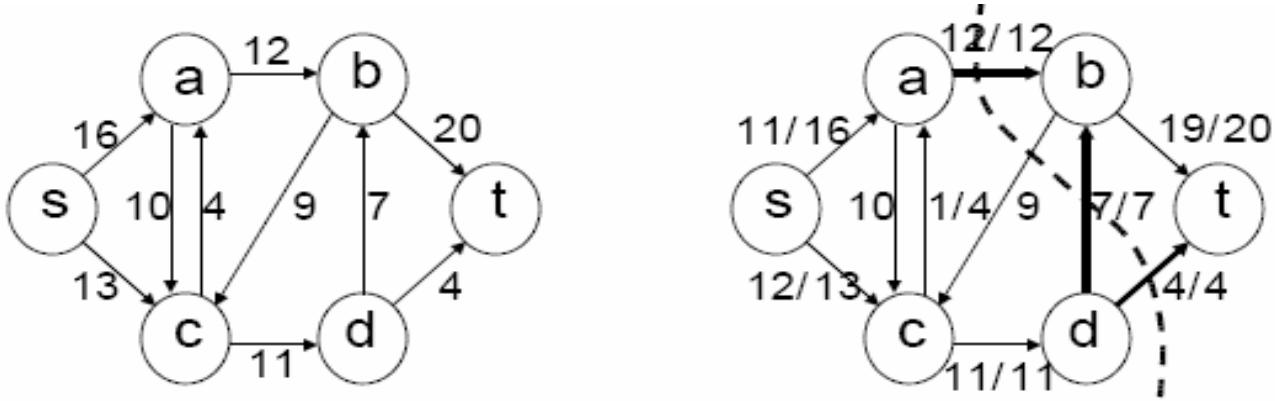
Find:

A bipartition ($V_0|V_1$) which satisfies:

$$(1-s)\frac{A}{2} \leq A_0 \leq (1+s)\frac{A}{2}$$

and minimizes a(cutsize)+(1-a)(Max_hopcount)

Network Flow Technique



min-cut = max-flow

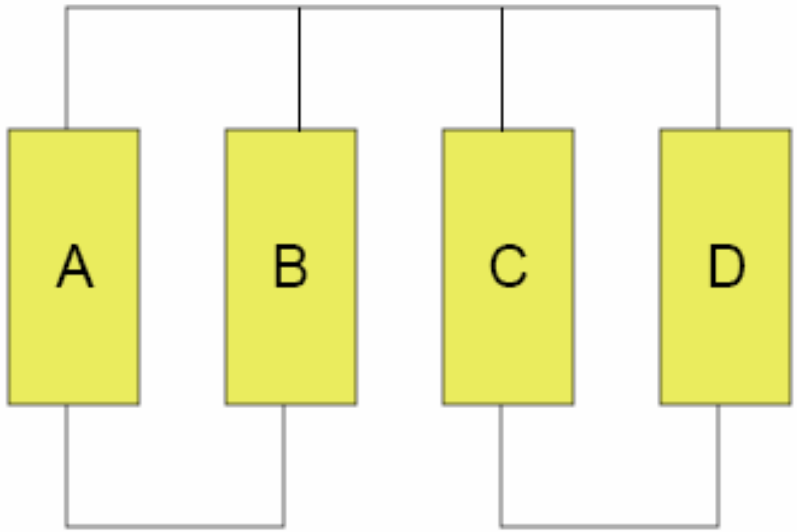
- The network flow technique can find the min-cut bipartition optimally, but not necessarily balanced.
- Apply the algorithm repeatedly to obtain a balanced bipartition.

Network Flow in Partitioning

- We can apply the network flow algorithm in partitioning circuits.
- The biggest problem is that the two partitions may not be balanced.
- The problem of obtaining two balanced partitions with minimum cut is NP-complete.
- However we can apply some heuristics to balance the two partitions.

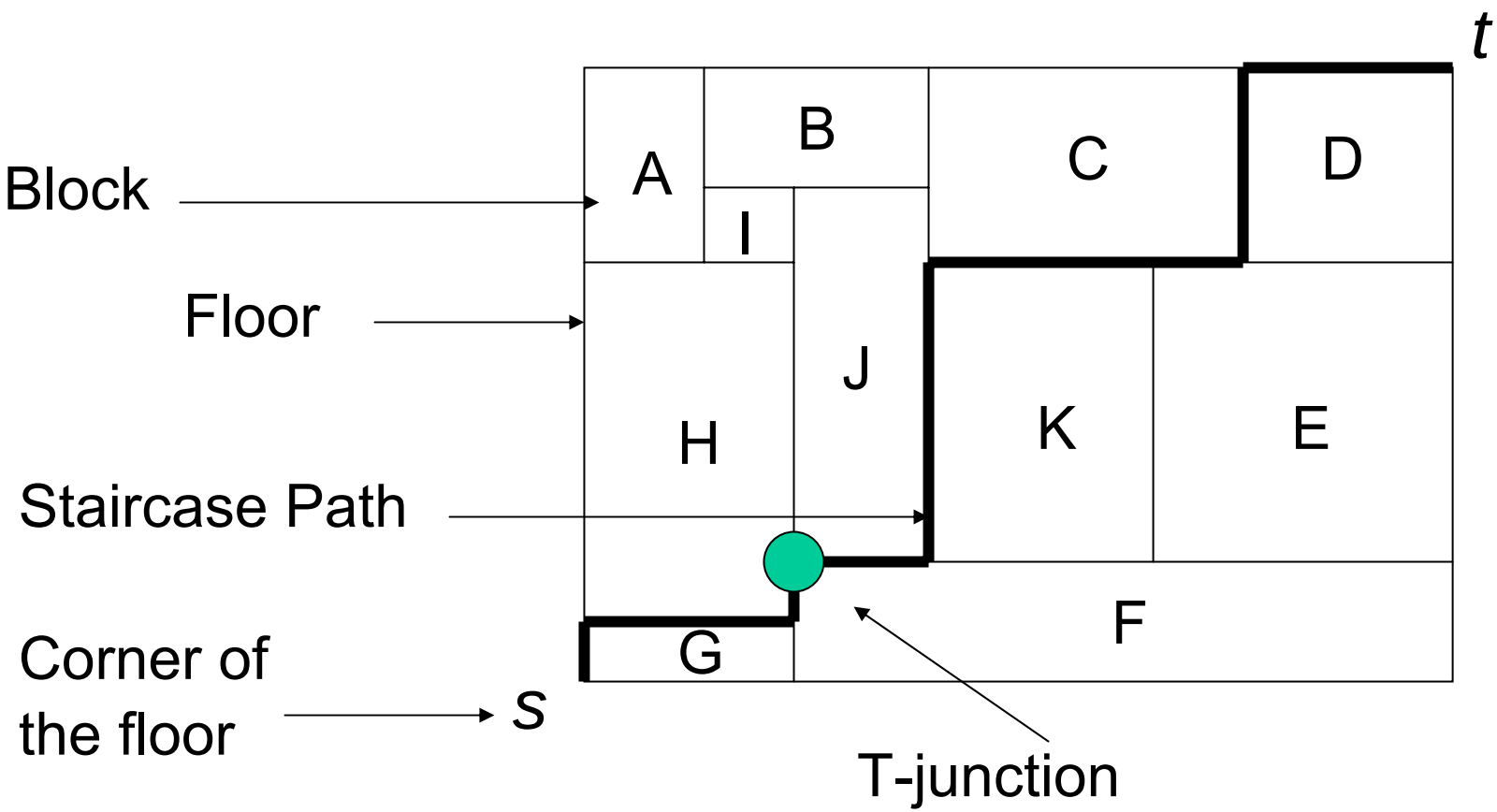
Circuit Representation

- Another problem in applying the network flow technique in circuit partitioning is how to represent a circuit correctly by a graph.



How to represent this netlist by a simple graph?

Monotone Area Bipartition : Preliminaries

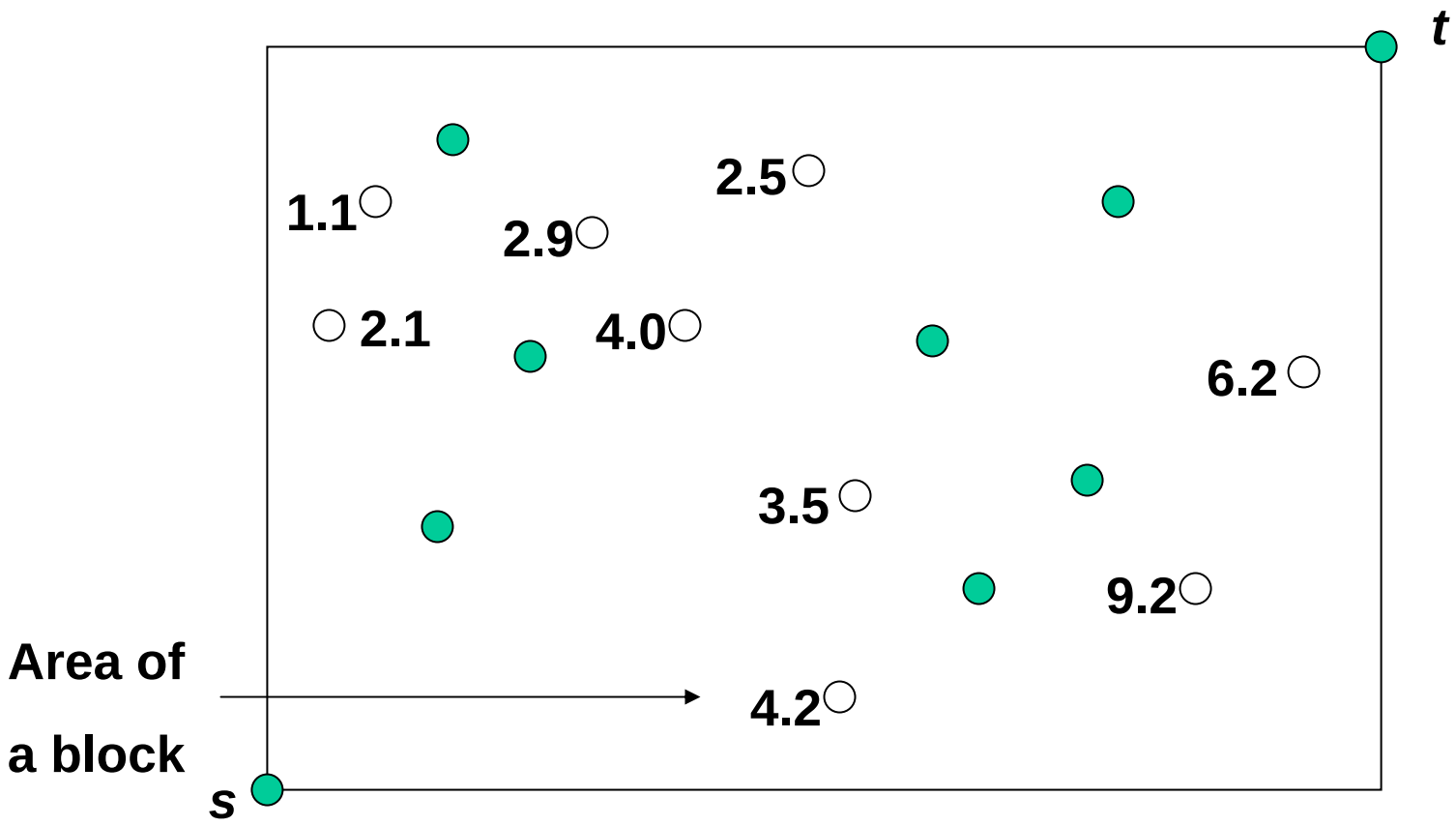


A Placement of Rectangular Blocks

Monotone Area Bipartition : Problem definition

- *Input* : A Placement of Rectangular Blocks with no wasted area, and area of each block.
- *Objective* : To find a Staircase Path through the blocks from one corner of the floor to its diagonally opposite corner such that the difference of the sum of areas of the blocks on its two sides is *minimum.*
- *Constraint*: The staircase path can not pierce through any block.

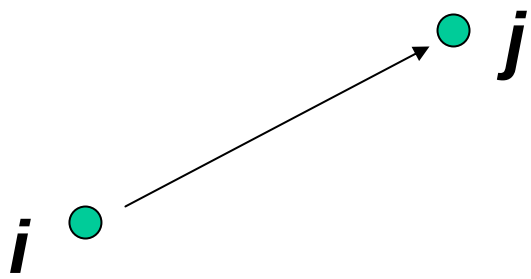
Points with Real Weights



○ Type-A point (Centre of a block)

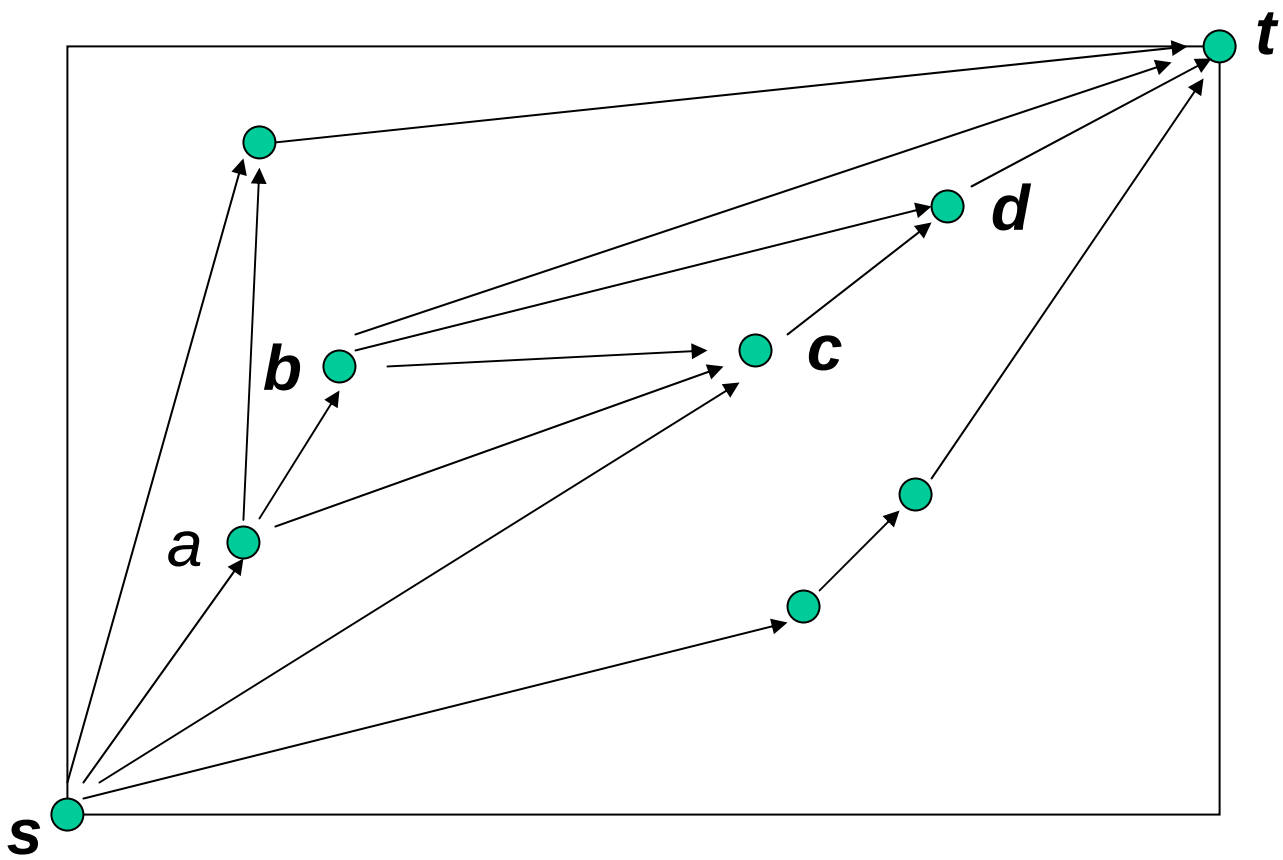
● Type-B point (T-Junction)

Monotone Edge



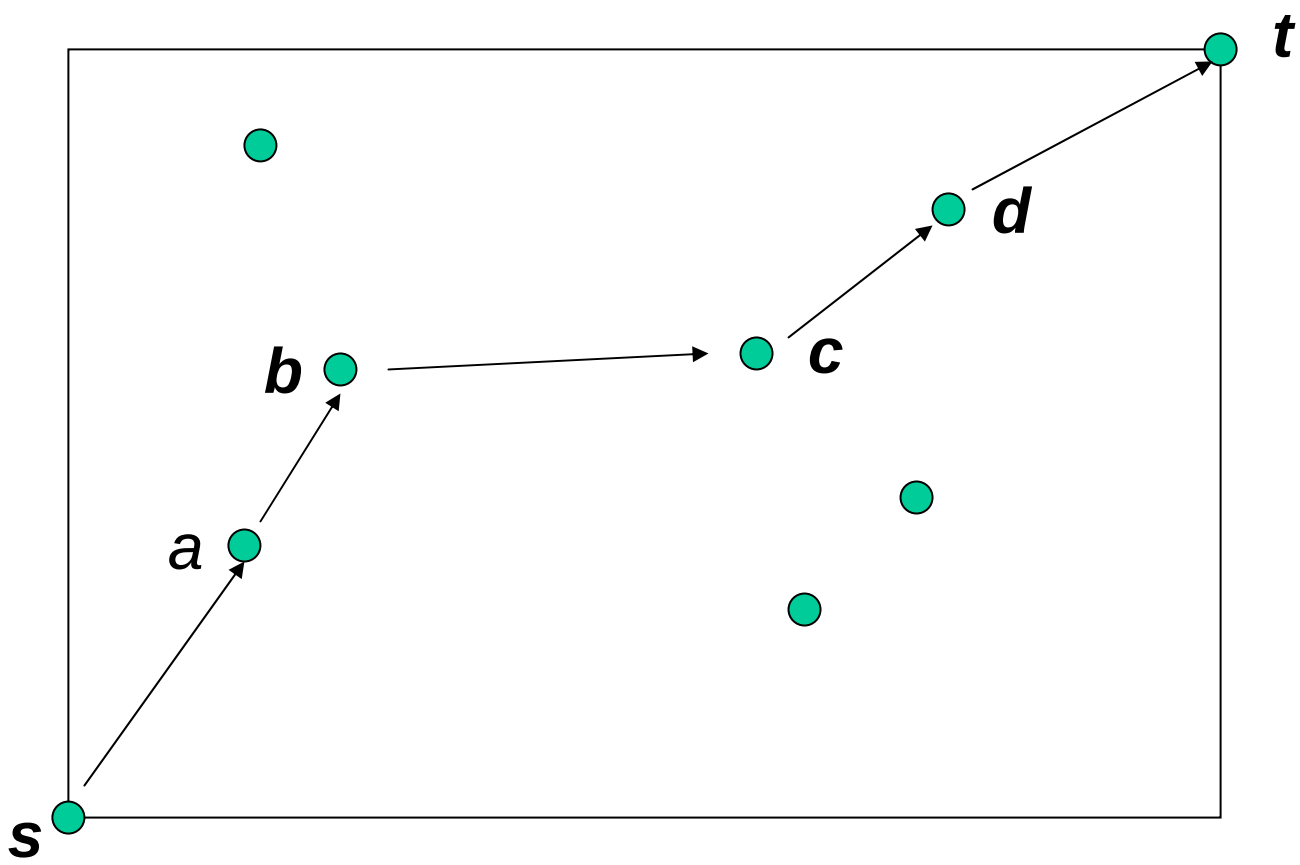
$i \longrightarrow j \Rightarrow x_i \leq x_j,\ y_i \leq y_j$

Monotone Paths



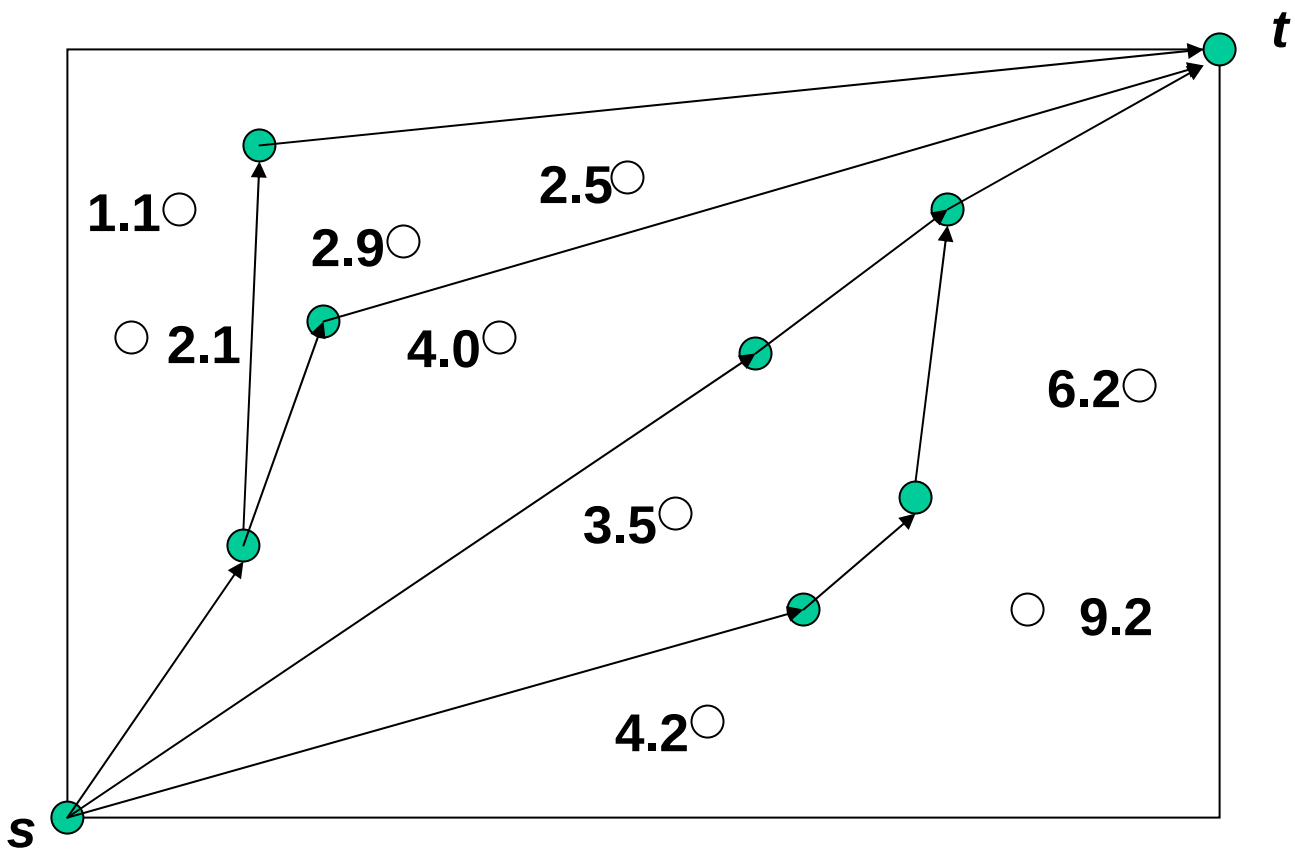
Some Examples of Monotone Paths

Maximal Monotone Path (MMP)



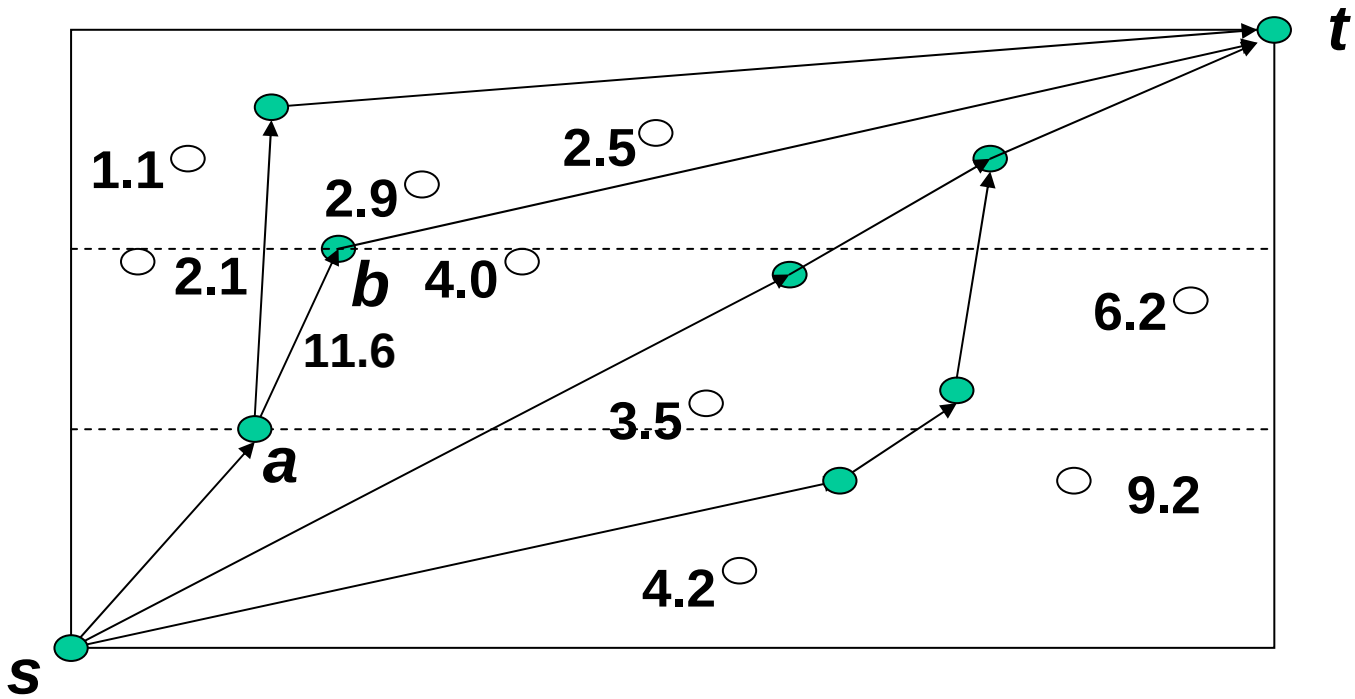
An Example of a Maximal Monotone Path

Maximal Monotone Graph (MMG)



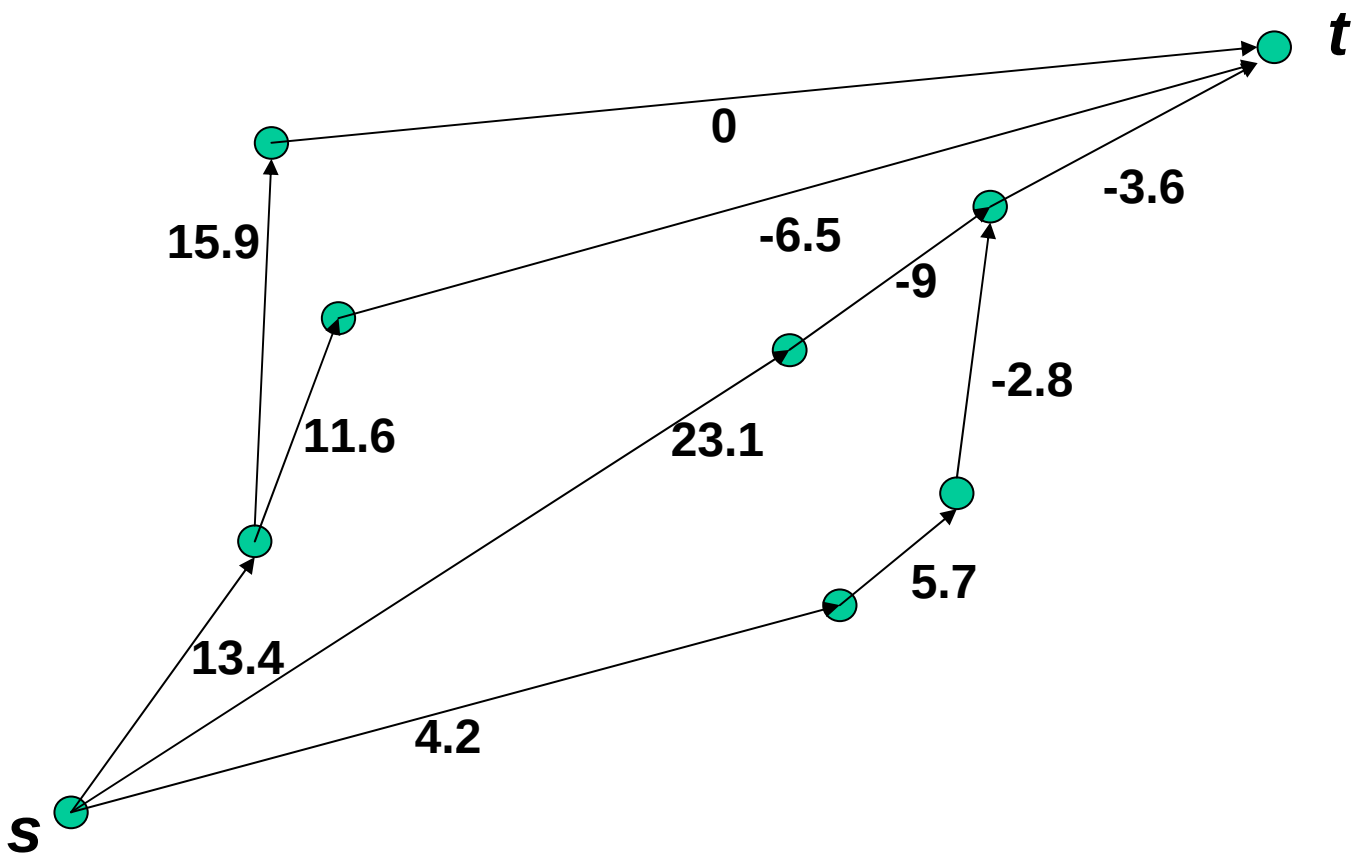
Computation of Weights of Edges

Definition: Weight of an Edge = Sum of weights of points on right of edge - Sum of weights of points on left of edge



Example: Weight of edge (*a, b*) = 13.7 - 2.1 = 11.6

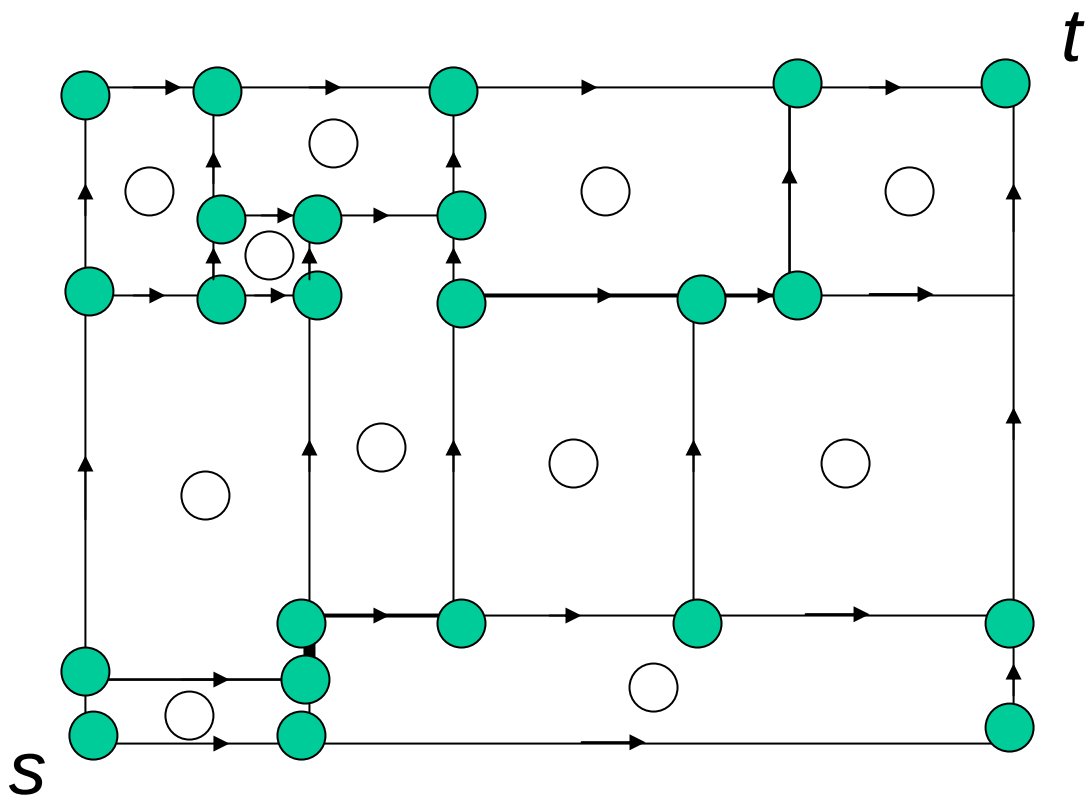
Graph with Unrestricted Edge Costs



Characteristics of the MMG

- Directed
- Monotone Edges
- Acyclic
- Single source and single terminal
- Edges with Real-valued costs
  (positive / zero / negative)

A Floor Plan Graph as a MMG



The Graph-Theoretic Formulation

Input : A directed acyclic graph with positive, zero and negative real edge costs, and with designated source and terminal vertices.

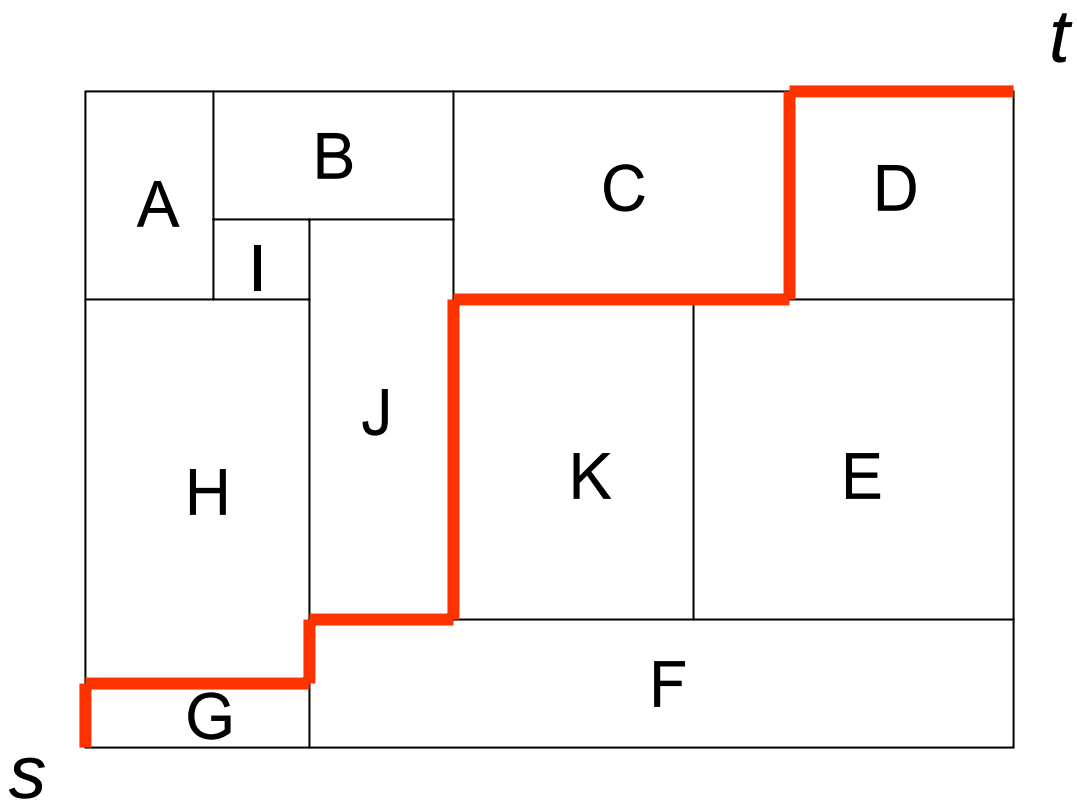
Objective : Find the path having minimum absolute value of the sum of edge costs from the source to the terminal vertex.

Problem is known to be NP-hard

A Proposed Solution

Depth-first graph search with selective pruning of nodes

Monotone Number Bipartition: Preliminaries



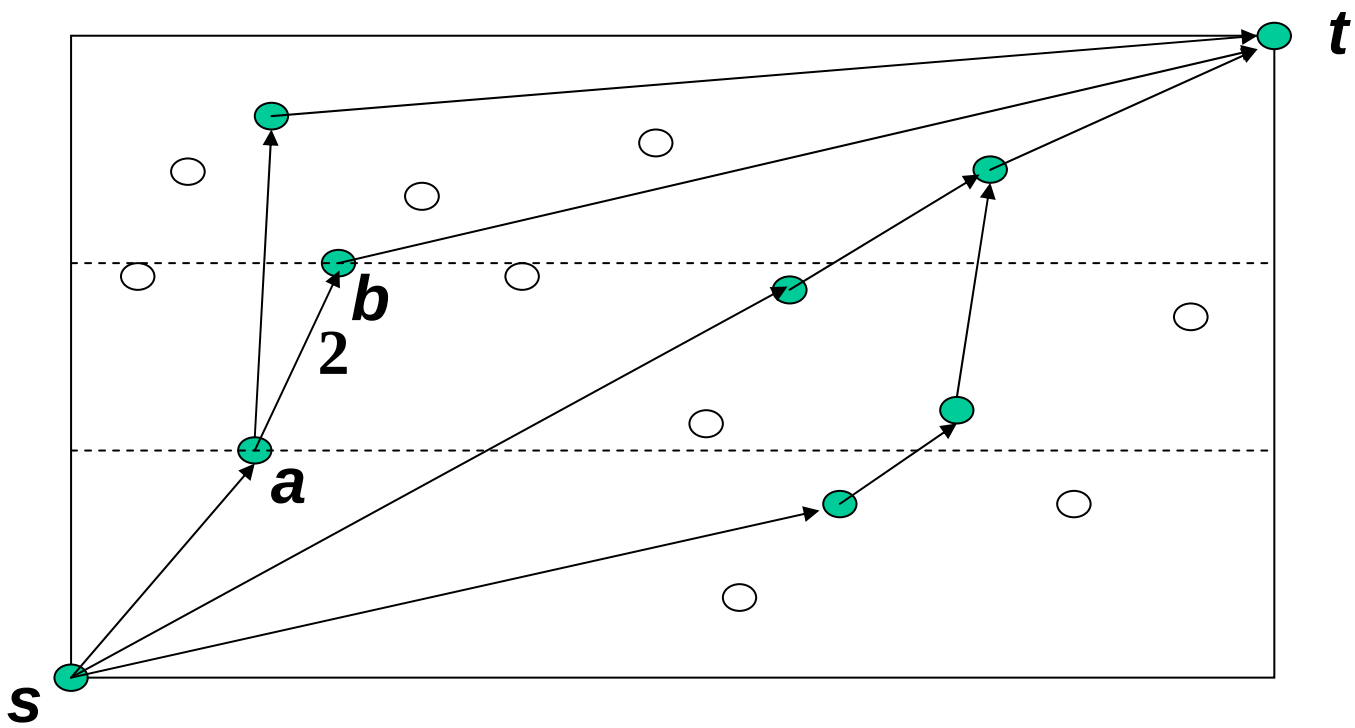
Balance the Number of Blocks on two sides of a Staircase Path

Monotone Number Bipartition : Problem definition

- *Input* : A Placement of Rectangular Blocks with no wasted area.
- *Objective* : To find a Staircase Path through the Blocks from one corner of the floor to its diagonally opposite corner such that the difference of the number of the blocks on its two sides is *minimum.*
- *Constraint*: The staircase path can not pierce through any block.

Computation of Weights of Edges

Definition: Weight of an Edge = Number of points on right of edge - Number of points on left of edge

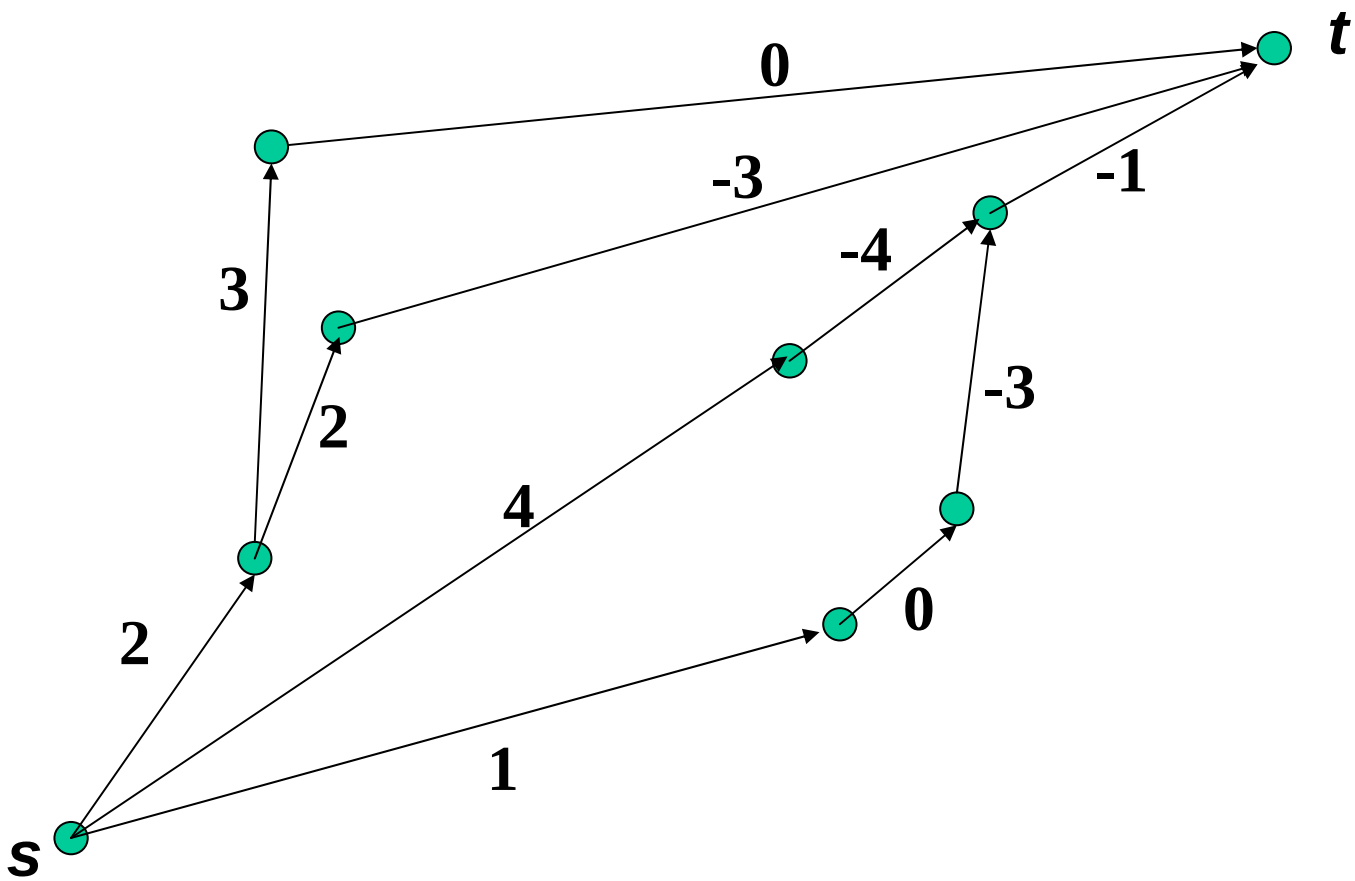


Example: Weight of edge (*a, b*) = 3 - 1 = 2

Characteristics of the Graph

- Directed
- Monotone Edge (i, j) => $x_i \leq x_j$, $y_i \leq y_j$
- Acyclic
- Single-source and single-terminal
- Edges with Integer costs
  (positive / zero / negative)

A Graph with Unrestricted Integer Edge Costs



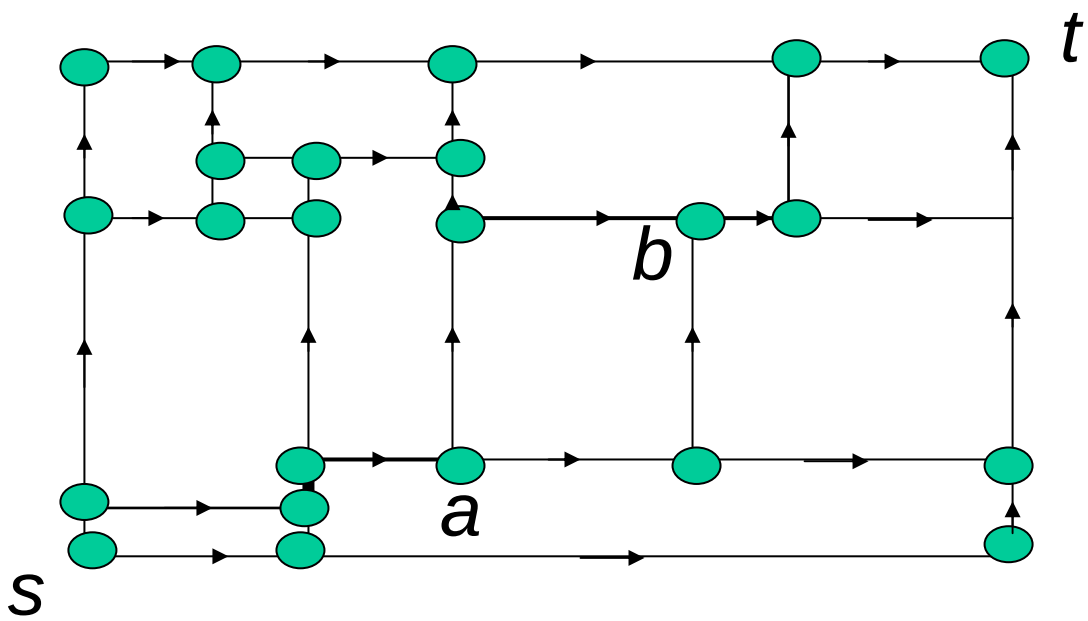
A Graph Problem Formulation

Input : A directed acyclic graph with a single source vertex *s* and a single terminal vertex *t*, and having an integer (positive, zero or negative) cost associated with each edge.

Objective : Find the path having minimum absolute value of the sum of edge costs from the source to the terminal vertex.

A Linear-time algorithm : Definitions

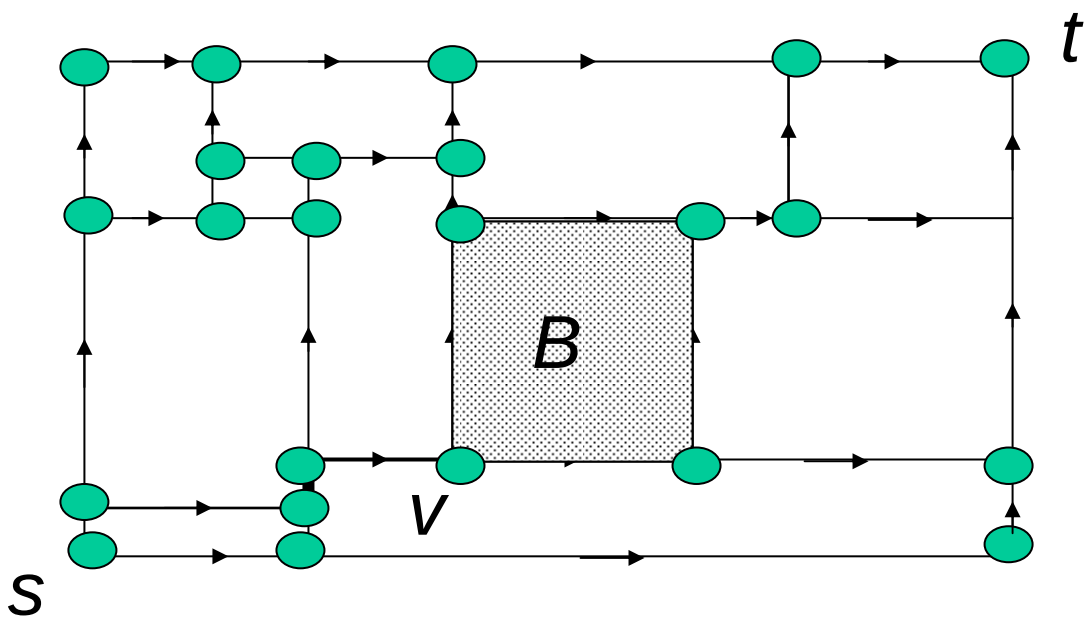
- Vertices of a Floor Plan graph are of *Types* 0, 1 and 2 if their *out-degrees* are respectively 0, 1 and 2.



Vertices *t, b* and *a* are of *Types* 0, 1 and 2 respectively

Linear-time algorithm : Definitions

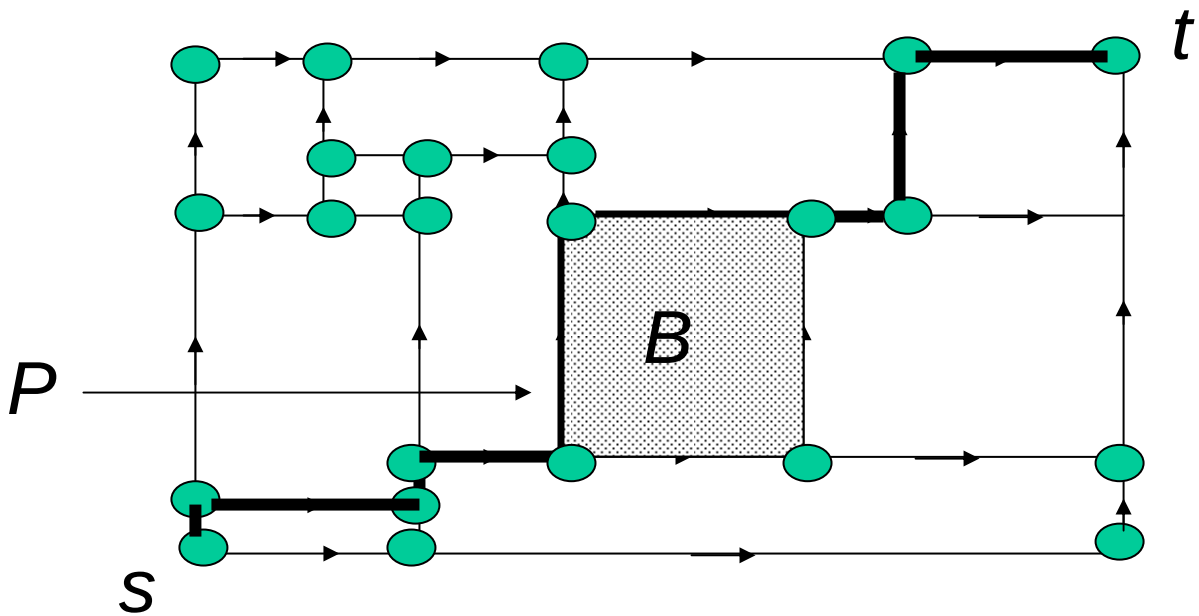
- A block *B* is said to be attached to a vertex *v* of the floor plan graph if the left-bottom corner of *B* coincides with the *T*-junction corresponding to *v*



Block *B* is attached to vertex *v*

Linear-time algorithm : Definitions

- A block B is said to be bounded by a staircase path *P* if the top and left boundaries of *B* are fully spanned by a sequence of edges of *P*



Block *B* is bounded by staircase path *P*

A Critical Fact

It has been shown by us -

Every staircase path through a placement of rectangular blocks must have at least one block bounded by it

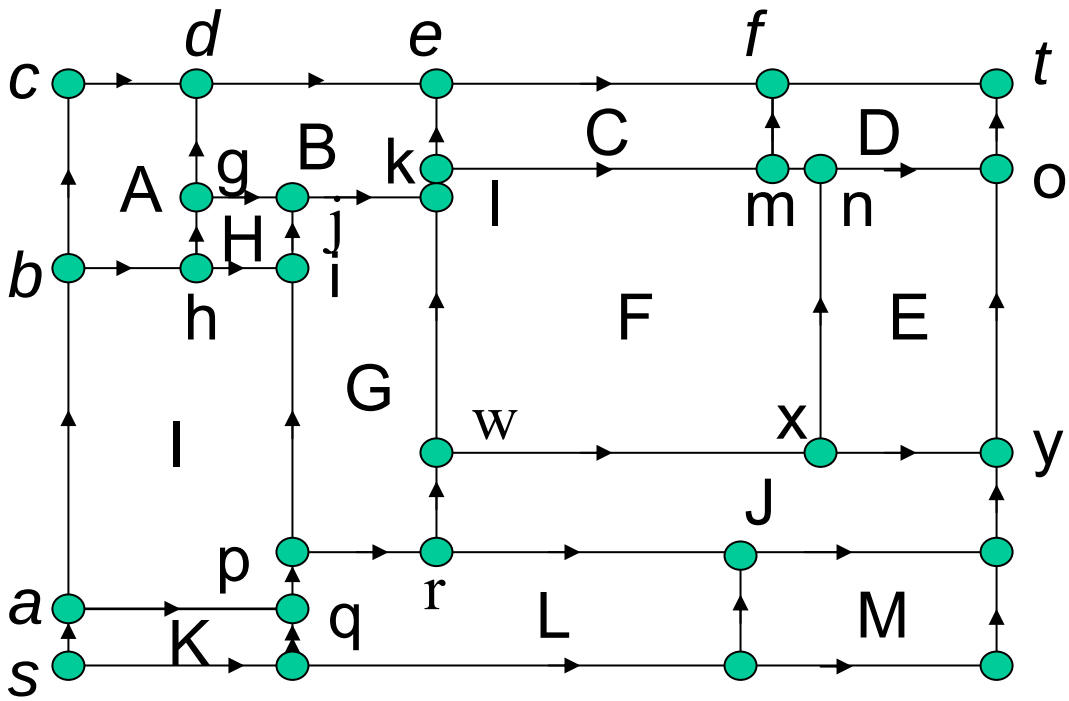
Block Labeling Method

repeat

- start from *s* and terminate at *t*
- initialize label count $\lambda = 1$
- traverse the edges of floor plan graph in a depth-first manner, always traversing left successor before the right successor
- the last block during backtrack is labeled λ
- set $\lambda = \lambda + 1$

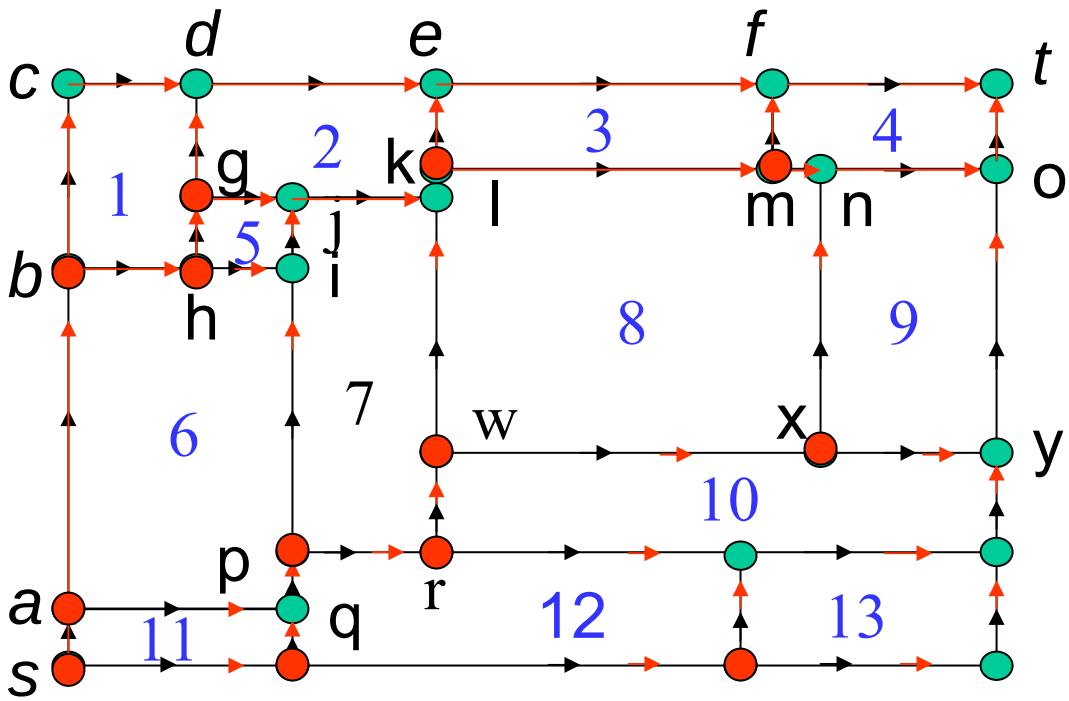
until s is reached during backtrack, and both its successors are already explored

Block Labeling Method : An Example



- start at *s* and move along *a, b, c, d, e, f, t*
- backtrack to type-2 vertex *b*, and label block A by 1
- next move along *h*, *g* up to *d* (already visited type-1 vertex)
- now backtrack from *g* and label *B* as 2
- .....

Block Labeling Method : A Demonstration



Block Labeling Method : Important Results

Result 1. Maximum block label on the left of a staircase path can not exceed the minimum block label on its right.

Result 2. Both the blocks, one with the maximum label on the left of the staircase path and the one with the minimum label on the right of the same path are aligned along the path.

Result 3. Blocks on the left of a staircase path having maximum block label *k* have labels *1, 2, …, k*

Finding Staircase Path with *k* *b*locks on its left

repeat

- start from *s* and terminate at *t*
- initialize label count $\lambda = 1$
- traverse the edges of floor plan graph in a depth-first manner, always traversing left successor before the right successor
- the last block during backtrack is labeled λ
- set $\lambda = \lambda + 1$

until $\lambda > k$

Some Relevant Facts

- Block labeling labels each block *exactly once*, with *distinct* integers
- Time complexity of the Labeling algorithm is $O(n)$
- For a given k $(0 < k < n)$, there exists at least one staircase path with k blocks to its left
- Optimal Bipartition has $k = n/2$;
- Time complexity = $O(n)$

Concluding Remarks

- Partitioning: An important phase in VLSI layout design
- Metric used: Cut size, Area, Number of blocks, Delay, etc.
- Most of the problems are NP-hard; polynomial in some special cases