MODULE- 5

ASIC Construction Floor Planning and Placement and Routing: Physical Design, System Partitioning, Floor planning: tools, I/O and power planning, clock planning, Placement: iterative placement improvement, Physical Design flow, global Routing, Detail Routing, Special Routing, Circuit Extraction and DRC.

by,

Dr.P.Vimala
Department of ECE
Dayananda Sagar College of Engineering
Bangalore

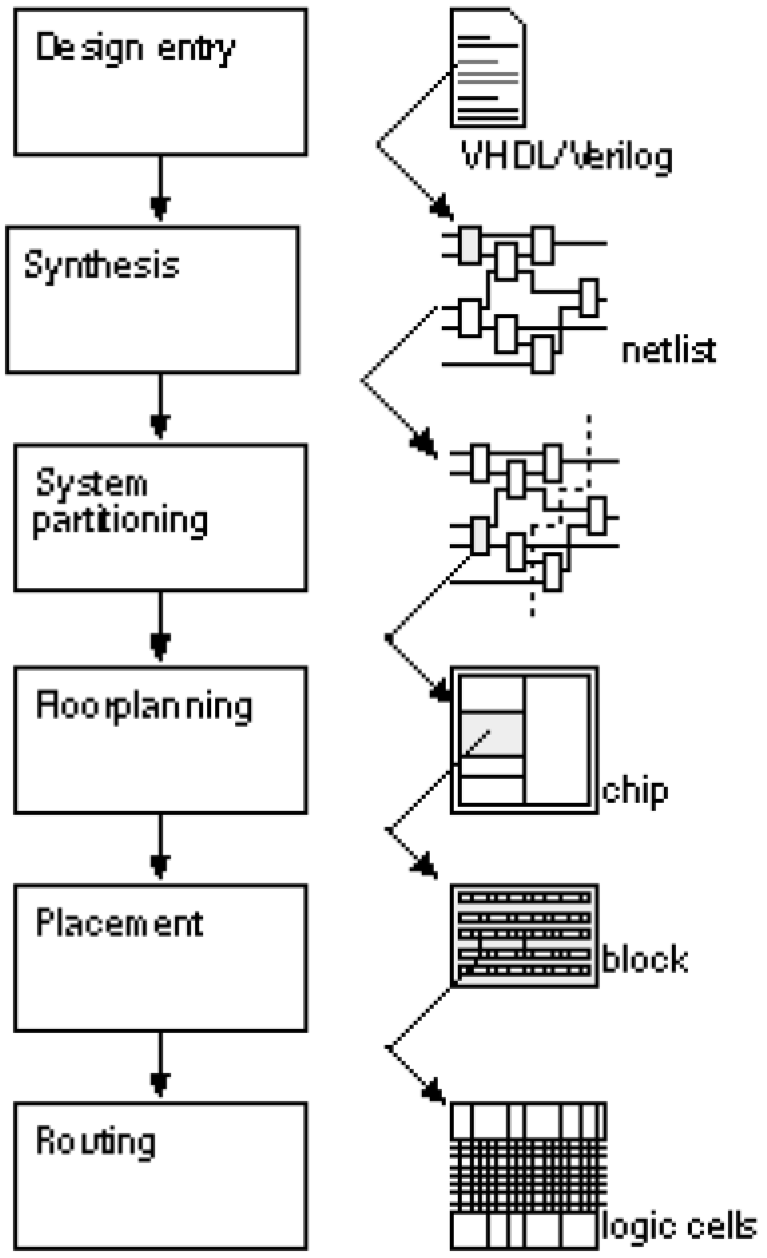
Figures from M.J.S .Smith, - “**Application - Specific Integrated Circuits**”

ASIC Construction

- The physical design of ASICs is normally divided into **system partitioning, floorplanning, placement, and routing.**
- A microelectronic system is the town and the ASICs are the buildings.
  - **System partitioning** corresponds to town planning.
  - **Floorplanning** is the architect’s job.
  - **Placement** is done by the builder.
  - **Routing** is done by the electrician.
- We shall design most, but not all, ASICs using these design steps.

Physical Design

- **System partitioning**: to **divide a microelectronics system into separate ASICs**.
- **Floorplanning: estimate sizes and set the initial relative locations of the various blocks in our ASIC** (sometimes we also call this chip planning).
- At the same time we allocate **space for clock and power wiring and decide on the location of the I/O and power pads.**



Placement:

- **Defines the location of the logic cells** within the flexible blocks and **sets aside space for the interconnect to each logic cell.**
- Placement for a gate-array or standard-cell design **assigns each logic cell to a position in a row.**
- **For an FPGA,** placement **chooses which of the fixed logic resources on the chip** are used for which logic cells.
- **Floorplanning and placement** are **closely related** and are sometimes **combined in a single CAD tool.**

Routing:

- Makes the **connections between logic cells**.
- Routing is a hard problem by itself and is normally split into two distinct steps, called **global and local routing**.
- **Global routing** determines where the interconnections between the placed logic cells and blocks will be situated.
- Only the routes to be used by the **interconnections are decided** in this step, not the actual locations of the interconnections within the wiring areas.
- Global routing is sometimes called **loose routing** for this reason.
- **Local routing** joins the logic cells with interconnections. **Information on which interconnection areas to use comes from the global router.** Only at this stage of layout do we finally decide on the width, mask layer, and exact location of the interconnections. Local routing is also known as detailed routing.

System Partitioning

- Microelectronic systems typically consist of many functional blocks.
- If a functional block is **too large to fit in one ASIC**, we may have **to split, or partition, the function into pieces**
- **Example:** want **to minimize the number of pins for each ASIC to minimize package cost.** We can use **CAD tools** to help us with this type of system partitioning.
- Some of the partitioning of the system is determined by whether to use **ASSPs or custom ASICs.**
- Some of these design decisions are based on intangible issues: **time to market, previous experience with a technology, the ability to reuse part of a design** from a previous product. **No CAD tools can help with such decisions**.
- CAD tools cannot answer a question such as: "What is the cheapest way to build my system?" but can help the designer answer the question: "How do I split this circuit into pieces that will fit on a chip?"

Power Dissipation

Power dissipation in CMOS logic arises from the following sources:

- **dynamic power dissipation**
  - due to **switching current** from charging and discharging parasitic capacitance.
  - due to **short-circuit current** when both n -channel and p -channel transistors are momentarily on at the same time.
- **Static** (leakage current and subthreshold current) **power dissipation**
  - due to **leakage current and subthreshold current**.

Switching Current:

- When the **p -channel transistor** in an inverter is charging a capacitance, C , at a frequency, f , the **current** through the transistor is **C (d V /d t).**
- The power dissipation is thus CV (d V /d t) for one-half the period of the input, t = 1/(2 f).
- The power dissipated in the p -channel transistor is,

$$\int_{0}^{1/2f} CV\frac{dv}{dt}dt = \int_{0}^{V_{DD}} CV\,dv = 0.5CV_{DD}{}^{2}$$

- When the **n -channel transistor discharges the capacitor, the power dissipation is equal.**
- total power dissipation: $P_1 = fCV^2{}_{DD}$
- The **best way to reduce power is to reduce V $_{\mathbf{DD}}$** (because it appears as a squared term), and **to reduce C** , the amount of capacitance we have to switch.

Short-Circuit Current :

- Also known as **crowbar current**
- Can be particularly **important for output drivers and large clock buffers**.
- The power dissipation due to the crowbar current is:

$$P_2 = (1/12)\beta\, f\, t_{rf}(V_{DD} - 2\, V_{tn})$$

- where we **assume the following**:
  - Ratio the p -channel and n -channel **transistor sizes,** β = (W/L)μCox is the **same** for both p - and n -channel transistors
  - **Magnitude of the threshold voltages** V_{tn} are assumed **equal** for both transistor types
  - t_{rf} is the **rise and fall time (assumed equal)** of the input signal
- short-circuit current is typically **less than 20 percent of the switching current**

Subthreshold and Leakage Current

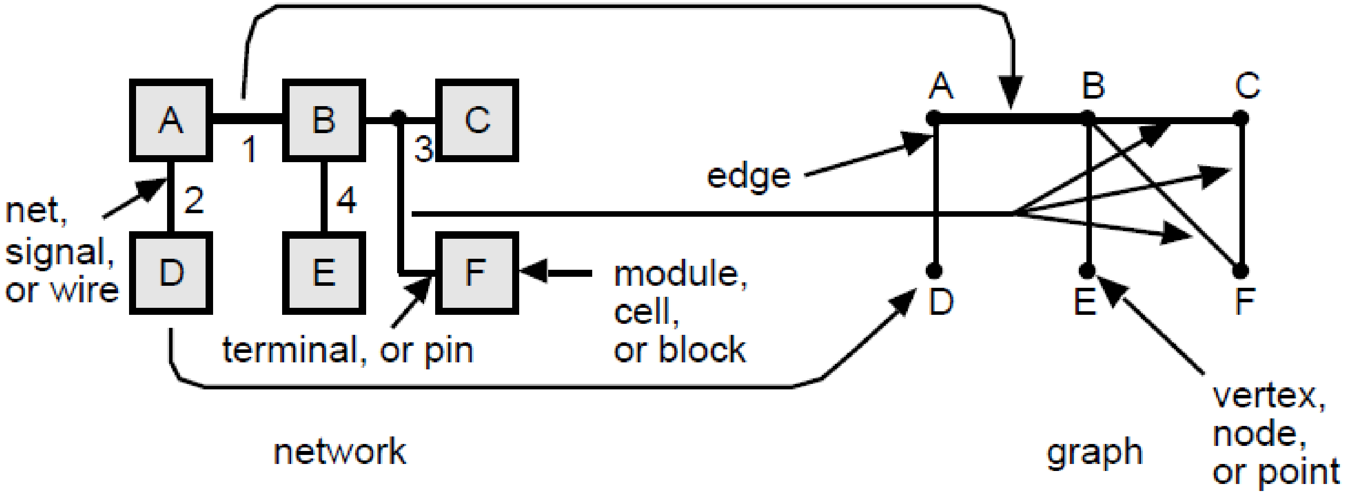
- Subthreshold current:
  - When the gate-to-source voltage, V_{GS} , of an MOS transistor is less than the threshold voltage, V_t , the transistor **conducts a very small subthreshold current** in the subthreshold region$(\boldsymbol{V_{gs} < V_t})$.
  - subthreshold current is normally **less than 5pAμm^{-1} of gate width**
  - subthreshold current for **10 million transistors (each 10μm wide) is 0.1mA**
  - subthreshold current **does not scale**
- Leakage current:
  - Transistor leakage is caused by the fact that a **reverse-biased diode conducts a very small leakage current.**
  - The **sources and drains of every transistor**, as well as **the junctions between the wells and substrate**, form parasitic diodes.

- This is the same order of magnitude (a few microamperes) as the **quiescent leakage current**, I_{DDQ} , that we expect to measure when **we test an ASIC with power applied, but with no signal activity.**
- A measurement of **more current** than this in a **nonactive CMOS ASIC** indicates a **problem with the chip manufacture or the design.**
- We use this measurement to test an ASIC using an **I_{DDQ} test.**

Partitioning Methods

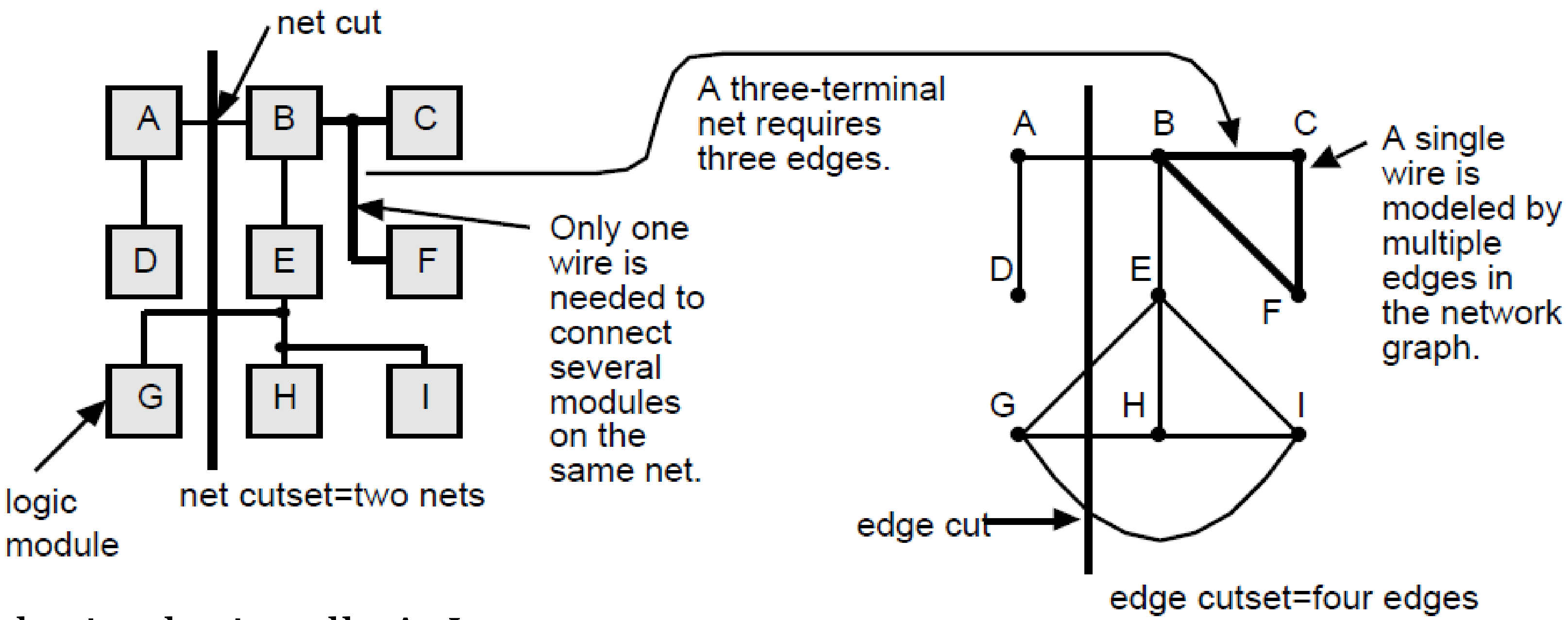
- System partitioning requires **goals and objectives**, **methods and algorithms to find solutions, and ways to evaluate these solutions**.
- Assume that **we have decided which parts of the system will use ASICs**.
- The **goal** of partitioning is **to divide this part of the system** so that each partition is a single ASIC.
- To do this we may need to take into account any or all of the following **objectives**:
  - A maximum size for each ASIC
  - A maximum number of ASICs
  - A maximum number of connections for each ASIC
  - A maximum number of total connections between all ASICs
- explain ways to measure the last two.

Measuring Connectivity



- First diagram shows a **circuit schematic, netlist**, or network.
- The network consists of **circuit modules** A–F.
- Equivalent terms for a **circuit module are a cell, logic cell, macro, or a block.**
  - A **cell or logic cell** usually refers to a **small logic gate** (NAND etc.), but can also be a collection of other cells;
  - **macro** refers to **gate-array cells**;
  - **block** is usually **a collection of gates or cells**.
- use the term logic cell to cover all of these.

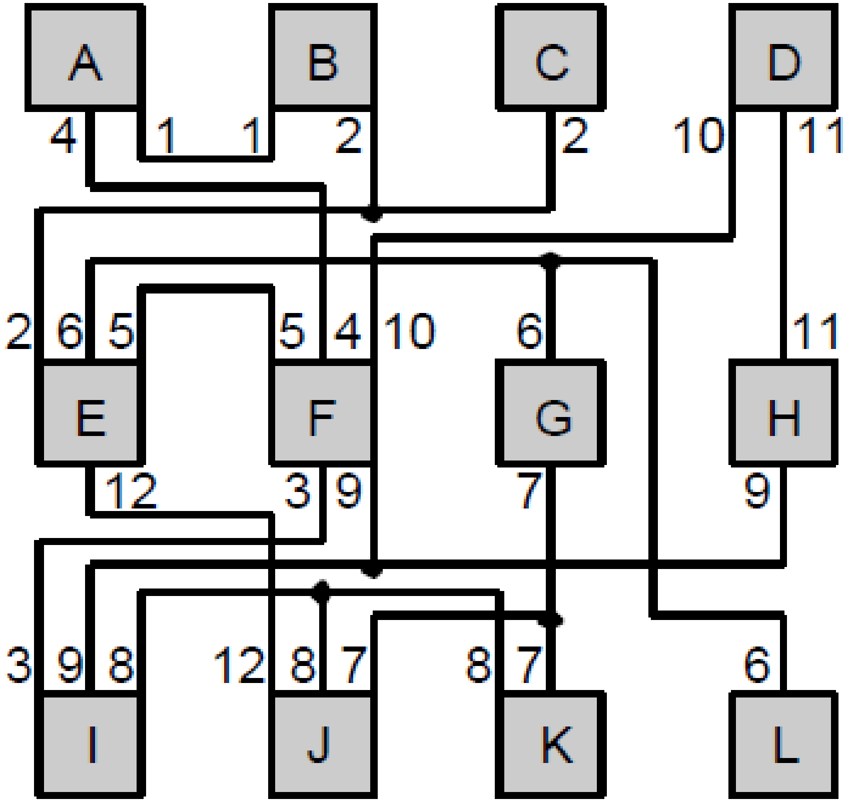
- Each logic cell has **electrical connections between the terminals** (connectors or pins).
- The network can be represented as the **mathematical graph** shown in second diagram.
- A graph is like a spider's web:
  - it contains vertexes (or vertices) A–F (also known as graph nodes or points) that are connected by edges.
  - A **graph vertex** corresponds to a **logic cell.**
  - An **electrical connection** (a net or a signal) between two logic cells corresponds to a **graph edge.**


net cut
A
B
C
D
E
F
G
H
I
logic module
net cutset=two nets
Only one wire is needed to connect several modules on the same net.
A three-terminal net requires three edges.
A
B
C
D
E
F
G
H
I
A single wire is modeled by multiple edges in the network graph.
edge cut
edge cutset=four edges

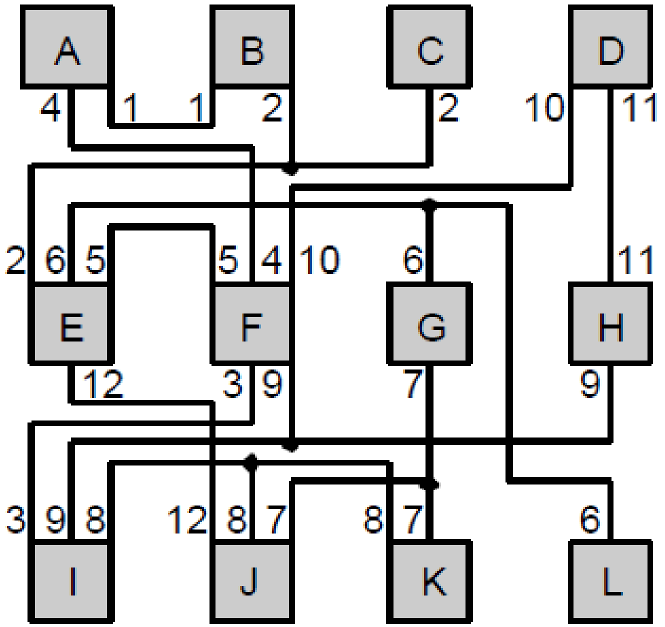
- network with nine logic cells A–I.
- A connection, for example between logic cells A and B, is written as **net (A, B)**. Net (A, B) is represented by the single edge (A, B) in the network graph.
- **A net with three terminals**, for example net (B, C, F), must be modeled with **three edges in the network graph: edges (B, C), (B, F), and (C, F)**. A net with **four terminals** requires six edges and so on.
- **A net can have more than two terminals, but a terminal has only one net.**
- If we divide, or partition, the network into two parts, corresponding to creating two ASICs, we can divide the network's graph in the same way, called a **cutset.**
- We say that there is a **net cutset** (for the network) and an **edge cutset** (for the graph).
- The connections between the two ASICs are **external connections**, the connections inside each ASIC are **internal connections.**

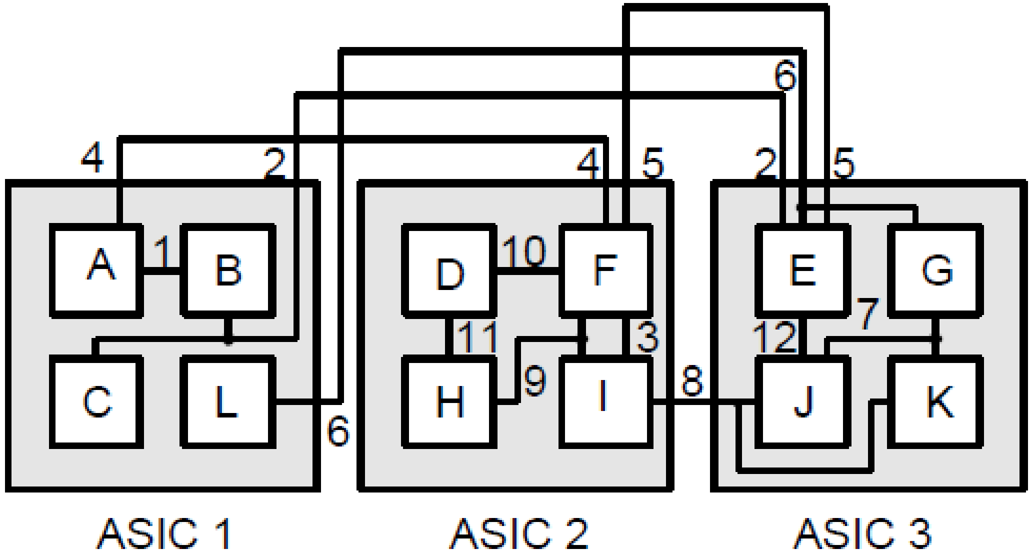
- Notice that the **number of external connections is not modeled correctly by the network graph.**
- When we divide the network into two by drawing a line across connections, we make net cuts. The resulting **set of net cuts** is the net cutset.
- The **number of net cuts** we make corresponds to the **number of external connections between the two partitions**.
- When we divide the network graph into the same partitions we make **set of edge cuts** and we create the edge cutset.
- **nets and graph edges are not equivalent when a net has more than two terminals.**
  - The net cutset contains two nets, but the corresponding edge cutset contains four edges. This means **a graph is not an exact model of a network for partitioning purposes.**
- Thus the **number of edge cuts made when we partition a graph** into two is **not necessarily equal to the number of net cuts** in the network.
- **differences between nets and graph edges is important** when we consider partitioning a network by partitioning its graph

A Simple Partitioning Example



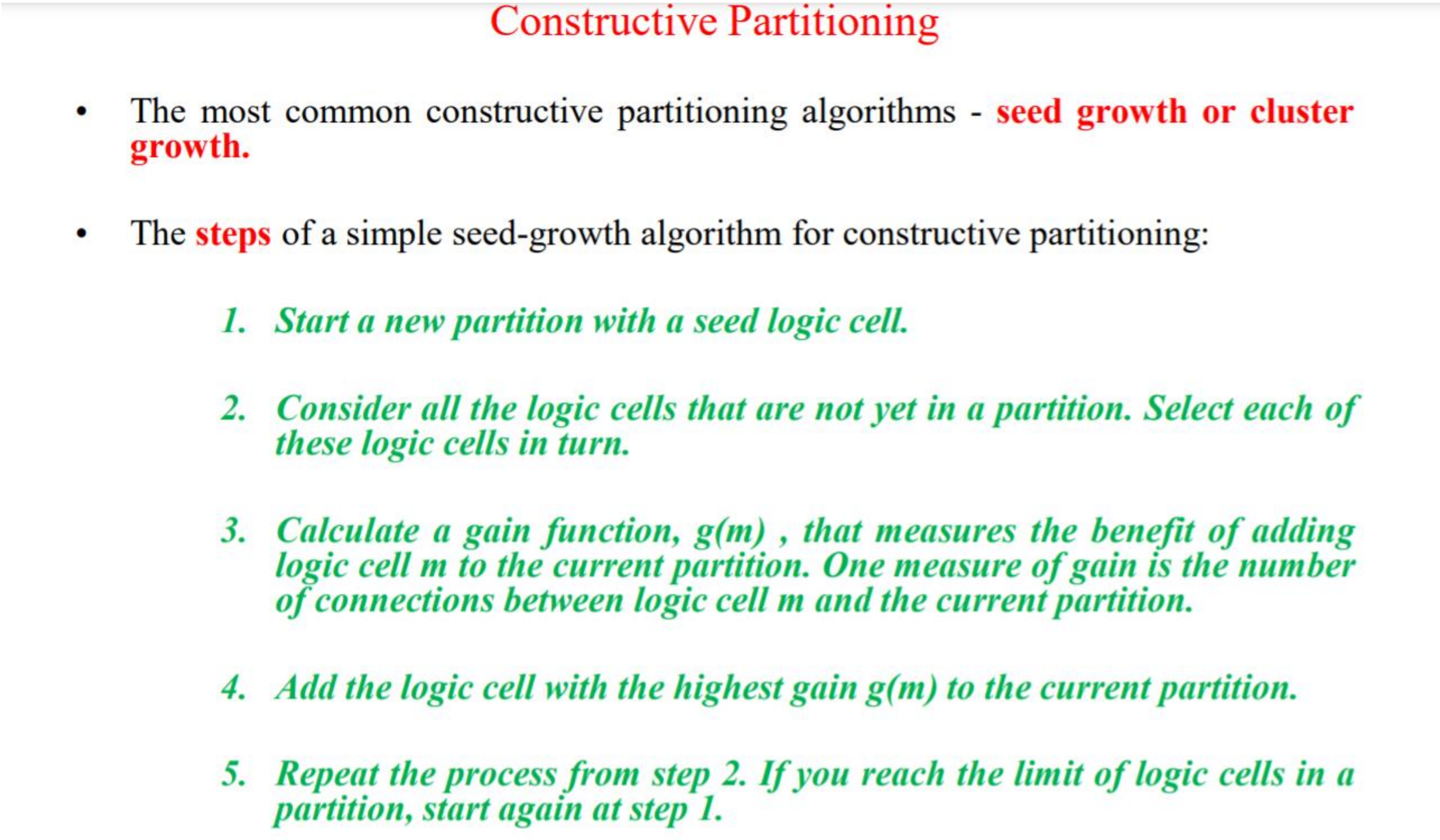
- **12 logic cells**, labeled A–L, **connected by 12 nets** (labeled 1–12)
- each logic cell is a large circuit block and might be RAM, ROM, an ALU, and so on.
- Each net might also be a bus but, we assume that each net is a single connection and all nets are weighted equally.
- The **goal** is **to partition our simple network into ASICs**.
- Our **objectives** are the following:
  i. Use no more than three ASICs.
  ii. Each ASIC is to contain no more than four logic cells.
  iii. Use the minimum number of external connections for each ASIC.
  iv. Use the minimum total number of external connections.





- A partitioning **with five external connections** (nets 2, 4, 5, 6, and 8)—the minimum number.
- two of the ASICs have three pins; the third has four pins.

- Splitting a network into several pieces is a **network partitioning problem.**
- **Two types of algorithms** to solve this problem
  - **constructive partitioning**, which uses a set of rules to find a solution.
  - **iterative partitioning improvement** (or iterative partitioning refinement), which takes an existing solution and tries to improve it.
- Often, **iterative improvement applies to a constructive partitioning.**
- These partitioning algorithms are **uses in solving floorplanning and placement problems.**



Constructive Partitioning

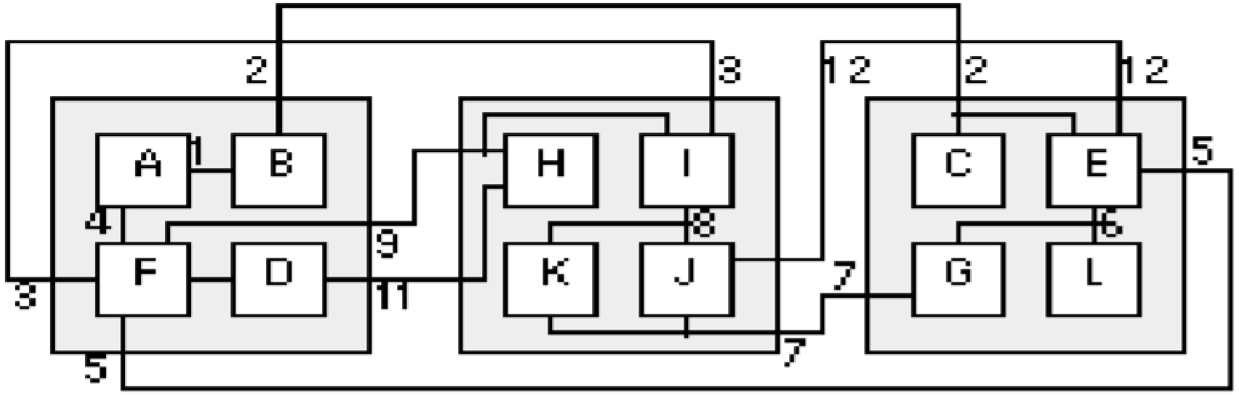
- The most common constructive partitioning algorithms - **seed growth or cluster growth.**

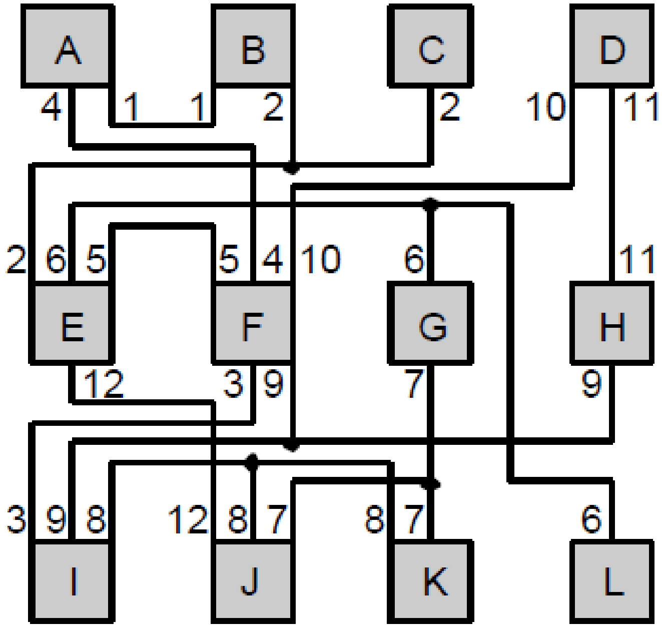
- The **steps** of a simple seed-growth algorithm for constructive partitioning:

  1. ***Start a new partition with a seed logic cell.***
  2. ***Consider all the logic cells that are not yet in a partition. Select each of these logic cells in turn.***
  3. ***Calculate a gain function, g(m) , that measures the benefit of adding logic cell m to the current partition. One measure of gain is the number of connections between logic cell m and the current partition.***
  4. ***Add the logic cell with the highest gain g(m) to the current partition.***
  5. ***Repeat the process from step 2. If you reach the limit of logic cells in a partition, start again at step 1.***

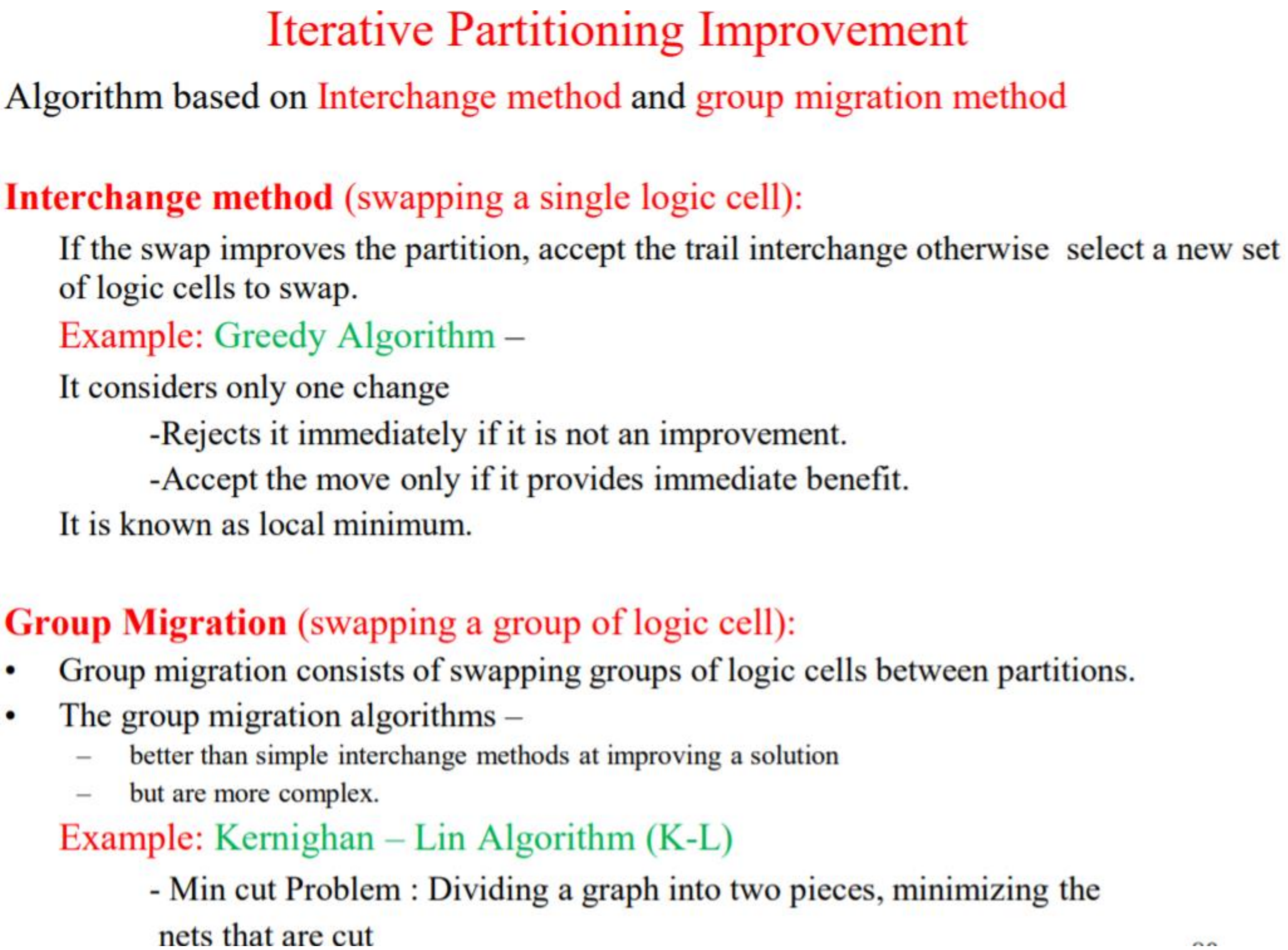
- We may choose **different gain functions according to our objectives** (but we have to be careful to distinguish between connections and nets).
- The **algorithm starts with the choice of a seed logic cell** (seed module, or just seed).
- The logic cell with the **most nets is a good choice** as the seed logic cell.
- You can also use a **set of seed logic cells known as a cluster.**
- Some people also use the term **clique —**borrowed from graph theory.
- A clique of a graph is a subset of nodes where each pair of nodes is connected by an edge.
- In some tools you can use **schematic pages** (at the leaf or lowest hierarchical level) **as a starting point for partitioning**.
- If you use **a high-level design language**, you can use a Verilog module (different from a circuit module) or VHDL entity/architecture as seeds (again at the leaf level).

Constructive Partitioning





•A constructed partition **using logic cell C as a seed**. It is difficult to get from this local minimum, with seven external connections (2, 3, 5, 7, 9,11,12), to the optimum solution of b.



Iterative Partitioning Improvement

Algorithm based on Interchange method and group migration method

Interchange method (swapping a single logic cell):

If the swap improves the partition, accept the trail interchange otherwise select a new set of logic cells to swap.

Example: Greedy Algorithm –

It considers only one change

-Rejects it immediately if it is not an improvement.

-Accept the move only if it provides immediate benefit.

It is known as local minimum.

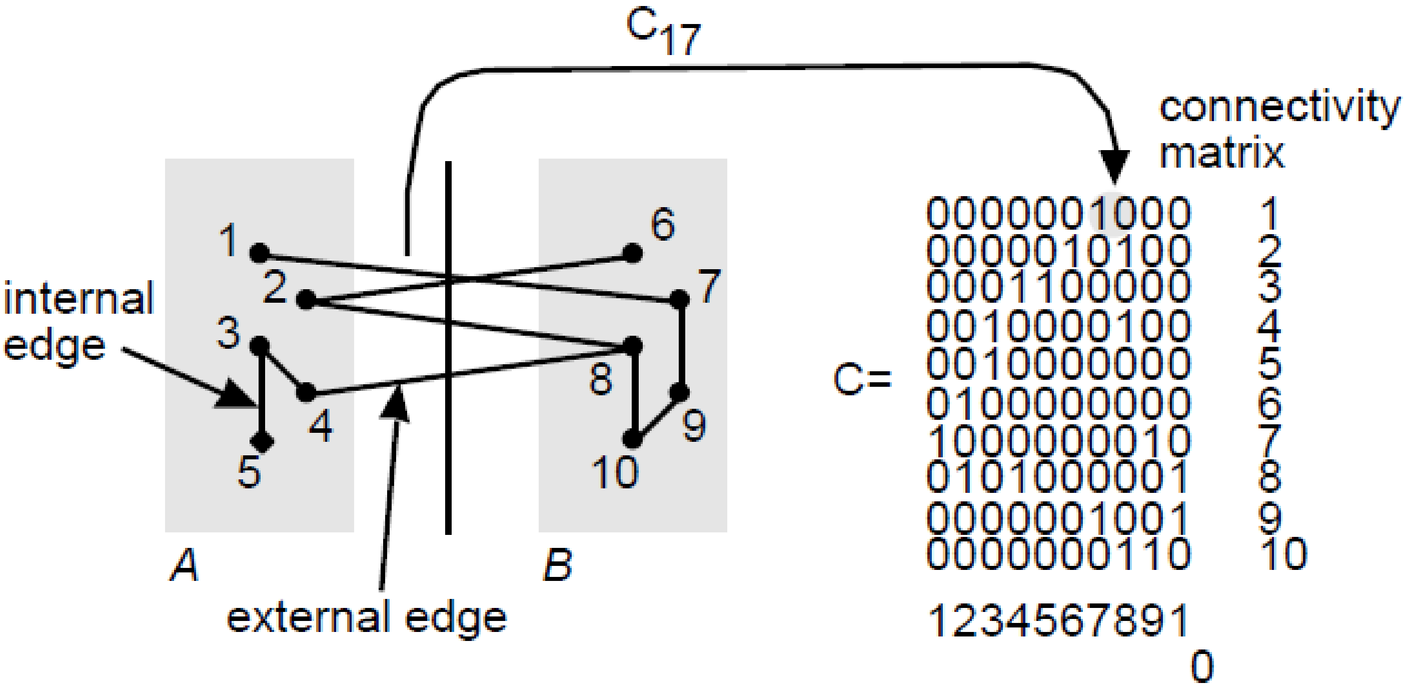
Group Migration (swapping a group of logic cell):

- Group migration consists of swapping groups of logic cells between partitions.
- The group migration algorithms –
  - better than simple interchange methods at improving a solution
  - but are more complex.

Example: Kernighan – Lin Algorithm (K-L)

- Min cut Problem : Dividing a graph into two pieces, minimizing the nets that are cut

The Kernighan–Lin Algorithm



$$C=\begin{array}{cccccccccc|c}
0&0&0&0&0&0&1&0&0&0&1\\
0&0&0&0&0&1&0&1&0&0&2\\
0&0&0&1&1&0&0&0&0&0&3\\
0&0&1&0&0&0&0&1&0&0&4\\
0&0&1&0&0&0&0&0&0&0&5\\
0&1&0&0&0&0&0&0&0&0&6\\
1&0&0&0&0&0&0&0&1&0&7\\
0&1&0&1&0&0&0&0&0&1&8\\
0&0&0&0&0&0&1&0&0&1&9\\
0&0&0&0&0&0&0&1&1&0&10\\
\hline
1&2&3&4&5&6&7&8&9&10&
\end{array}$$

(a) (b)

(a) An example network graph. (b) The connectivity matrix, C; the column and rows are labeled to help you see how the matrix entries correspond to the node numbers in the graph. For example, C_{17} (column 1, row 7) equals 1 because nodes 1 and 7 are connected. In this example all edges have an equal weight of 1, but in general the edges may have different weights.

- **External edges:** cross between partitions;
- **internal edges** are contained inside a partition.
- Consider a network with **2m nodes** (where m is an integer) each of equal size.
- If we assign a cost to each edge of the network graph, we can define a **cost matrix** $\mathbf{C = c_{ij}}$,

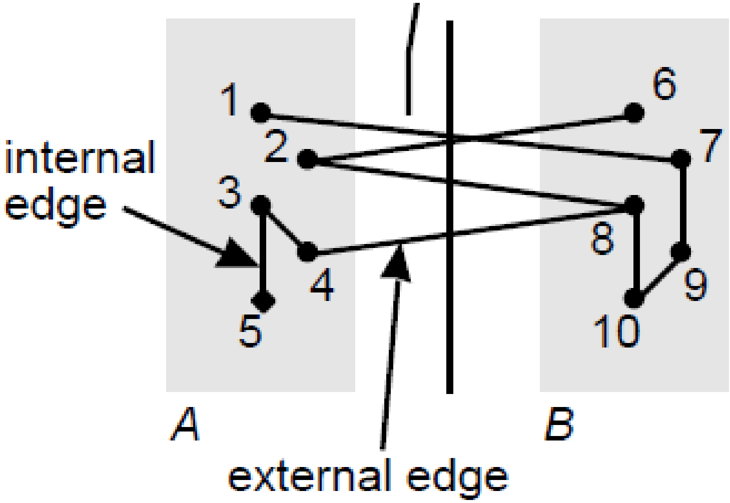
  where $\mathbf{c_{ij} = c_{ji}}$ and $\mathbf{c_{ii} = 0}$**.**
- If all connections are equal in importance, **the elements of the cost matrix are 1 or 0**
- Costs higher than **1 could represent the number of wires in a bus**, multiple connections to a single logic cell, or nets that we need to keep close for timing reasons.

- we already have split a network into two partitions, A and B , each with m nodes (perhaps using a constructed partitioning).
- **Our goal now is to swap nodes between A and B** with the **objective of minimizing the number of external edges connecting the two partitions.**
- **Each external edge may be weighted by a cost**, and **our objective corresponds to minimizing a cost function** that we shall call the total external cost, cut cost, or cut weight, W :

- Total external cost, cut cost, cut weight

$$W = \sum_{a \in A, b \in B} c_{ab}$$

Cut weight= 4 (all the edges have weights of 1)



- External edge cost

$$E_a = \sum_{y \in B} c_{ay}$$

E_1 = 1, and **E_3 = 0**

- Internal edge cost

$$I_a = \sum_{z \in A} c_{az}$$

I_1 = 0, and **I_3 = 2**

- Gain

$$g = D_a + D_b - 2C_{ab}$$

- if we swap nodes 1 and 6, then g = $D_1 + D_6 - 2c_{16} = 1 + 1 - 0$.
- If we swap nodes 2 and 8, then g = $D_2 + D_8 - 2c_{28} = 2 + 1 - 2$.

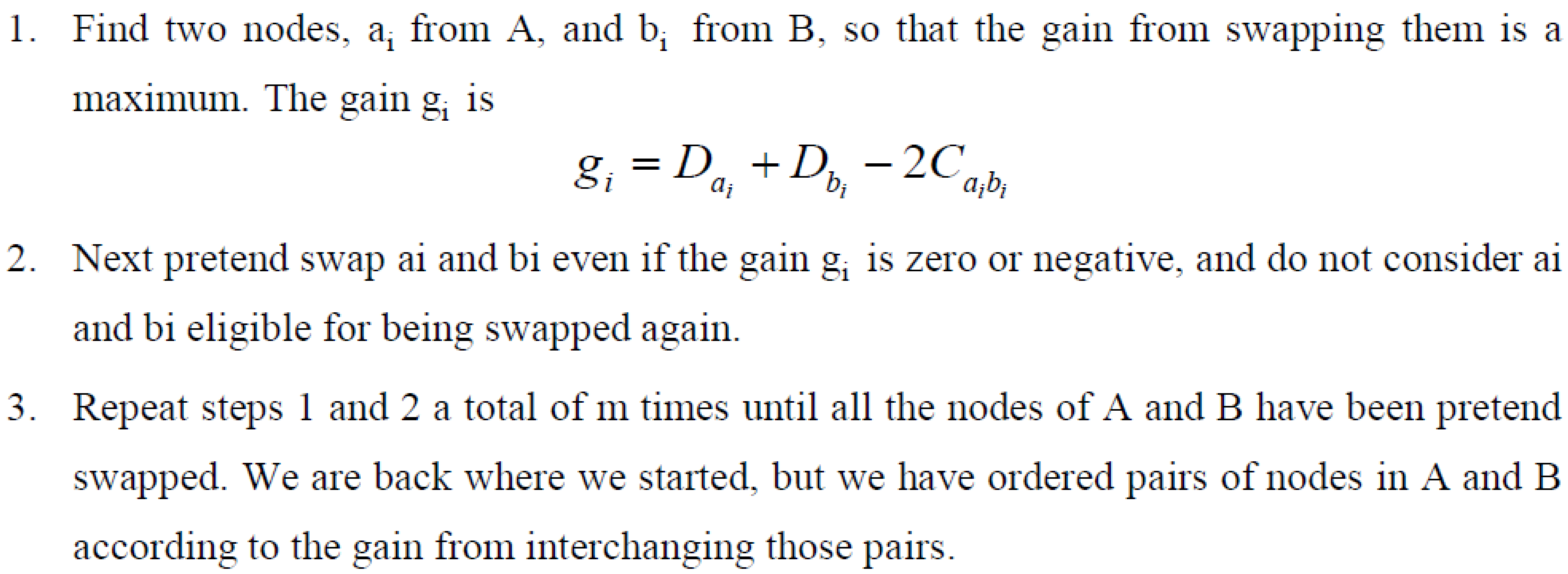
where

$$D_a = E_a - I_a$$

D_1 = 1, D_3 = – 2

- The K–L algorithm **finds a group of node pairs to swap that increases the gain** even though swapping individual node pairs from that group might decrease the gain.
- **First we pretend to swap all of the nodes a pair at a time.**
- **Pretend swaps** are like **studying chess games** when you make a series of trial moves in your head.

The Steps for K–L algorithm:



1. Find two nodes, a_i from A, and b_i from B, so that the gain from swapping them is a maximum. The gain g_i is

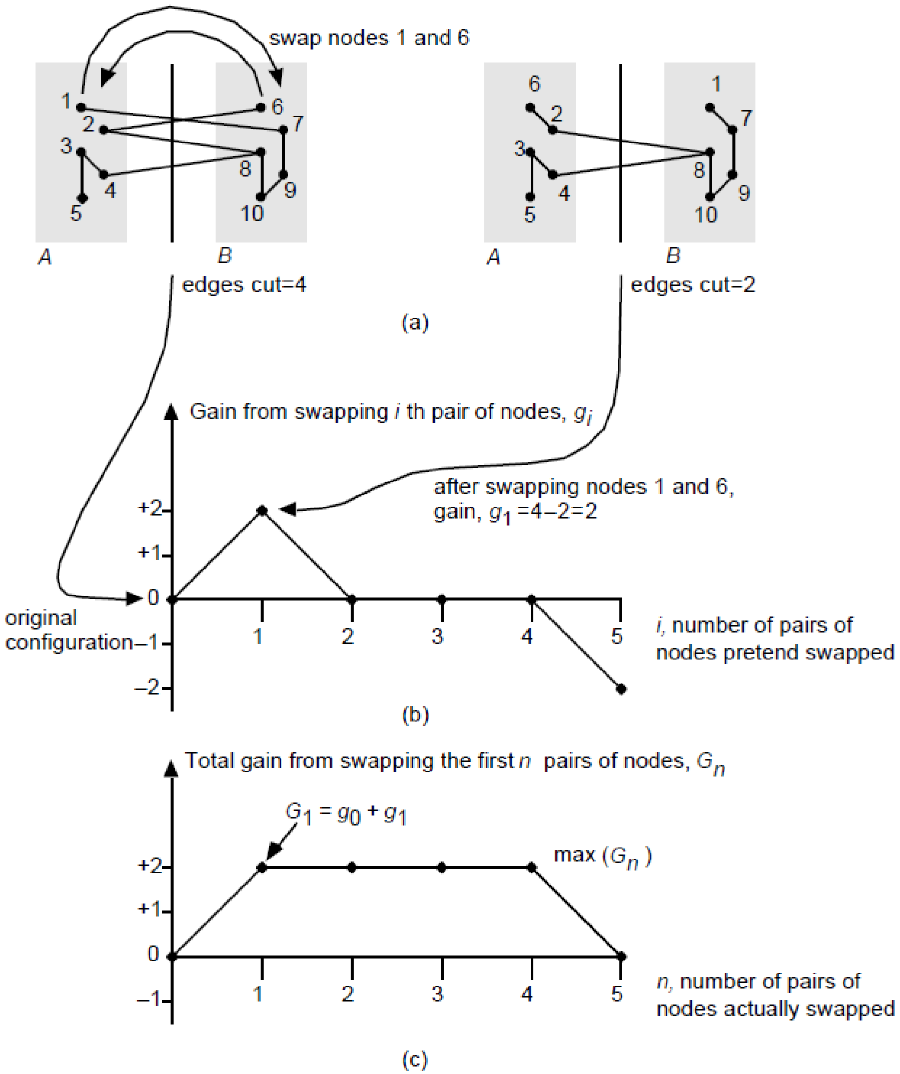
$$g_i = D_{a_i} + D_{b_i} - 2C_{a_i b_i}$$

2. Next pretend swap ai and bi even if the gain g_i is zero or negative, and do not consider ai and bi eligible for being swapped again.
3. Repeat steps 1 and 2 a total of m times until all the nodes of A and B have been pretend swapped. We are back where we started, but we have ordered pairs of nodes in A and B according to the gain from interchanging those pairs.

4. Now we can choose which nodes we shall actually swap. Suppose we only swap the first n pairs of nodes that we found in the preceding process. In other words we swap nodes X = a1, a2, &…., an from A with nodes Y = b1, b2,&…..,bn from B. The total gain would be, $$G_n = \sum_{i=1}^{n} g_i$$

5. We now choose n corresponding to the maximum value of G_n

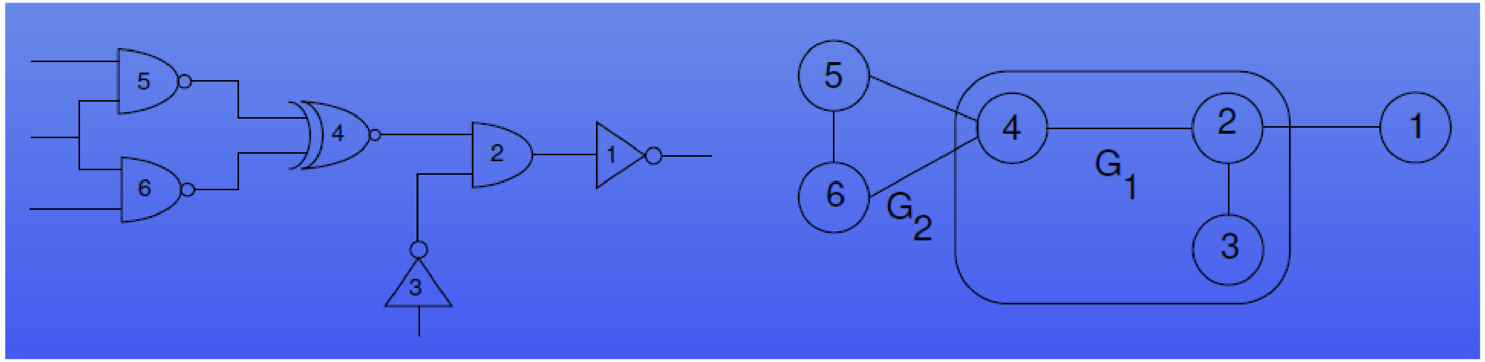
   - If the maximum value of $G_n > 0$, *then swap the sets of nodes X and Y and thus* reduce the cut weight by G_n .
   - *Use this new partitioning to start the process again* at the first step.
   - If the maximum value of $G_n = 0$, *then we cannot improve the current* partitioning and we stop.
   - We have found a locally optimum solution.

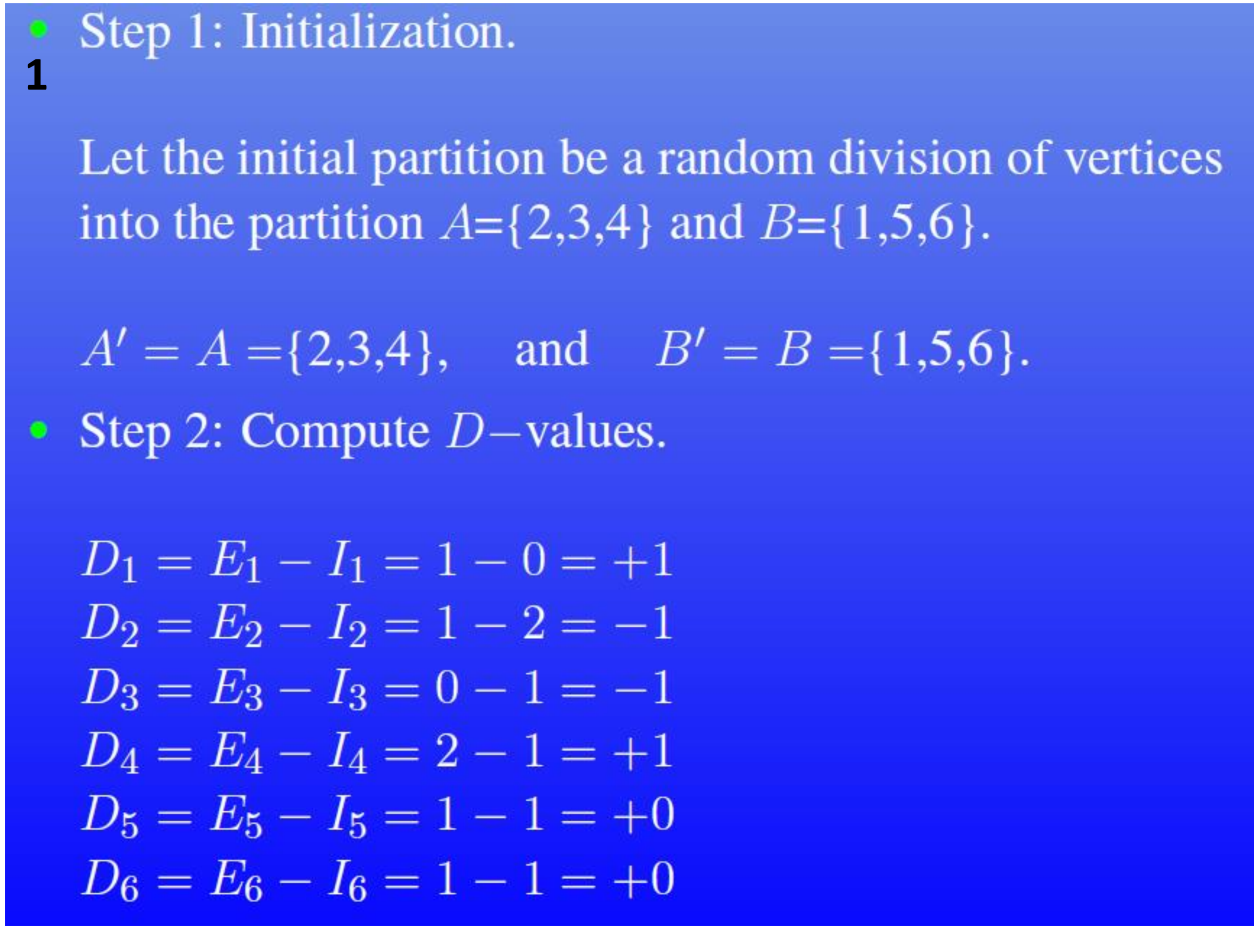

swap nodes 1 and 6
1
2
3
4
5
6
7
8
9
10
A
B
edges cut=4
A
B
edges cut=2
(a)
Gain from swapping *i* th pair of nodes, g_i
after swapping nodes 1 and 6,
gain, g_1 =4–2=2
+2
+1
0
–1
–2
original configuration
1
2
3
4
5
i, number of pairs of nodes pretend swapped
(b)
Total gain from swapping the first *n* pairs of nodes, G_n
$G_1 = g_0 + g_1$
max (G_n)
+2
+1
0
–1
1
2
3
4
5
n, number of pairs of nodes actually swapped
(c)

Partitioning a graph using the Kernighan–Lin algorithm. (a) Shows how swapping node 1 of partition A with node 6 of partition B results in a gain of g = 1. (b) A graph of the gain resulting from swapping pairs of nodes. (c) The total gain is equal to the sum of the gains obtained at each step.

- Each completion of steps 1 through 5 is a pass through the algorithm.
- Kernighan and Lin found that typically **2–4 passes were required to reach a solution.**
- The **most important feature** of the K–L algorithm is that **we are prepared to consider moves even though they seem to make things worse.**
- Sometimes you need to make things worse so **they can get better later.**
- The **K–L algorithm works well** for partitioning graphs.

Example-1





- Step 1: Initialization.

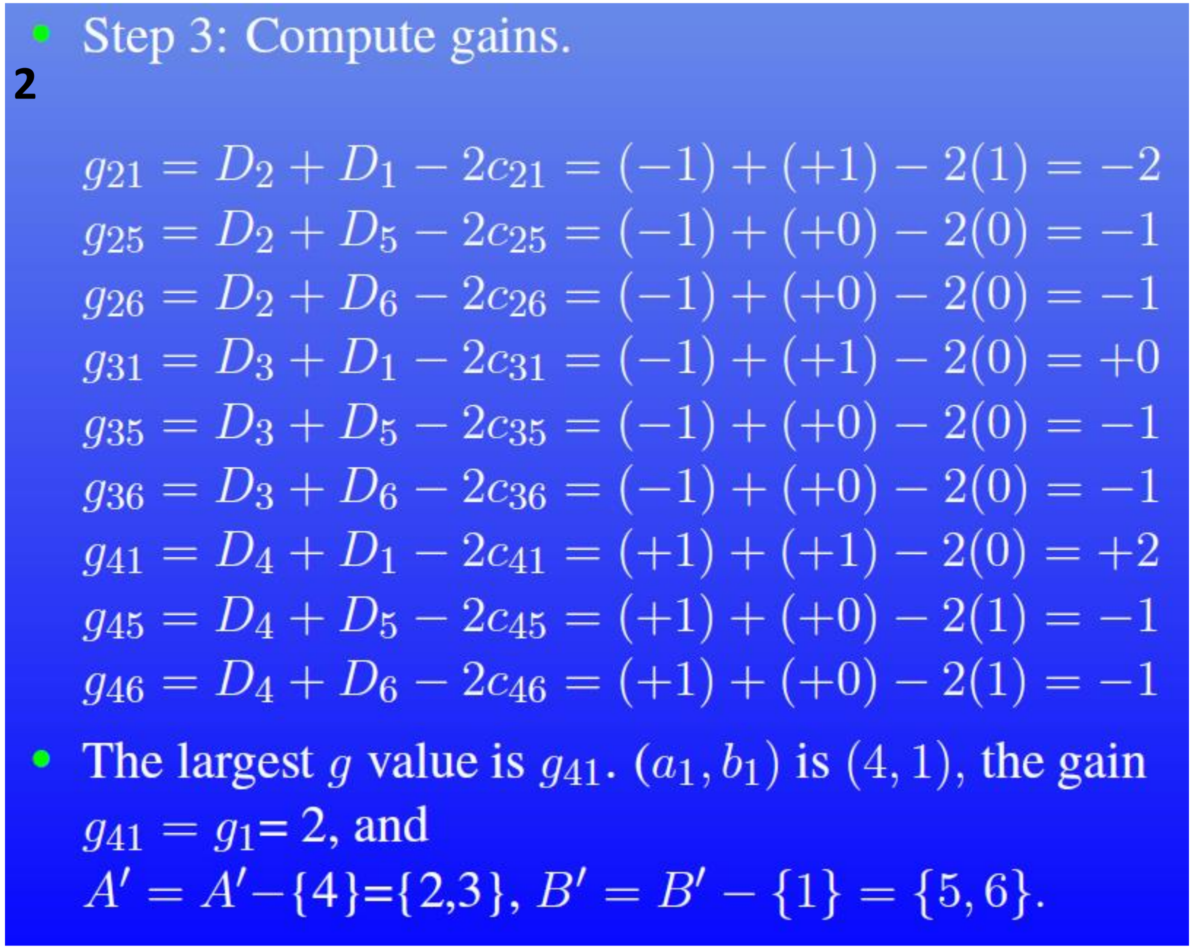
1

Let the initial partition be a random division of vertices into the partition A={2,3,4} and B={1,5,6}.

$A' = A$ ={2,3,4}, and $B' = B$ ={1,5,6}.

- Step 2: Compute $D-$values.

$$D_1 = E_1 - I_1 = 1 - 0 = +1$$
$$D_2 = E_2 - I_2 = 1 - 2 = -1$$
$$D_3 = E_3 - I_3 = 0 - 1 = -1$$
$$D_4 = E_4 - I_4 = 2 - 1 = +1$$
$$D_5 = E_5 - I_5 = 1 - 1 = +0$$
$$D_6 = E_6 - I_6 = 1 - 1 = +0$$



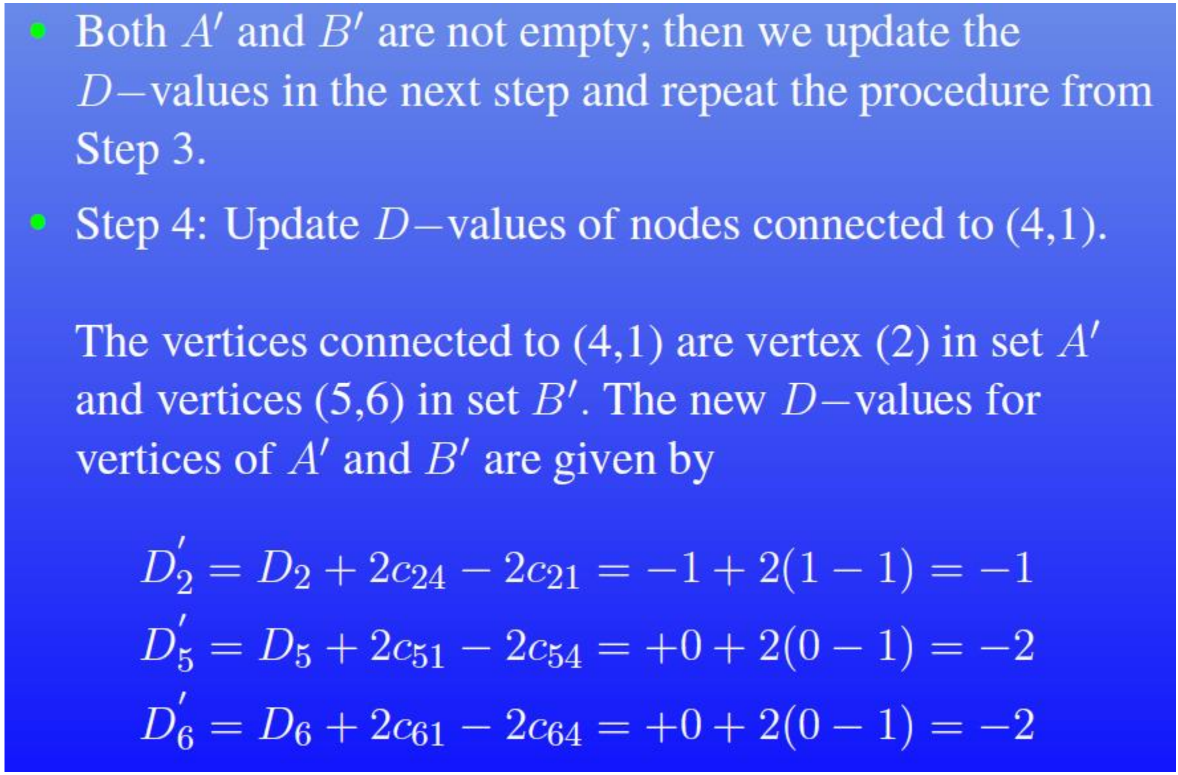
- Step 3: Compute gains.

2

$$g_{21} = D_2 + D_1 - 2c_{21} = (-1) + (+1) - 2(1) = -2$$
$$g_{25} = D_2 + D_5 - 2c_{25} = (-1) + (+0) - 2(0) = -1$$
$$g_{26} = D_2 + D_6 - 2c_{26} = (-1) + (+0) - 2(0) = -1$$
$$g_{31} = D_3 + D_1 - 2c_{31} = (-1) + (+1) - 2(0) = +0$$
$$g_{35} = D_3 + D_5 - 2c_{35} = (-1) + (+0) - 2(0) = -1$$
$$g_{36} = D_3 + D_6 - 2c_{36} = (-1) + (+0) - 2(0) = -1$$
$$g_{41} = D_4 + D_1 - 2c_{41} = (+1) + (+1) - 2(0) = +2$$
$$g_{45} = D_4 + D_5 - 2c_{45} = (+1) + (+0) - 2(1) = -1$$
$$g_{46} = D_4 + D_6 - 2c_{46} = (+1) + (+0) - 2(1) = -1$$

- The largest g value is g_{41}. (a_1, b_1) is $(4, 1)$, the gain $g_{41} = g_1$= 2, and
  $A' = A' - \{4\}$={2,3}, $B' = B' - \{1\} = \{5, 6\}$.

3

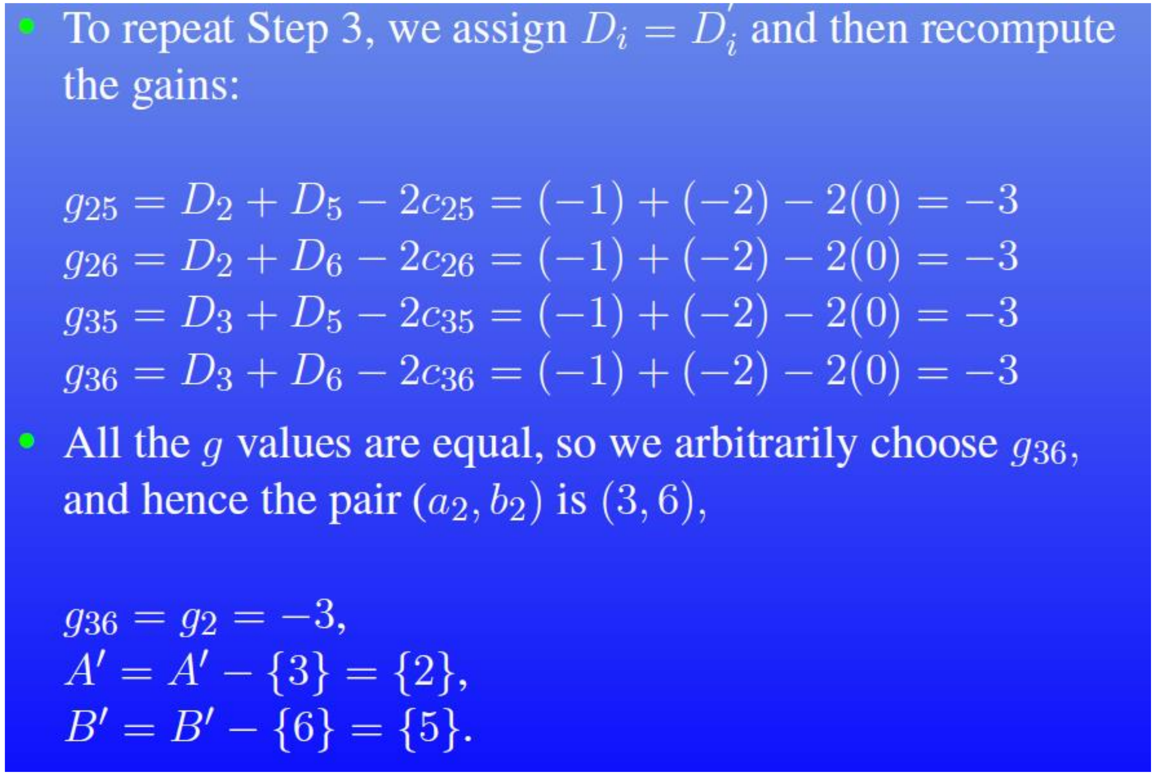


- Both A' and B' are not empty; then we update the $D-$values in the next step and repeat the procedure from Step 3.
- Step 4: Update $D-$values of nodes connected to (4,1).

  The vertices connected to (4,1) are vertex (2) in set A' and vertices (5,6) in set B'. The new $D-$values for vertices of A' and B' are given by

$$
\begin{aligned}
D_2^{'} &= D_2 + 2c_{24} - 2c_{21} = -1 + 2(1-1) = -1 \\
D_5^{'} &= D_5 + 2c_{51} - 2c_{54} = +0 + 2(0-1) = -2 \\
D_6^{'} &= D_6 + 2c_{61} - 2c_{64} = +0 + 2(0-1) = -2
\end{aligned}
$$

4



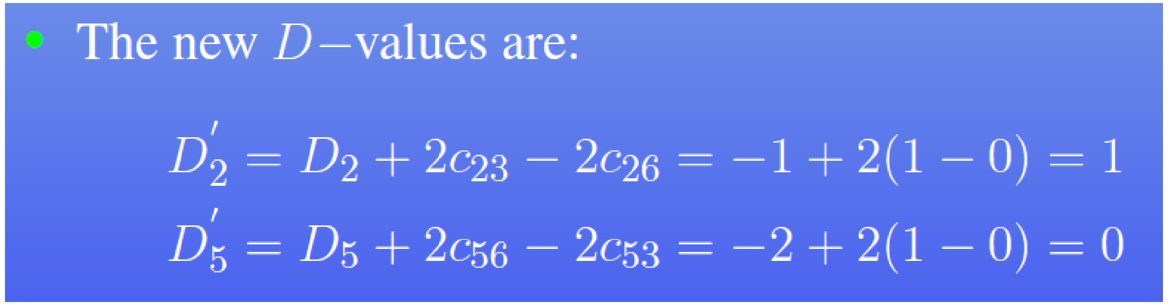
- To repeat Step 3, we assign $D_i = D_i^{'}$ and then recompute the gains:

$$
\begin{aligned}
g_{25} &= D_2 + D_5 - 2c_{25} = (-1) + (-2) - 2(0) = -3 \\
g_{26} &= D_2 + D_6 - 2c_{26} = (-1) + (-2) - 2(0) = -3 \\
g_{35} &= D_3 + D_5 - 2c_{35} = (-1) + (-2) - 2(0) = -3 \\
g_{36} &= D_3 + D_6 - 2c_{36} = (-1) + (-2) - 2(0) = -3
\end{aligned}
$$

- All the g values are equal, so we arbitrarily choose g_{36}, and hence the pair (a_2, b_2) is $(3, 6)$,

$$
\begin{aligned}
&g_{36} = g_2 = -3, \\
&A' = A' - \{3\} = \{2\}, \\
&B' = B' - \{6\} = \{5\}.
\end{aligned}
$$

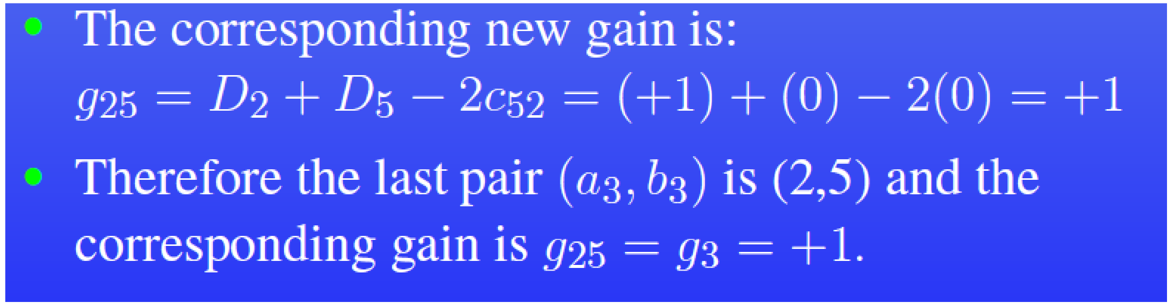
5



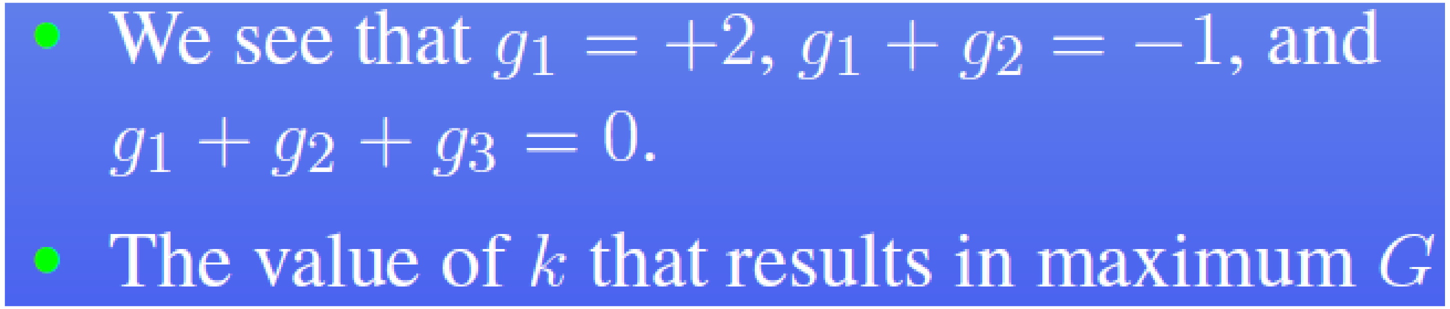
- The new $D-$values are:

$$
\begin{aligned}
D_2^{'} &= D_2 + 2c_{23} - 2c_{26} = -1 + 2(1-0) = 1 \\
D_5^{'} &= D_5 + 2c_{56} - 2c_{53} = -2 + 2(1-0) = 0
\end{aligned}
$$

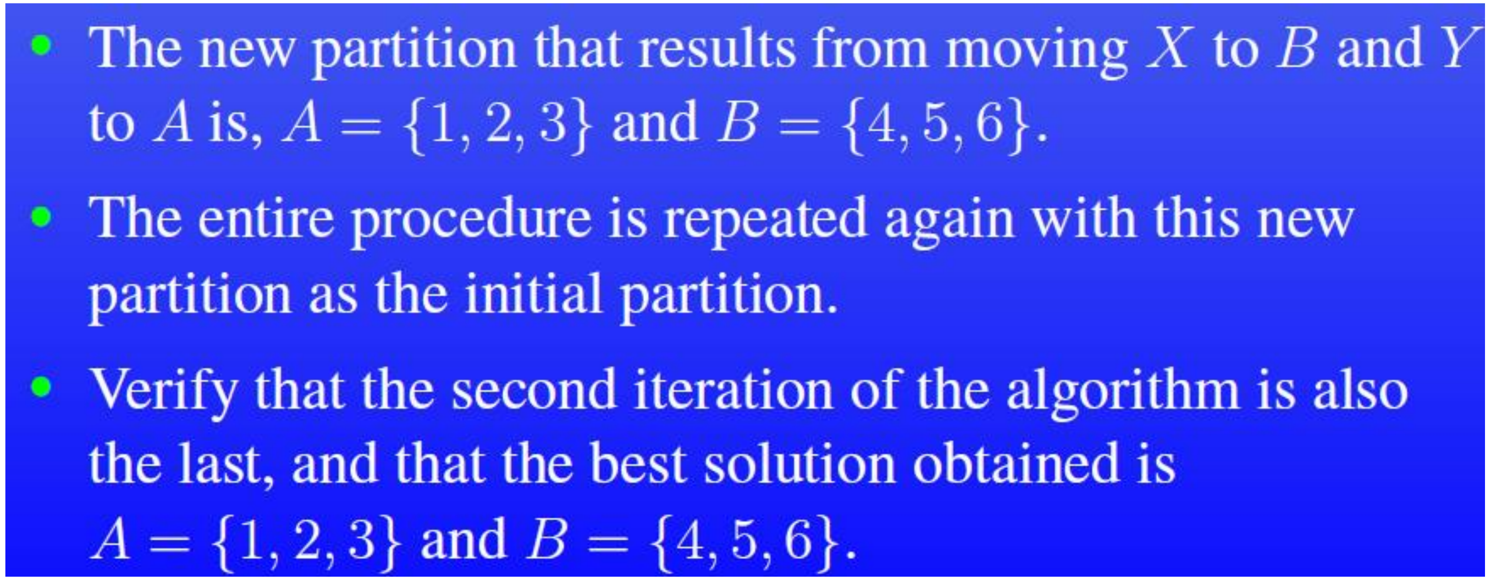
6



- The corresponding new gain is:
  $g_{25} = D_2 + D_5 - 2c_{52} = (+1) + (0) - 2(0) = +1$
- Therefore the last pair (a_3, b_3) is (2,5) and the corresponding gain is $g_{25} = g_3 = +1$.



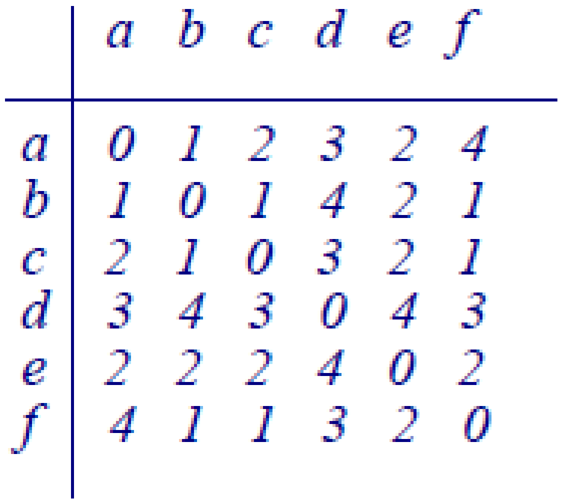
- We see that $g_1 = +2$, $g_1 + g_2 = -1$, and $g_1 + g_2 + g_3 = 0$.
- The value of k that results in maximum G



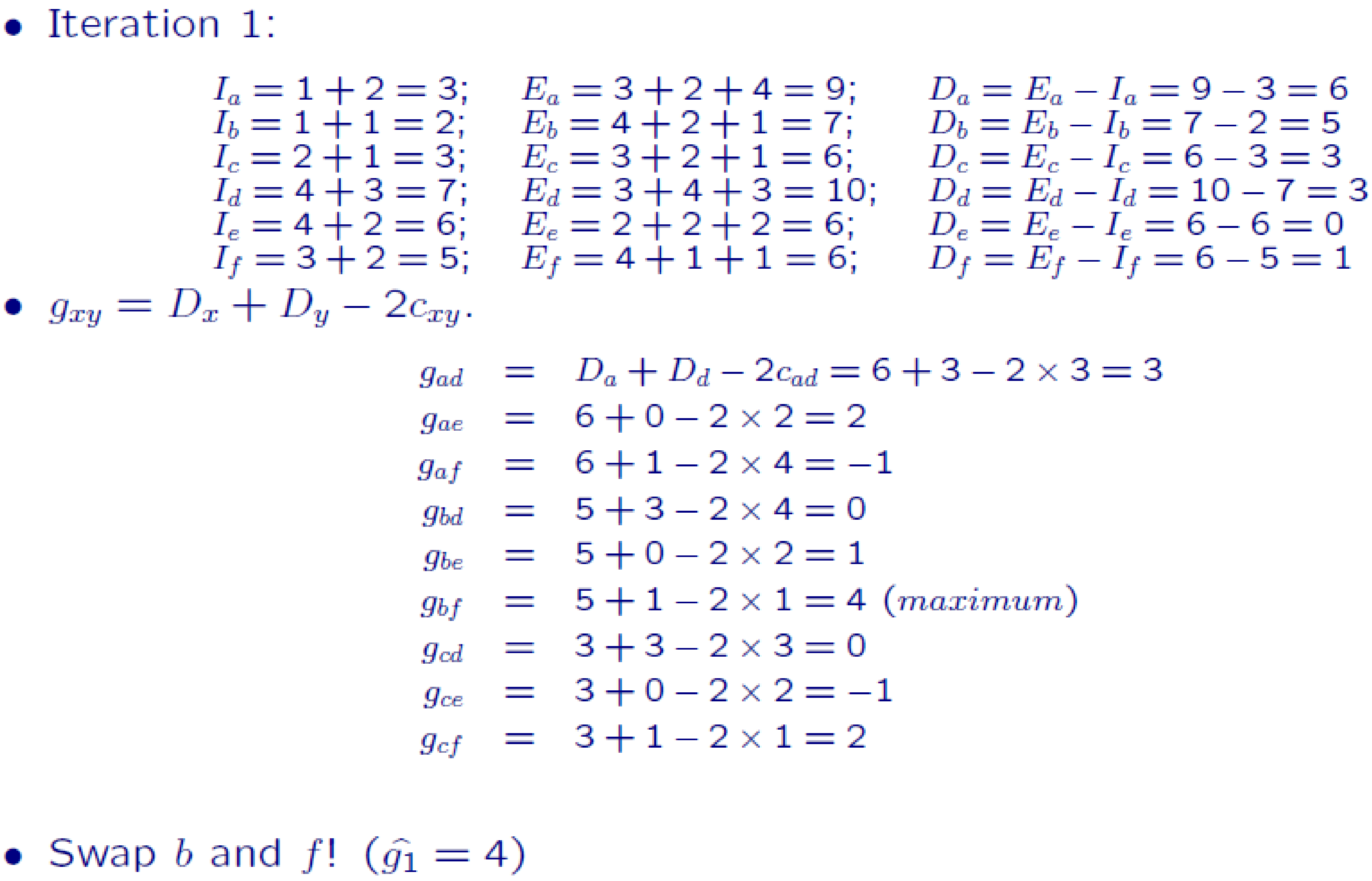
- The new partition that results from moving X to B and Y to A is, $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$.
- The entire procedure is repeated again with this new partition as the initial partition.
- Verify that the second iteration of the algorithm is also the last, and that the best solution obtained is $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$.

Example-2:

a,b,c are in 1st group
d,e,f are in 2nd group



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



- Iteration 1:

$$
\begin{array}{lll}
I_a = 1+2 = 3; & E_a = 3+2+4 = 9; & D_a = E_a - I_a = 9-3 = 6 \\
I_b = 1+1 = 2; & E_b = 4+2+1 = 7; & D_b = E_b - I_b = 7-2 = 5 \\
I_c = 2+1 = 3; & E_c = 3+2+1 = 6; & D_c = E_c - I_c = 6-3 = 3 \\
I_d = 4+3 = 7; & E_d = 3+4+3 = 10; & D_d = E_d - I_d = 10-7 = 3 \\
I_e = 4+2 = 6; & E_e = 2+2+2 = 6; & D_e = E_e - I_e = 6-6 = 0 \\
I_f = 3+2 = 5; & E_f = 4+1+1 = 6; & D_f = E_f - I_f = 6-5 = 1
\end{array}
$$

- $g_{xy} = D_x + D_y - 2c_{xy}$.

$$
\begin{aligned}
g_{ad} &= D_a + D_d - 2c_{ad} = 6+3-2\times 3 = 3 \\
g_{ae} &= 6+0-2\times 2 = 2 \\
g_{af} &= 6+1-2\times 4 = -1 \\
g_{bd} &= 5+3-2\times 4 = 0 \\
g_{be} &= 5+0-2\times 2 = 1 \\
g_{bf} &= 5+1-2\times 1 = 4 \ (maximum) \\
g_{cd} &= 3+3-2\times 3 = 0 \\
g_{ce} &= 3+0-2\times 2 = -1 \\
g_{cf} &= 3+1-2\times 1 = 2
\end{aligned}
$$

- Swap b and f! $(\hat{g_1} = 4)$

- $D'_x = D_x + 2c_{xp} - 2c_{xq}, \forall x \in A - \{p\}$ (swap p and q, $p \in A$, $q \in B$)

$$\begin{aligned} D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\ D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \\ D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\ D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0 \end{aligned}$$

- $g_{xy} = D'_x + D'_y - 2c_{xy}$.

$$\begin{aligned} g_{ad} &= D'_a + D'_d - 2c_{ad} = 0 + 1 - 2 \times 3 = -5 \\ g_{ae} &= D'_a + D'_e - 2c_{ae} = 0 + 0 - 2 \times 2 = -4 \\ g_{cd} &= D'_c + D'_d - 2c_{cd} = 3 + 1 - 2 \times 3 = -2 \\ g_{ce} &= D'_c + D'_e - 2c_{ce} = 3 + 0 - 2 \times 2 = -1 \ (maximum) \end{aligned}$$

- Swap c and e! ($\hat{g_2} = -1$)

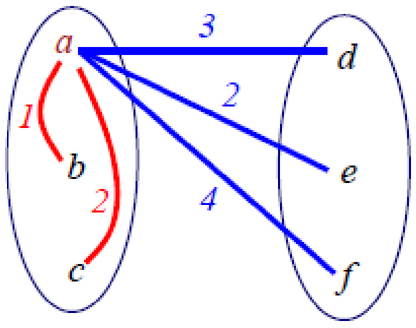
- $D''_x = D'_x + 2c_{xp} - 2c_{xq}, \forall x \in A - \{p\}$

$$\begin{aligned} D''_a &= D'_a + 2c_{ac} - 2c_{ae} = 0 + 2 \times 2 - 2 \times 2 = 0 \\ D''_d &= D'_d + 2c_{de} - 2c_{dc} = 1 + 2 \times 4 - 2 \times 3 = 3 \end{aligned}$$

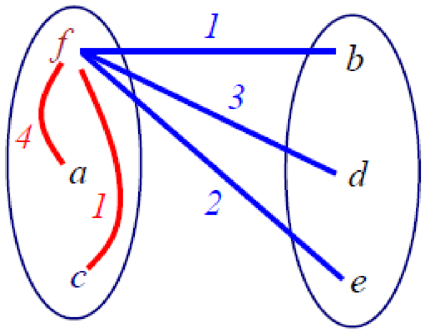
- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3(\hat{g_3} = -3)$$

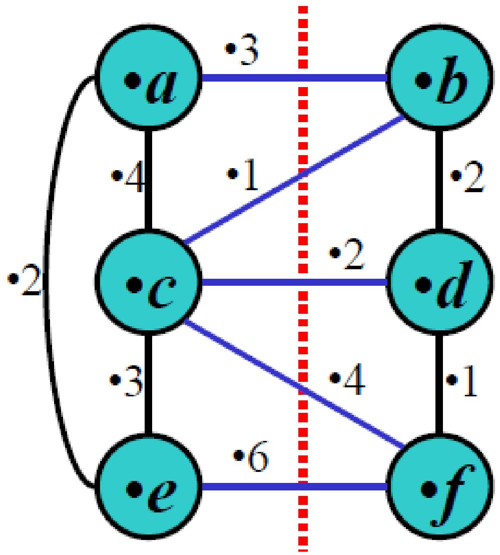
- Note that this step is redundant ($\sum_{i=1}^{n} \hat{g}_i = 0$).
- Summary: $\hat{g_1} = g_{bf} = 4$, $\hat{g_2} = g_{ce} = -1$, $\hat{g_3} = g_{ad} = -3$.
- Largest partial sum $\max \sum_{i=1}^{k} \hat{g}_i = 4$ $(k = 1) \Rightarrow$ Swap b and f.

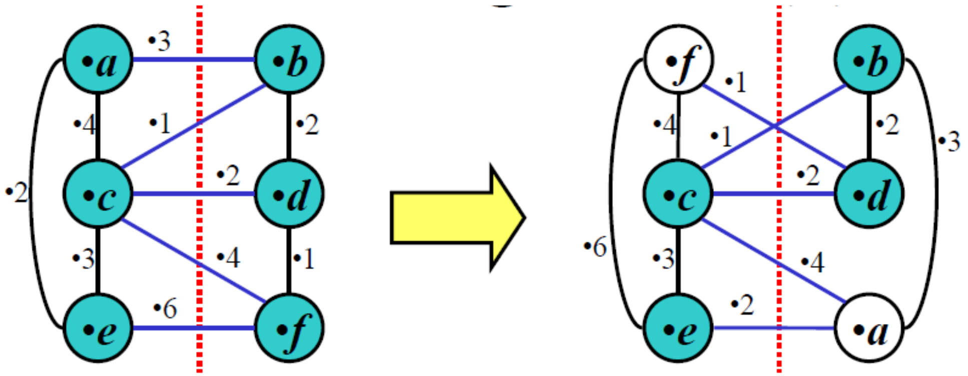


costs associated with a



Example:3





•cut-size = 16

•cut-size = 16 – 6 = 10

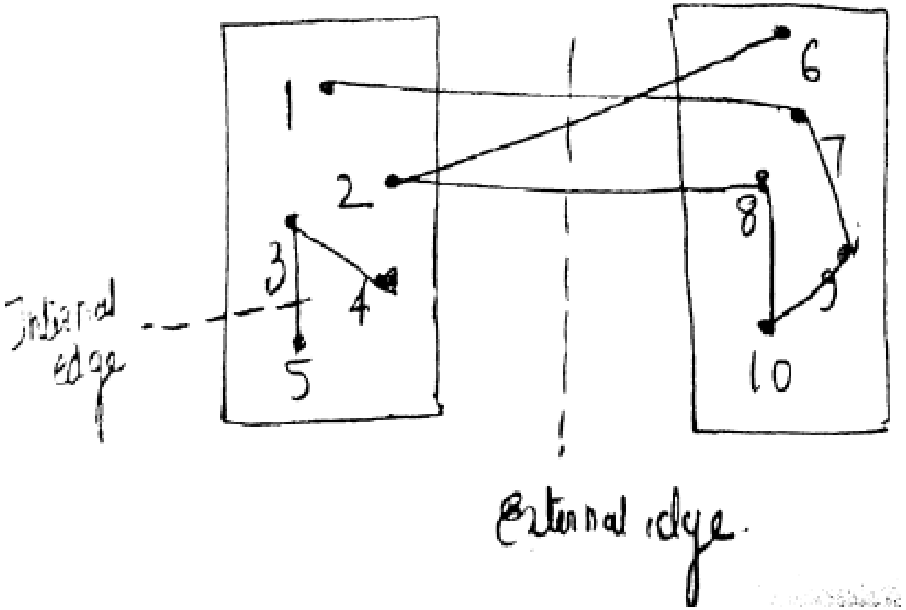
$$\bullet g_{af} = G_a + G_f - 2c_{af} = -3 + 9 - 2\cdot 0 = +6$$

Example:4

With equivalent equations, explain the KL algorithm. Construct the connectivity matrix for the network graph shown in figure. Also, find the gain in the network and cut size, if:

(i) Nodes 1 and 6 are swapped

(ii) Nodes 2 and 8 are swapped

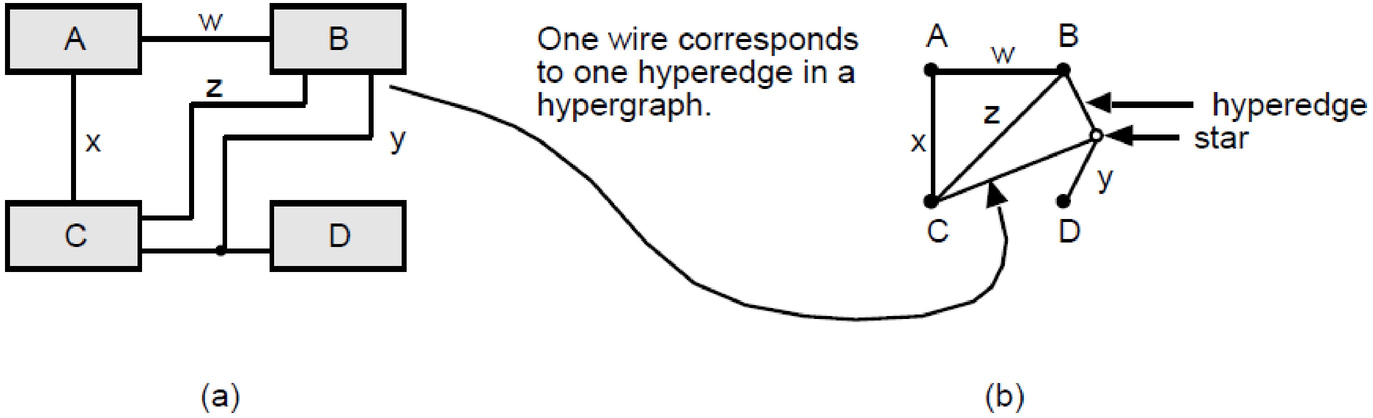


problems need to address before apply the algorithm to network partitioning:

- It **minimizes the number of edges cut**, not the number of nets cut.
- It **does not allow logic cells to be different sizes**.
- It is **expensive in computation time**.
- It does not allow partitions to be **unequal or find the optimum partition size**.
- It does **not allow for selected logic cells to be fixed in place**.
- The results are **random**.
- It does not **directly allow for more than two partitions**.

Fiduccia–Mattheyses (FM) algorithm

- To represent nets with multiple terminals in a network accurately, we can extend the definition of a network graph
- hypergraph with a special type of vertex, a star, and a hyperedge, represents a net with more than two terminals in a network.



A hypergraph. (a) The network contains a net y with three terminals. (b) In the network hypergraph we can model net y by a single hyperedge (B, C, D) and a star node.
Now there is a direct correspondence between wires or nets in the network and hyperedges in the graph.

- In the K–L algorithm, the internal and external edge costs have to be calculated for all the nodes before we can select the nodes to be swapped.
- Then we have to find the pair of nodes that give the largest gain when swapped.
- This requires an amount of computer time that grows as $n^2 \log n$ for a graph with 2n nodes. This n^2 dependency is a major problem for partitioning large networks.
- The Fiduccia–Mattheyses algorithm (the F–M algorithm) is an extension to the K–L algorithm that addresses the differences between nets and edges and also reduces the computational effort.

The **key features** of this algorithm are the following:

- Only one logic cell, the **base logic cell**, moves at a time. In order to stop the algorithm from moving all the logic cells to one large partition, the base logic cell is chosen to maintain balance between partitions.
    - The balance is the ratio of total logic cell size in one partition to the total logic cell size in the other. Altering the balance allows us to vary the sizes of the partitions.
- **Critical nets** are used to simplify the gain calculations.
    - A net is a critical net if it has an attached logic cell that, when swapped, **changes the number of nets cut.** It is only necessary to recalculate the gains of logic cells on critical nets that are attached to the base logic cell.
- The logic cells that are free to move are stored in a **doubly linked list**. The lists are sorted according to gain. This allows the logic cells with maximum gain to be found quickly.

The Ratio-Cut Algorithm

- **Removes the restriction of constant partition sizes.**

- The **cut weight W** for a cut that divides a network into two partitions, A and B , is given by ,

$$W = \sum_{a \in A, b \in B} c_{ab}$$

- The **ratio of a cut** is defined as

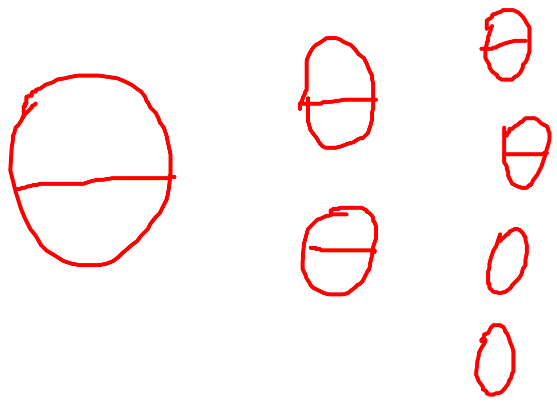
$$\mathbf{R = W / (|A| \; |B|)}$$

The |A| and |B| are size of a partition is equal to the number of nodes it contains (also known as the set cardinality). The cut that minimizes R is called the ratio cut.

- A network is partitioned into small, highly connected groups using ratio cuts.

- A reduced network is formed from these groups.
  - —Each small group of logic cells forms a node in the reduced network.

- Finally, apply the F–M algorithm to improve the reduced network



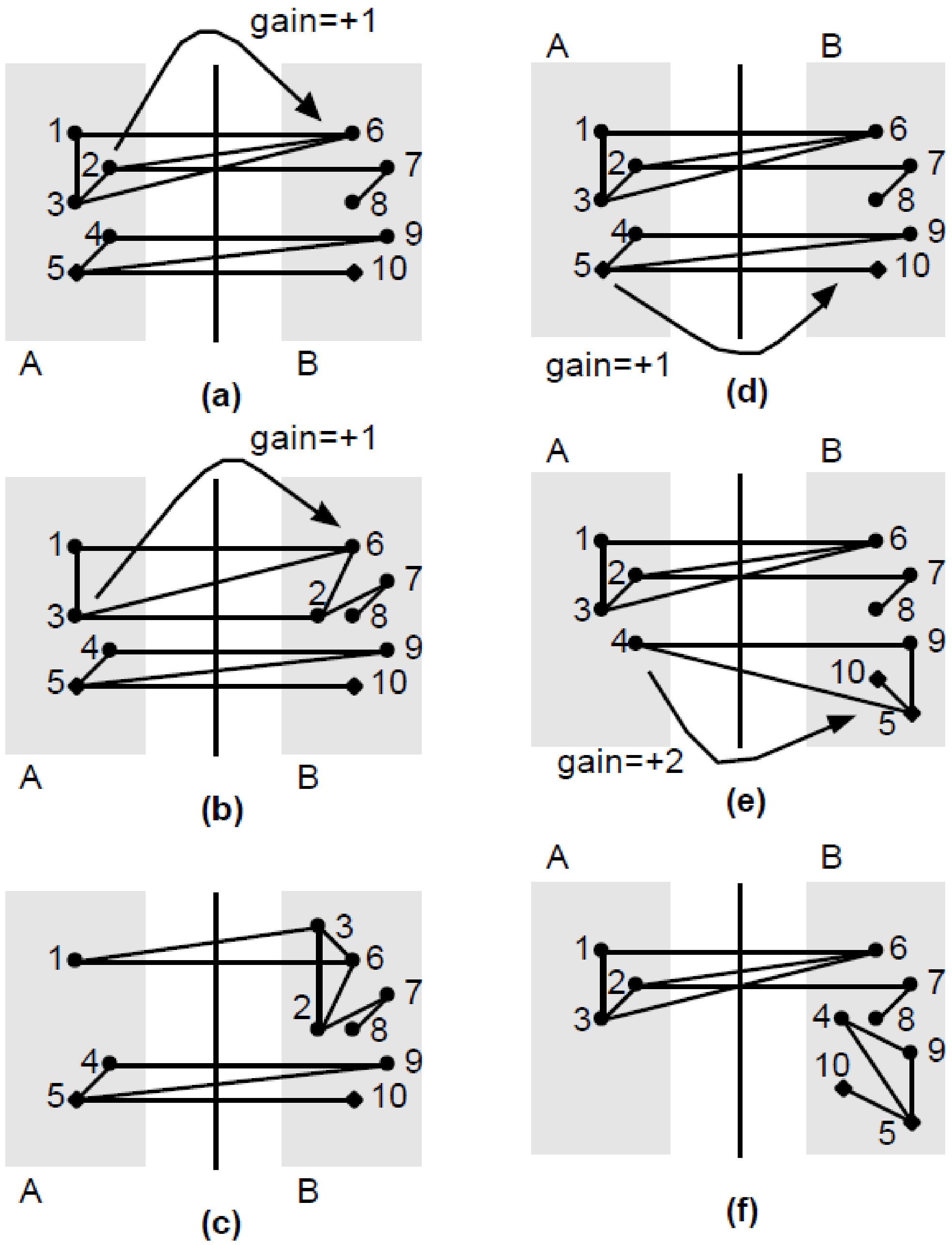
Look-ahead Algorithm

Why Look-ahead?

- **K–L and F–M algorithms consider only the immediate gain** to be made by moving a node.
- When **there is a tie between nodes with equal gain** (as often happens), there is no mechanism to make the best choice.

Algorithm

- The gain for the initial move is called as the **first-level gain.**
- Gains from subsequent moves are then **second-level and higher gains.**
- Define a **gain vector** that contains these gains.
- The **choice of nodes to be swapped are found** Using the **gain vector** .
- This **reduces both the mean and variation** in the number of cuts in the resulting partitions.



- Partitionings (a), (b), and (c) show one sequence of moves – Partition I
- Partitionings (d), (e), and (f) show a second sequence – Partition II

- Partition I:

- The partitioning in (a) can be improved by moving node 2 from A to B with a gain of 1. The result of this move is shown in (b). This partitioning can be improved by moving node 3 to B, again with a gain of 1.

- Partition II:

- The partitioning shown in (d) is the same as (a). We can move node 5 to B with a gain of 1 as shown in (e), but now we can move node 4 to B with a **gain of 2**

Other Partitioning Objectives

Constraints or Objectivies	Purpose	Implemented
Timing Constraints	certain logic cells in a system may need to be located on the same ASIC in order to avoid adding the delay of any external interconnections	Adding weights to nets to make them more important than others.
Power Constraints	Some logic cells may consume more power than others	To assign more than rough estimates of power consumption for each logic cell at the system planning stage, before any simulation has been completed.
Technology Constraints	To include memory on an ASIC	It will keep logic cell together requiring similar technology
Cost Constraints	To use low cost package	To keep ASICs below a certain size.
Test Constraints	Maintain Observability and Controllability	It require that we force certain connection to external

FLOORPLANNING AND PLACEMENT

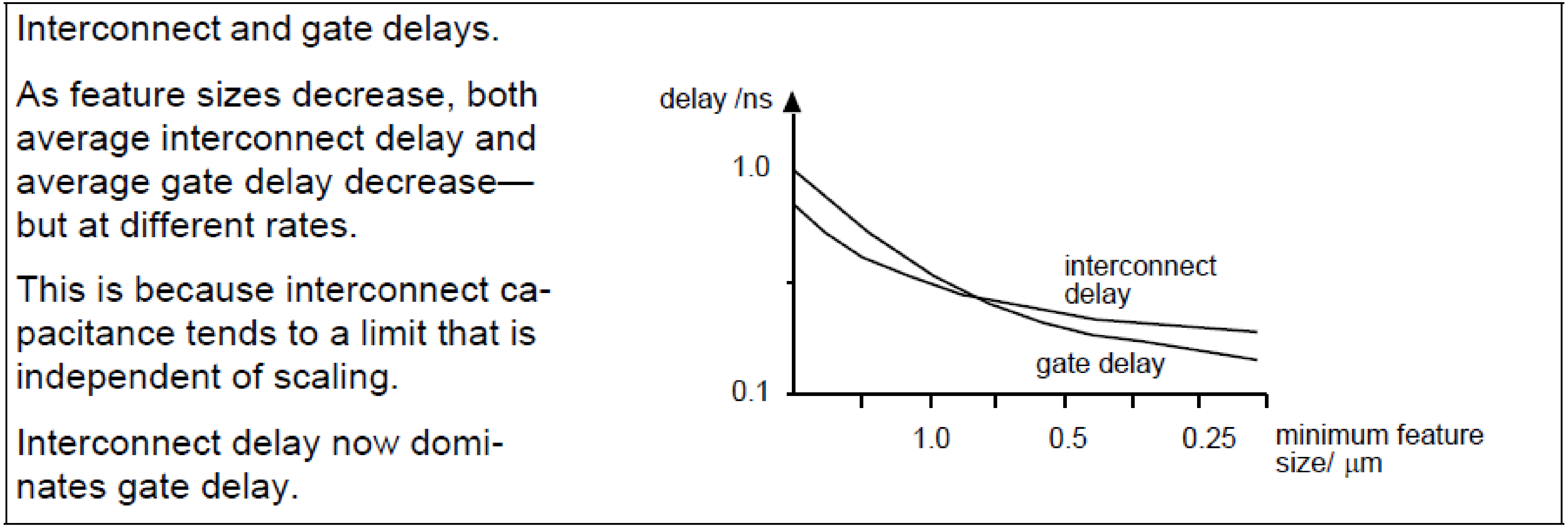
Key terms and concepts: The input to floor planning is the output of system partitioning and design entry—a netlist. The output of the placement step is a set of directions for the routing tools.

Floorplanning

- **Interconnect and gate delay** both **decrease with feature size**—but at different rates.

- Interconnect capacitance bottoms out at $2 pFcm^{-1}$ for a minimum-width wire, but gate delay continues to decrease.

- Floorplanning **predicts interconnect delay by estimating interconnect length.**



Interconnect and gate delays.

As feature sizes decrease, both average interconnect delay and average gate delay decrease—but at different rates.

This is because interconnect capacitance tends to a limit that is independent of scaling.

Interconnect delay now dominates gate delay.

Floorplanning Goals and Objectives

Goals of floorplanning:

- arrange the blocks on a chip
- decide the location of the I/O pads
- decide the location and number of the power pads
- decide the type of power distribution
- decide the location and type of clock distribution

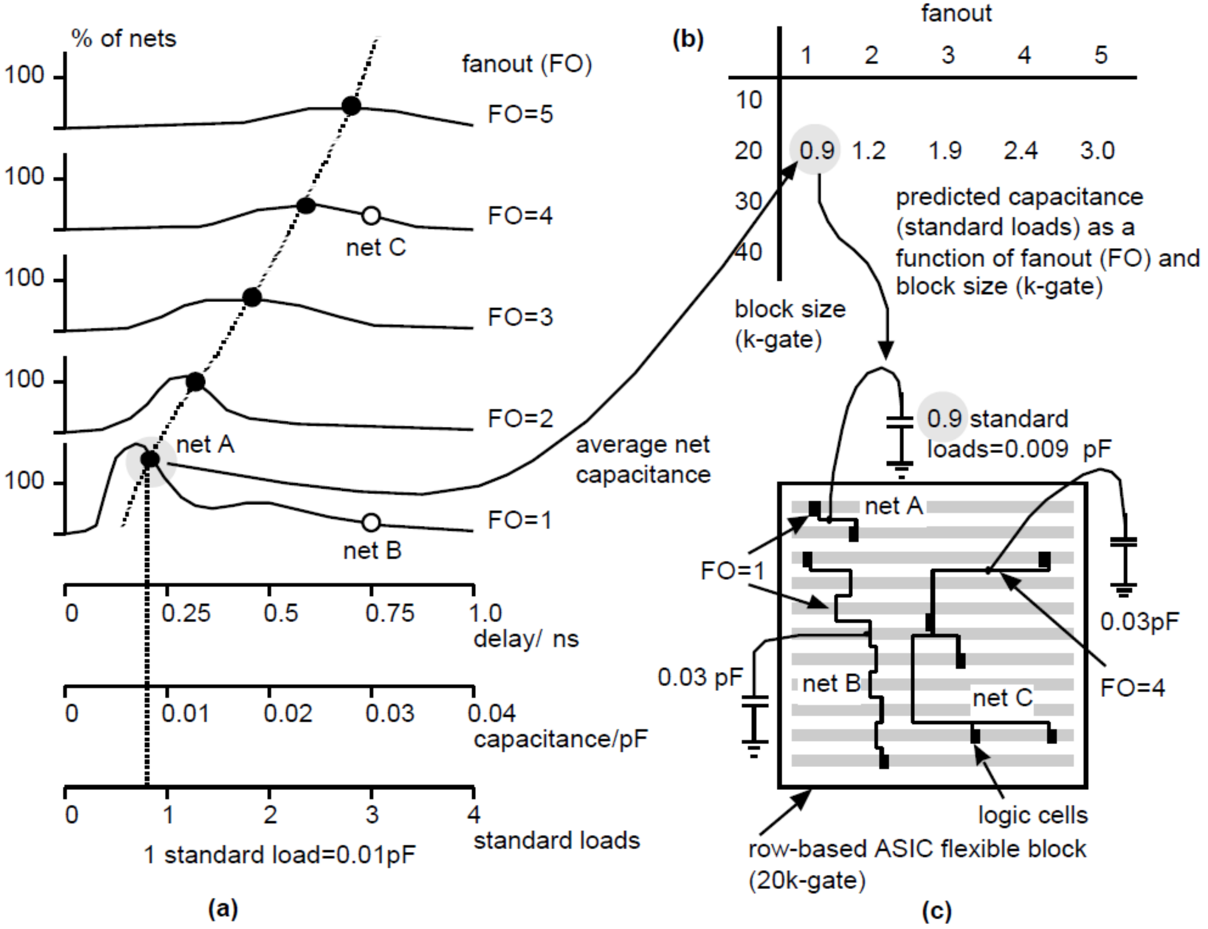
Objectives of floorplanning are:

- to minimize the chip area
- minimize delay

Measurement of Delay in Floorplanning

- In floorplanning we wish **to predict the interconnect delay** before we complete any routing.
- To predict delay we need to **know the parasitic associated with interconnect**:
  - interconnect capacitance (wiring capacitance or routing capacitance)
  - interconnect resistance.
- At the floorplanning stage we know only
  - the fanout (FO) of a net (the number of gates driven by a net)
  - the size of the block that the net belongs to.
- We cannot predict the resistance of the various pieces of the interconnect path since we do not yet know the shape of the interconnect for a net.
- However, we can estimate the total length of the interconnect and thus estimate the total capacitance.

- We **estimate interconnect length** by collecting statistics from **previously routed chips** and analyzing the results.
- From these statistics we **create tables that predict the interconnect capacitance** as a function of net fanout and block size.
- A floorplanning tool can then use these **predicted-capacitance tables** (also known as **interconnect-load tables** or **wire-load tables**).



Predicted capacitance.
(a) Interconnect lengths as a function of fanout (FO) and circuit-block size.
(b) Wire-load table.
There is only one capacitance value for each fanout (typically the average value).
(c) The wire-load table predicts the capacitance and delay of a net (with a considerable error).
Net A and net B both have a fanout of 1, both have the same predicted net delay, but net B in fact has a much greater delay than net A in the actual layout (of course we shall not know what the actual layout is until much later in the design process).

Floorplanning Tools

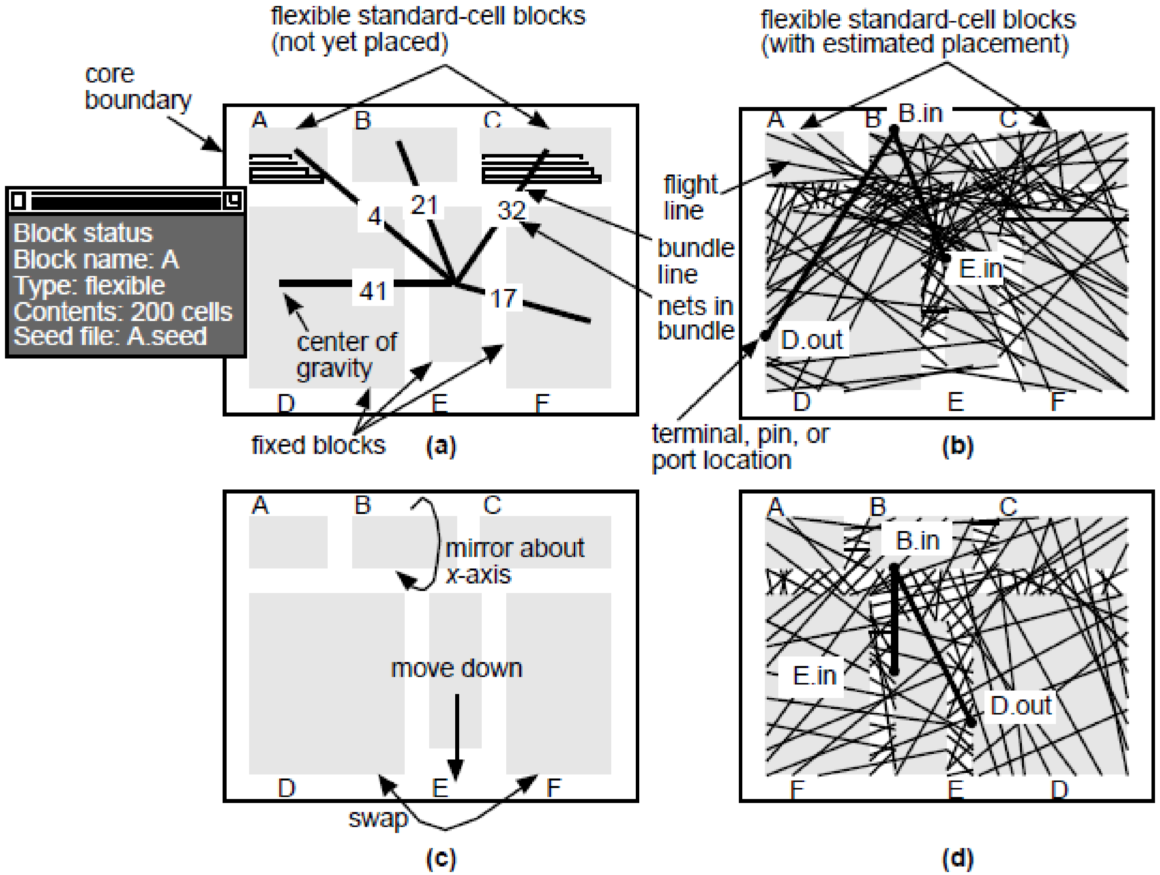
Floorplanning a cell-based ASIC:

(a) Initial floorplan generated by the floorplanning tool.

- Two of the blocks are flexible (A and C) and contain rows of standard cells (unplaced).
- A pop-up window shows the status of block A.
- These are flexible blocks (or variable blocks) because, although their total area is fixed
- Their shape (aspect ratio) and connector locations may be adjusted during the placement step.
- The dimensions and connector locations of the other fixed blocks (perhaps RAM, ROM, compiled cells, or megacells) can only be modified when they are created.
- We may force logic cells to be in selected flexible blocks by seeding. We choose seed cells by name.

  Example: ram_control*

- Seeding may be hard or soft.
  - A hard seed is fixed and not allowed to move during the remaining floorplanning and placement steps.
  - A soft seed is an initial suggestion only and can be altered if necessary by the floor planner.


Block status
Block name: A
Type: flexible
Contents: 200 cells
Seed file: A.seed

(b) An estimated placement for flexible blocks A and C. The connector positions are known and a rat's nest display shows the heavy congestion below block B.

(c) Moving blocks to improve the floorplan.

(d) The updated display shows the reduced congestion after the changes.

Congestion analysis:

We need to control the aspect ratio of our floorplan because we have to fit our chip into the die cavity (a fixed-size hole, usually square) inside a package.

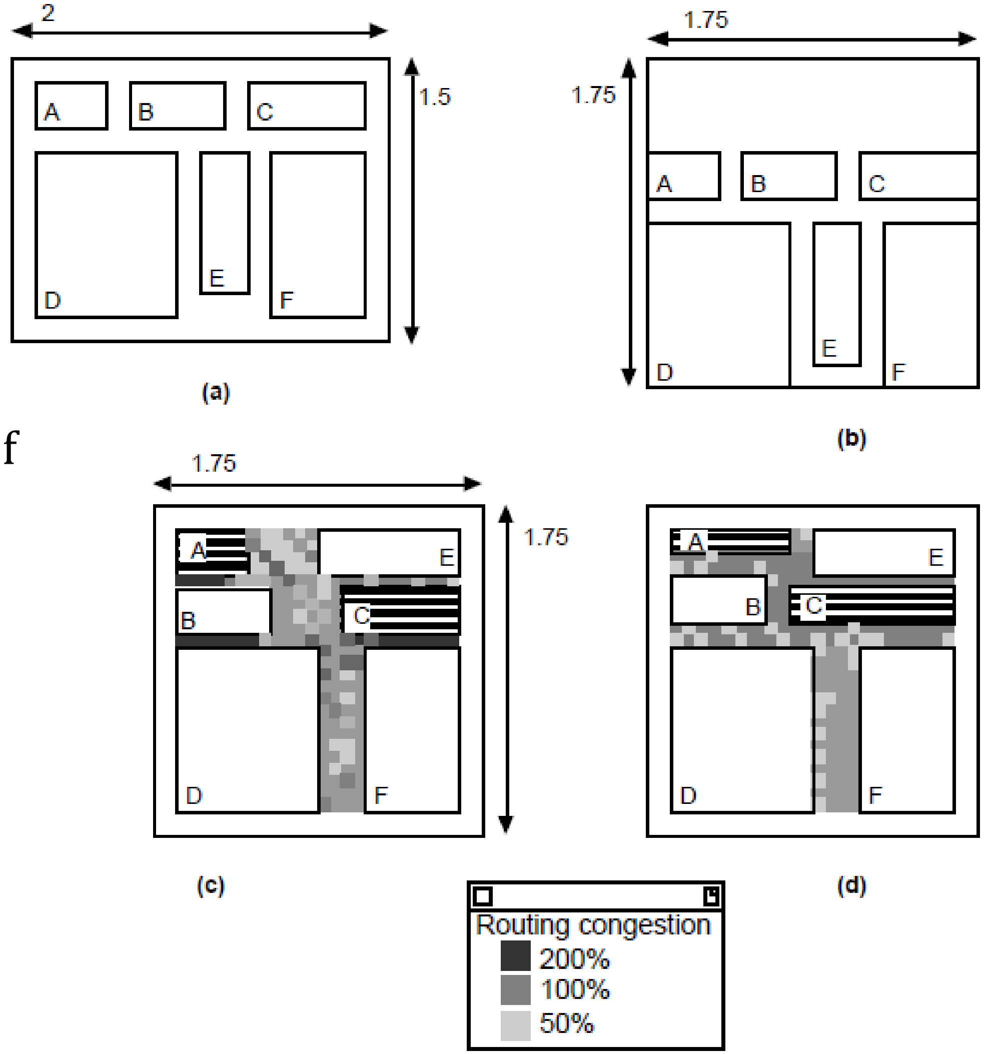
(a) The initial floorplan with a 2:1.5 die aspect ratio.

(b) Altering the floorplan to give a 1:1 chip aspect ratio.

(c) A trial floorplan with a congestion map (routability display).

- There is no standard measure of routability.
- Generally the interconnect channels , (or wiring channels) have a certain channel capacity ; that is, they can handle only a fixed number of interconnects.
- One measure of congestion is the **difference between** the number of interconnects that we actually need, called the **channel density** , and **the channel capacity.**
- Another measure, uses the **ratio of channel density to the channel capacity.**
- Shading indicates the ratio of channel density to the channel capacity.
- Dark areas show regions that cannot be routed because the channel congestion exceeds the estimated capacity.

(d) Resizing flexible blocks A and C alleviates congestion.

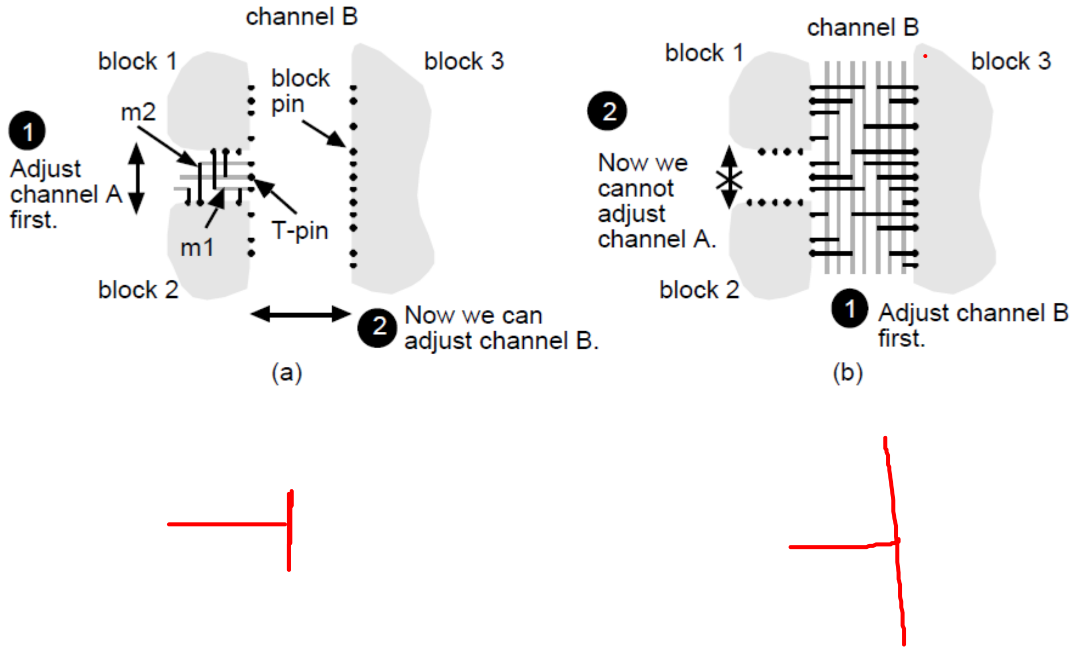


Channel Definition

- During the floorplanning step we assign the areas between blocks that are to be used for interconnect. This process is known as **channel definition** or **channel allocation** .

Routing a T-junction between two channels in two-level metal.

- The dots represent logic cell pins.
- (a) Routing channel A (the stem of the T) first allows us to adjust the width of channel B.
- (b) If we route channel B first (the top of the T), this fixes the width of channel A.
- We have to route the stem of a T-junction before we route the top.
- The general problem of choosing the order of rectangular channels to route is channel ordering .

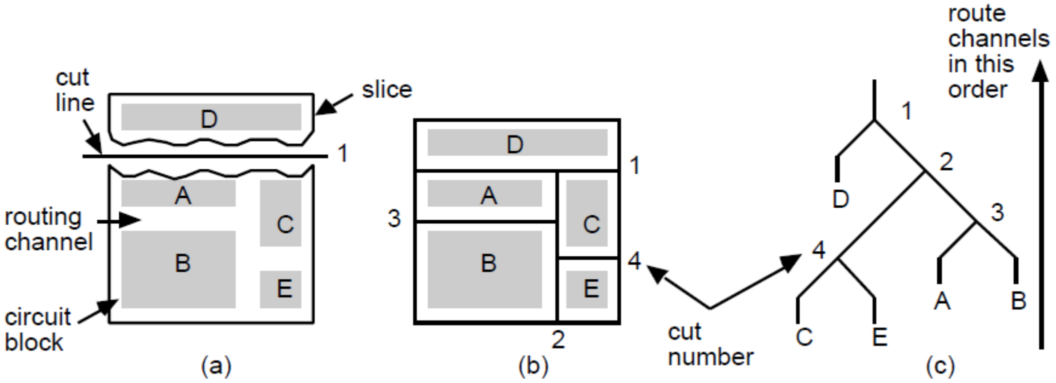


slicing floorplan:

- Defining the channel routing order for a slicing floorplan using a slicing tree.

(a) Make a cut all the way across the chip between circuit blocks.

- Continue slicing until each piece contains just one circuit block.
- Each cut divides a piece into two without cutting through a circuit block.

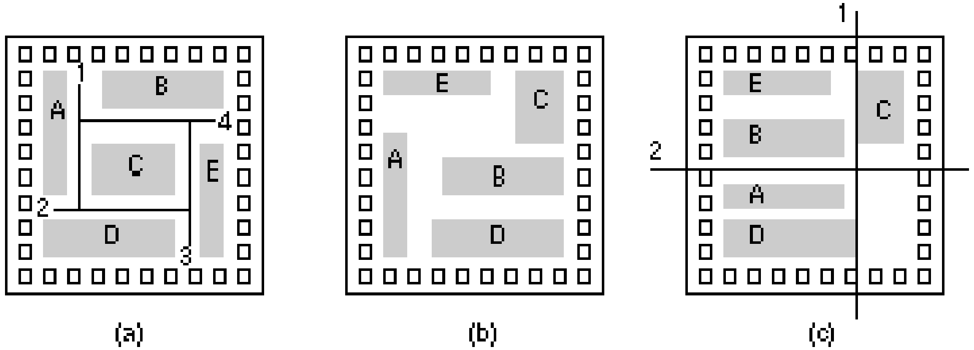


(b) A sequence of cuts: 1, 2, 3, and 4 that successively slices the chip until only circuit blocks are left.

(c) The slicing tree corresponding to the sequence of cuts gives the order in which to route the channels: 4, 3, 2, and finally 1.

Cyclic constraints：

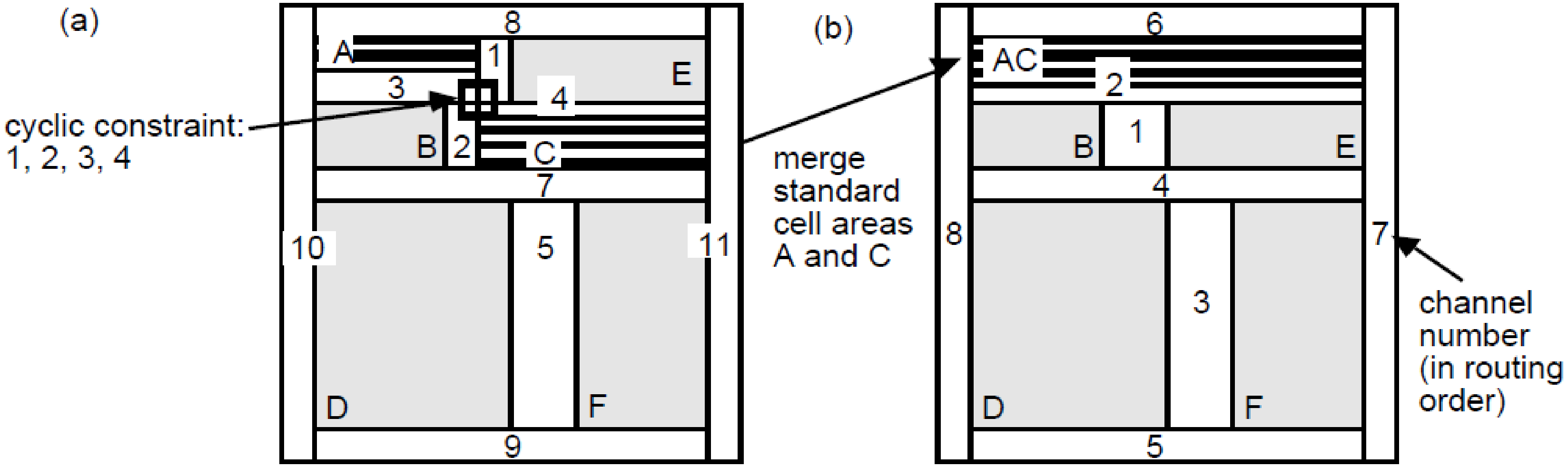
Floorplan that is not a slicing structure. We cannot cut the chip all the way across with a knife without chopping a circuit block in two. This means we cannot route any of the channels in this floorplan without routing all of the other channels first. We say there is a **cyclic constraint** in this floorplan.



(a) A nonslicing floorplan with a cyclic constraint that prevents channel routing.
(b) In this case it is difficult to find a slicing floorplan without increasing the chip area.
(c) This floorplan may be sliced (with initial cuts 1 or 2) and has no cyclic constraints, but it is inefficient in area use and will be very difficult to route

Channel definition and ordering:

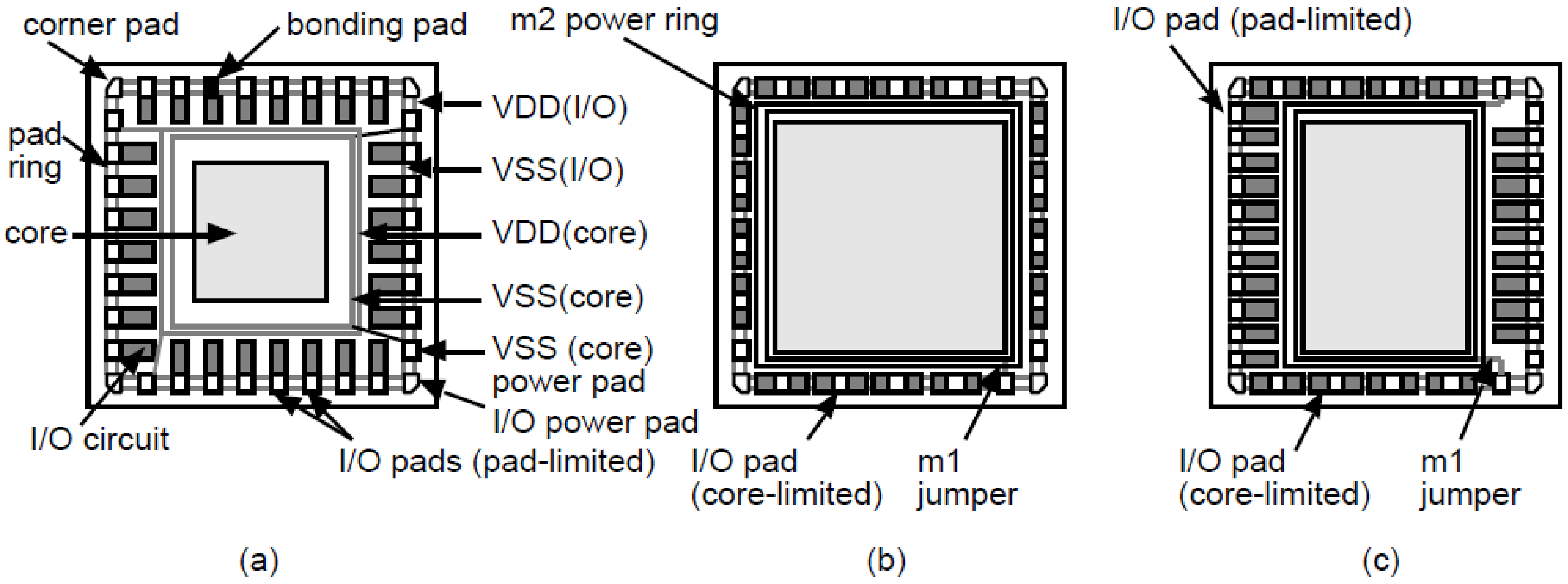
- We can remove the cyclic constraint by moving the blocks again, but this increases the chip size.
- An alternative solution:
  - We **merge the flexible standard cell areas** A and C.
  - We can do this by selective **flattening of the netlist**. Sometimes flattening can reduce the routing area because routing between blocks is usually less efficient than routing inside the row-based blocks.



(a) We can eliminate the cyclic constraint by merging the blocks A and C. (b) A slicing structure

I/O and Power Planning

- Signals flow **onto and off the chip** and we need to supply power.
- We need to **consider the I/O and power constraints** early in the floorplanning process.
- A silicon **chip or die** (plural die, dies, or dice) is **mounted on a chip carrier** inside a chip package .
- Connections are made by **bonding the chip pads to fingers on a metal lead frame** that is part of the package.
- The **metal lead-frame fingers** connect to the **package pins** .
- A die consists of a **logic core inside a pad ring** .



(a) A pad-limited die.

- The number of pads determines the die size.
- Use tall, thin pad-limited pads-maximize the number of pad- fit around the outside of the chip.

(b) A core-limited die

- The core logic determines the die size.
- Use short, wide core-limited pads.

(c) Using both pad-limited pads and core-limited pads for a square die.

- To change the aspect ratio of a die to be different from that of the core

- **Special power pads are used** for the positive supply, or VDD, power buses (or power rails) and the ground or negative supply, VSS or GND.
  - Usually one set of VDD/VSS pads supplies **one power ring** that runs around the pad ring and **supplies power to the I/O pads only**.
  - Another set of VDD/VSS pads connects to a **second power ring** that **supplies the logic core**.
- We sometimes call the I/O power as **dirty power** since it has to supply large transient currents to the output transistors.
- We keep dirty power separate **to avoid injecting noise** into the internal-logic power (the clean power).
- I/O pads also contain special circuits **to protect against electrostatic discharge** (ESD).
- These circuits can **withstand very short high-voltage** (several kilovolt) pulses that can be generated during human or machine handling.

Bonding pads:

(a) This chip uses both pad-limited and core-limited pads.

(b) A hybrid corner pad.

Magnified views of southeast corner of our example chip shows the different types of I/O cells.

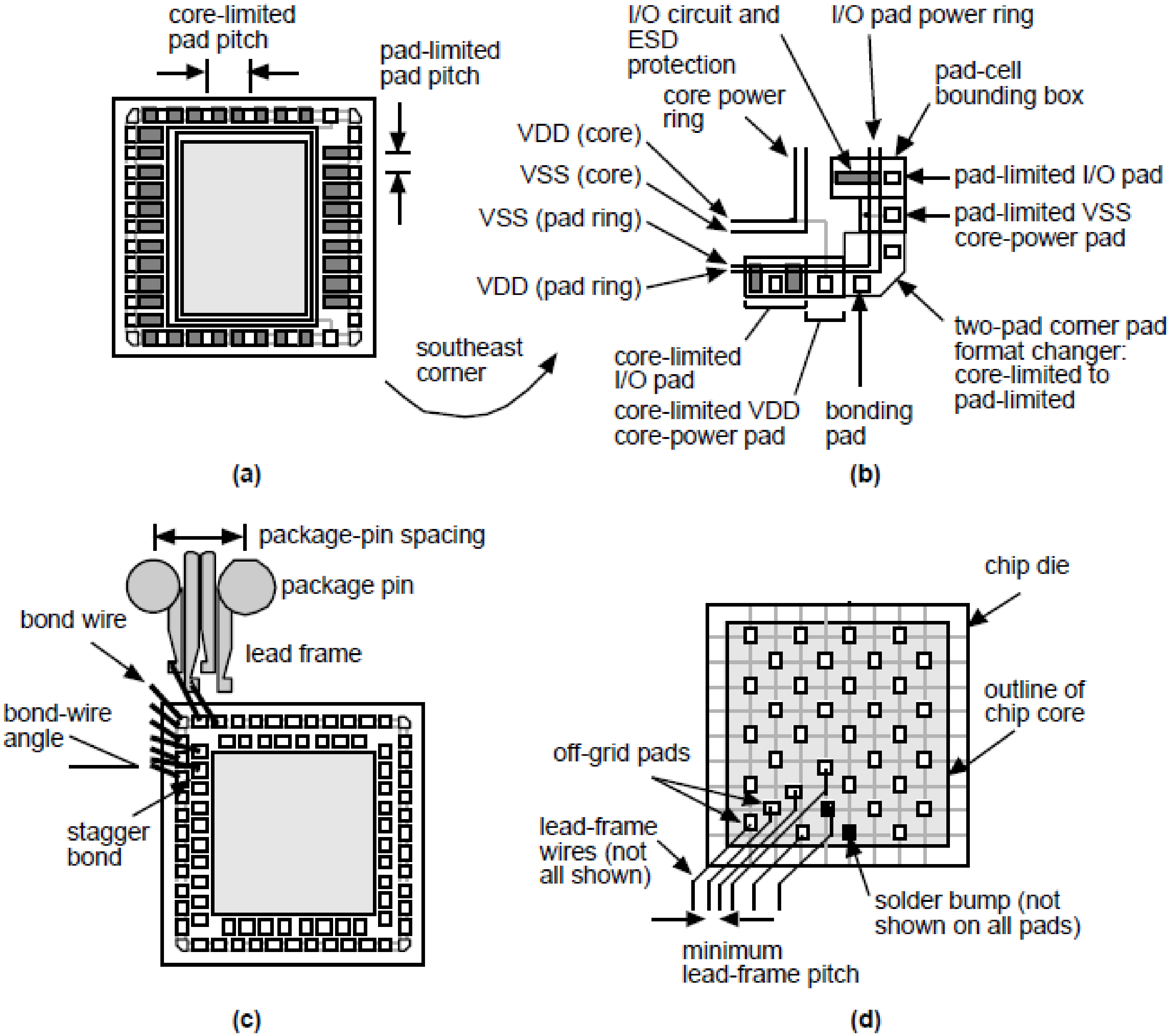
(c) A chip with stagger-bonded pads.

Using two rows of I/O pads.

Design rules of bonding wires become very important.

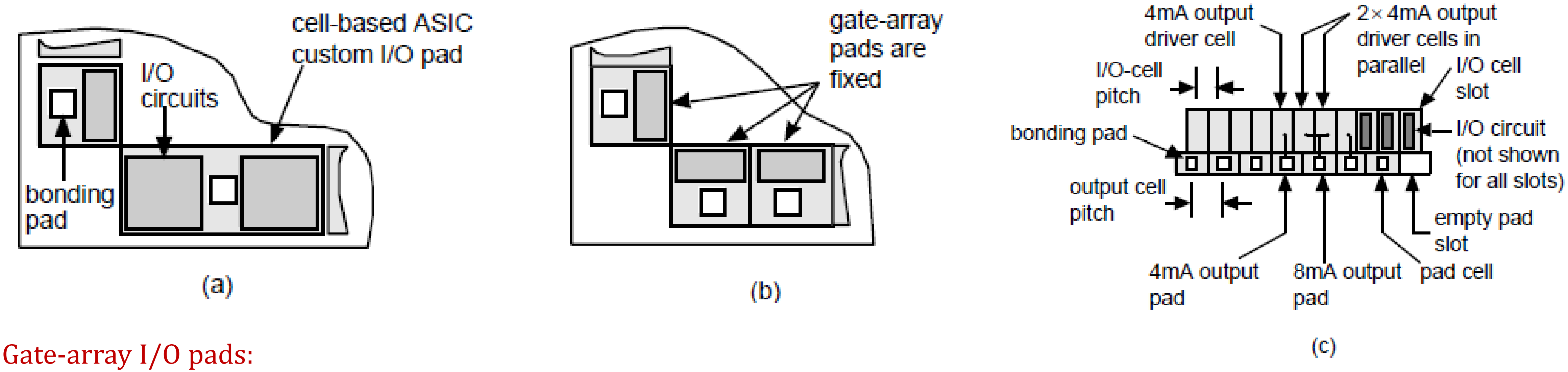
(d) An area-bump bonded chip (or flip-chip).

The chip is turned upside down and solder bumps connect the pads to the lead frame.



In MGA,

- Pad spacing and I/O cell spacing is fixed
- Each pad occupies a fixed pad slot
- This means that the properties of the pad I/O are also fixed but, if we need to, we can parallel adjacent output cells to increase the drive.
- Flexibility can increase by using further I/O cells, the I/O cell pitch (smaller than the pad pitch)
  - Example: three 4 mA driver cells can occupy two pad slots. Then we can use two 4 mA output cells in parallel to drive one pad, forming an 8 mA output pad.
- This arrangement also means the I/O pad cells can be changed without changing the base array.



Gate-array I/O pads:

(a) Cell-based ASICs may contain pad cells of different sizes and widths.(b) A corner of a gate-array base. (c) A gate-array base with different I/O cell and pad pitches.

Power distribution:

(a) Power distributed using m1 for VSS and m2 for VDD.

This helps minimize the number of vias and layer crossings needed but causes problems in the routing channels.

(b) In this floorplan m1 is run parallel to the longest side of all channels, the channel spine.

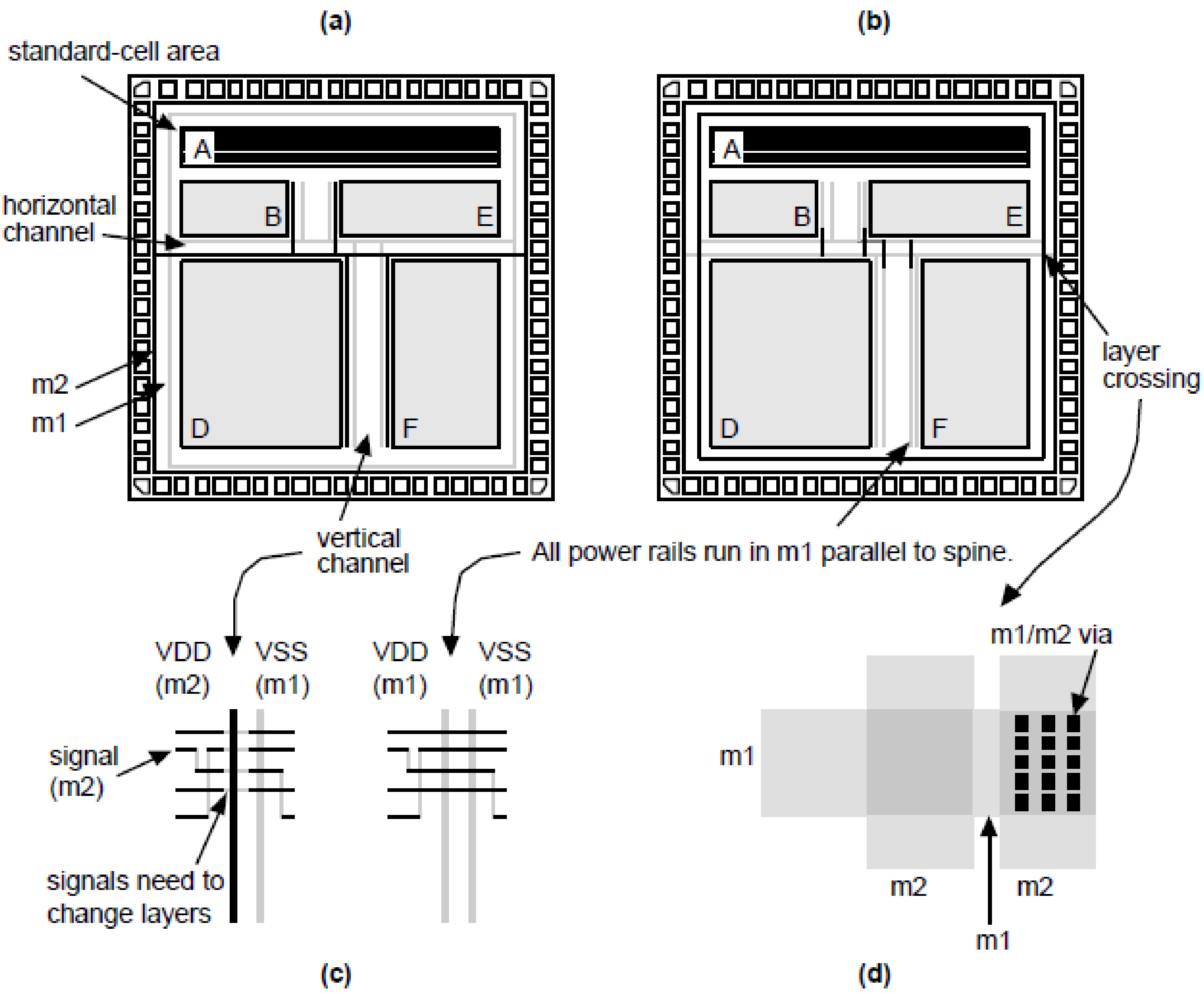
This can make automatic routing easier but may increase the number of vias and layer crossings.

(c) An expanded view of part of a channel (interconnect is shown as lines).

If power runs on different layers along the spine of a channel, this forces signals to change layers.

(d) A close-up of VDD and VSS buses as they cross.

Changing layers requires a large number of via contacts to reduce resistance.



Clock Planning

Clock distribution:

(a) A clock spine for a gate array.

- Clock Spine routing scheme or fish bone type clock distribution scheme
- all clock pins driven directly from the clock driver.
- MGAs and FPGAs use this scheme

(b) A clock spine for a cell-based ASIC

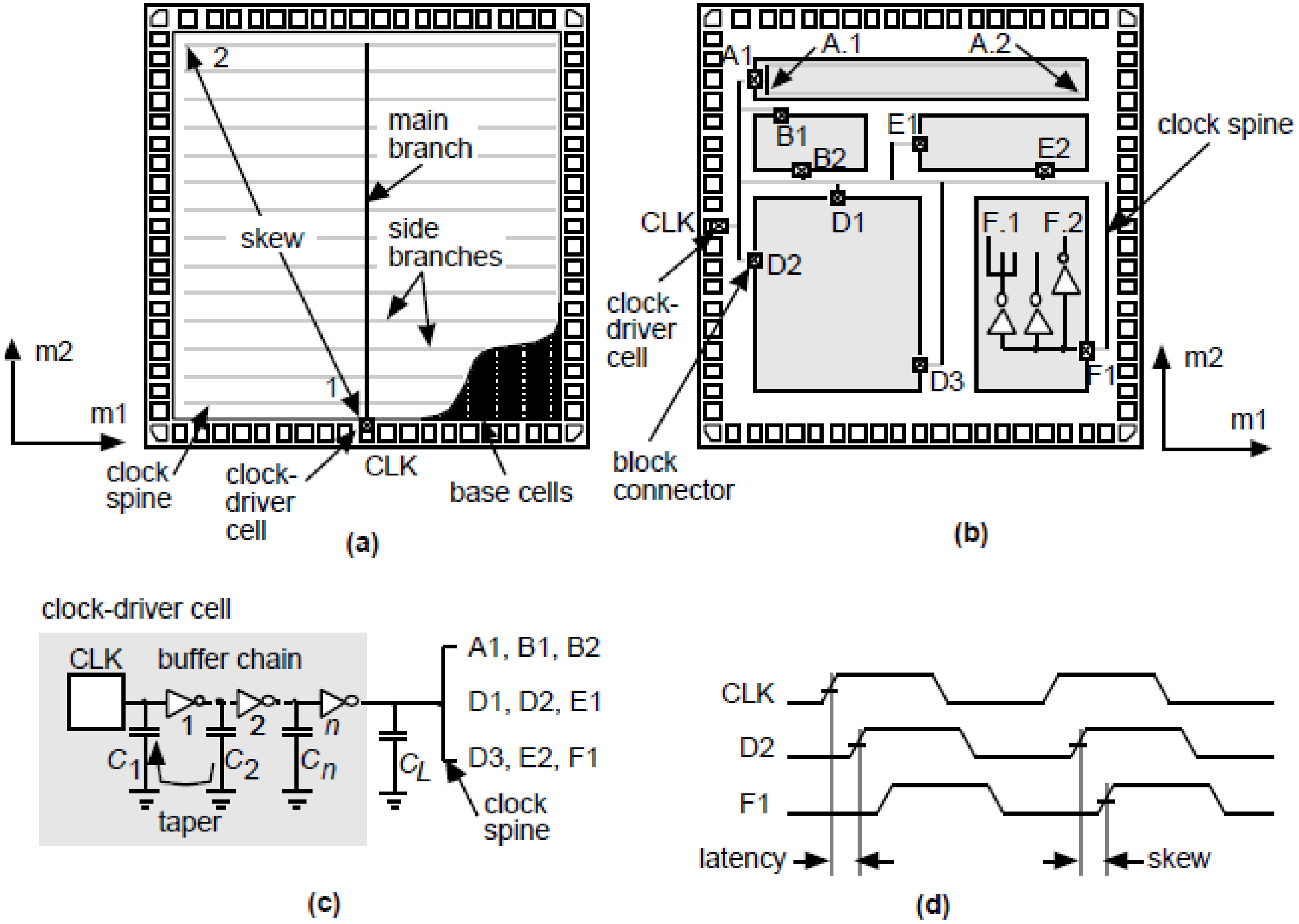
- typical chips have thousands of clock nets.

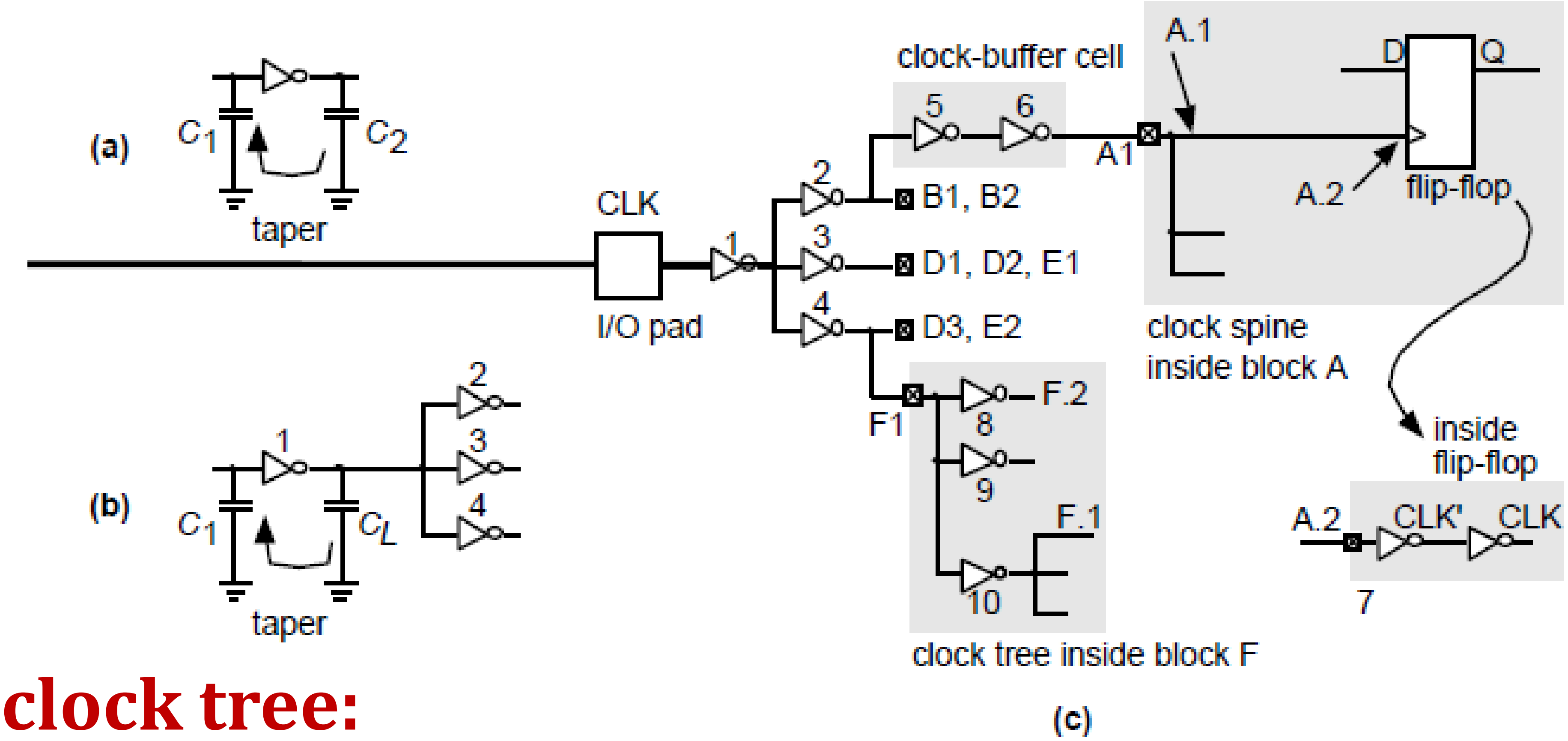
(c) A clock spine is usually driven from one or more clock-driver cells.

- Delay in the driver cell is a function of the number of stages and the ratio of output to input capacitance for each stage (taper).

(d) Clock latency and clock skew.

- We would like to minimize both latency and skew.





A clock tree:

(a) Minimum delay is achieved when the taper of successive stages is about 3.

(b) Using a fanout of three at successive nodes.

(c) A clock tree for a cell-based ASIC

We have to balance the clock arrival times at all of the leaf nodes to minimize clock skew.

Placement

Decide the locations of cells in a block

- After completing a floorplan we can begin placement of the logic cells within the flexible blocks.
- Placement is much more suited to automation than floorplanning.
- After we complete floorplanning and placement, we can predict both intrablock and interblock capacitances.
- This allows us to return to logic synthesis with more accurate estimates of the capacitive loads that each logic cell must drive.
- **Goals:**

  (1) Guarantee the router can complete the routing step

  (2) Minimize all the critical net delays

  (3) Make the chip as dense as possible
- **Objectives:**

  (1) Minimize power dissipation

  (2) Minimize crosstalk between signals

Placement Algorithms

- **Iterative Placement Improvement**

It takes an existing placement and tries to improve it by moving logic cells.

Two parts of algorithm:

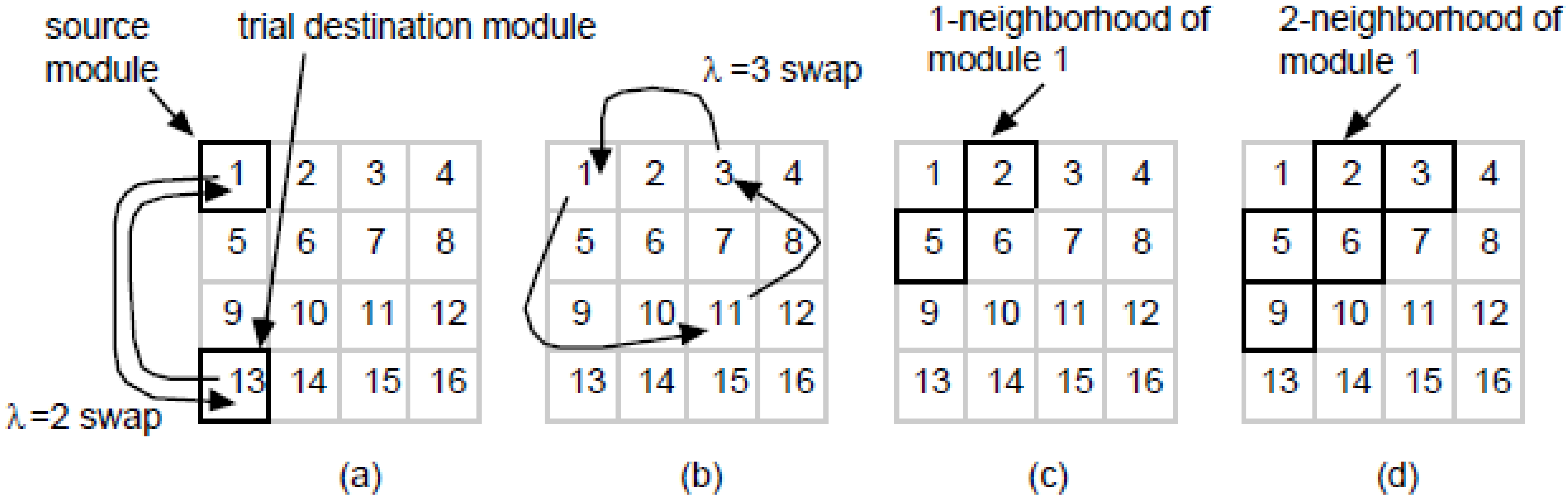
1. The selection criteria that decides which logic cells to try moving
2. The measurement criteria that decides whether to move selected cells.

Different interchange or iterative exchange methods:

1. Pairwise interchange
2. Force-directed interchange
3. Force directed relaxation
4. Force directed pairwise relaxtion

- All of these methods usually consider only pairs of logic cells to be exchanged.
- A source logic cell is picked for trial exchange with a destination logic cell.
- **Steps:**
  1. Select the source logic cell at random.
  2. Try all the other logic cells in turn as the destination logic cell.
  3. Use any of the measurement methods to decide on whether to accept the interchange.
  4. The process repeats from step 1, selecting each logic cell in turn as a source logic cell.

Pairwise interchange



(a) Swapping the source logic cell with a destination logic cell in pairwise interchange.

(b) Sometimes we have to swap more than two logic cells at a time to reach an optimum placement, but this is expensive in computation time.

- Limiting the search to neighborhoods reduces the search time.
- interchange that considers only destination logic cells in a neighborhood cells within a **certain distance, e,** of the source logic cell.

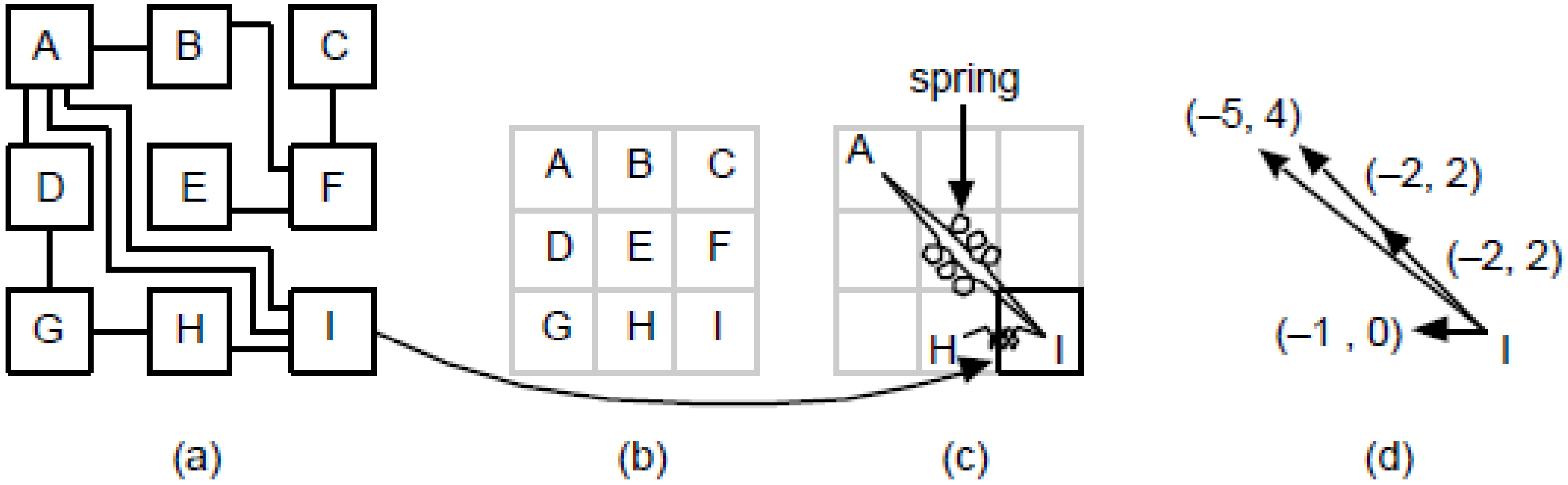
(c) A one-neighborhood.

(d) A two-neighborhood.

- Neighborhoods are also used in some of the **force-directed placement methods .**
- Imagine identical springs connecting all the logic cells we wish to place.
- The number of springs is equal to the number of connections between logic cells.
- The effect of the springs is to pull connected logic cells together.
- The more highly connected the logic cells, the stronger the pull of the springs.
- The force on a logic cell i due to logic cell j is given by Hooke’s law , which says the force of a spring is proportional to its extension:

$$F_{ij} = -c_{ij} x_{ij}$$

- The vector component x_{ij} is directed from the center of logic cell i to the center of logic cell j .
- The c_{ij} form the connectivity or cost matrix (the matrix element c_{ij} is the number of connections between logic cell I and logic cell j).



Force-directed placement.

(a) A network with nine logic cells.

(b) We make a grid (one logic cell per bin).

(c) Forces are calculated as if springs were attached to the centers of each logic cell for each connection.

The two nets connecting logic cells A and I correspond to two springs.

(d) The forces are proportional to the spring extensions.

Force-directed iterative placement Algorithms:

(a) Force-directed interchange.

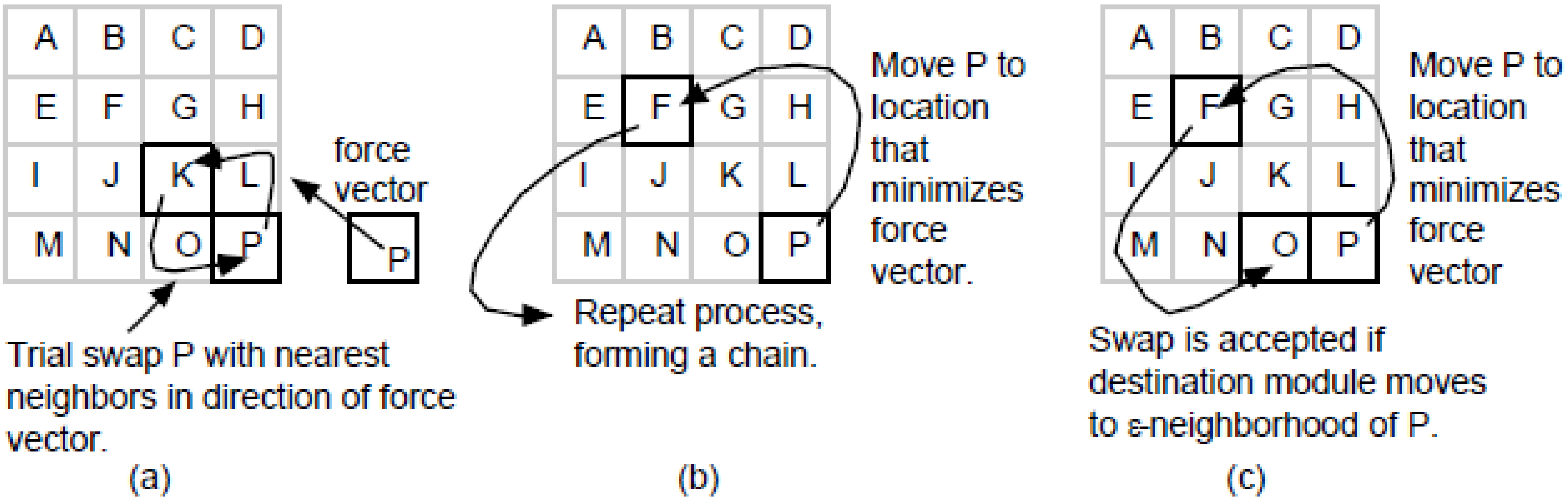
uses the force vector to select a pair of logic cells to swap.

(b) Force-directed relaxation.

a chain of logic cells is moved

(c) Force-directed pairwise relaxation.

swaps one pair of logic



Trial swap P with nearest neighbors in direction of force vector.

(a)

Move P to location that minimizes force vector.

Repeat process, forming a chain.

(b)

Move P to location that minimizes force vector

Swap is accepted if destination module moves to ε-neighborhood of P.

(c)

Floorplanning and Placement design flow

1. **Design entry.** The input is a **logical description with no physical information.**

2. **Initial synthesis.** The initial synthesis contains **little or no information on any interconnect loading.** The **output of the synthesis tool** (typically an EDIF netlist) **is the input to the floorplanner.**

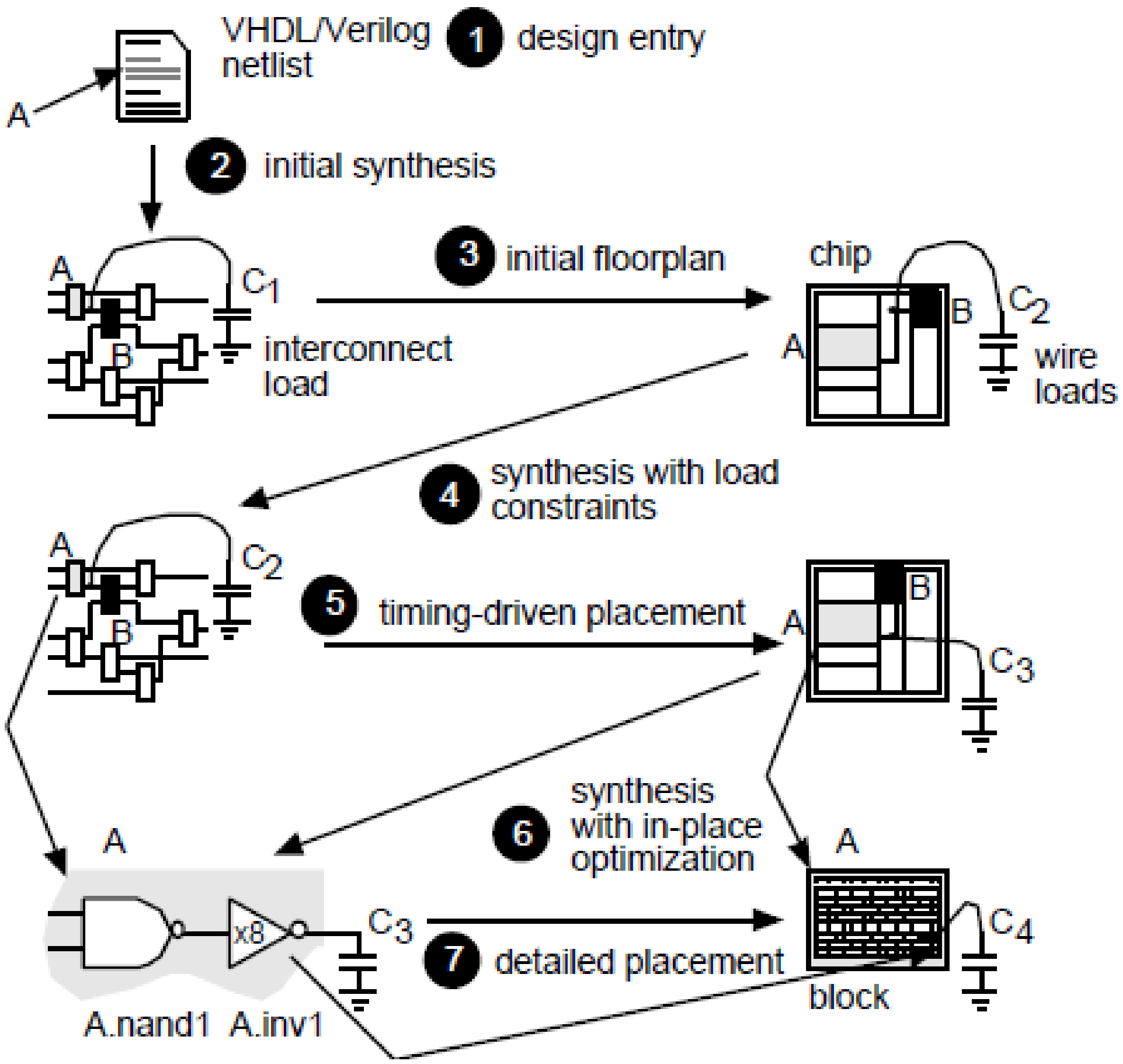
3. **Initial floorplan**. From the **initial floorplan interblock capacitances are input to the synthesis tool as load constraints and intrablock capacitances are input as wire-load tables.**

4. **Synthesis with load constraints**. At this point **the synthesis tool is able to resynthesize** the logic based on estimates of the interconnect capacitance each gate is driving. The synthesis tool **produces a forward annotation file to constrain path delays in the placement step.**

5. **Timing-driven placement.** After placement using constraints from the synthesis tool, **the location of every logic cell on the chip is fixed and accurate estimates of interconnect delay can be passed back to the synthesis tool.**

6. **Synthesis with in-place optimization (IPO).**The synthesis tool changes the **drive strength of gates based on the accurate interconnect delay estimates from the floorplanner without altering the netlist structure.**

7. **Detailed placement.** The placement information is **ready to be input to the routing step.**



Timing-driven floorplanning and placement design flow.

Routing

Once the designer has

- Floorplanned a chip
- The logic cells within the flexible blocks have been placed
- Time to make the connections by routing the chip.

- This is still a hard problem that is made easier by dividing it into smaller problems.
- Routing is usually split into
  - Global routing followed by detailed routing.

Global routing:

- **Goal:** Determine the location of all the interconnect.
- **Objective:** Minimize the total interconnect area used.

Detailed routing:

- **Goal :** Completely route all the interconnect on the chip.
- **Objective :** Minimize the total interconnect length used.

Global Routing

- The details of **global routing differ** slightly between
  - **cell-based ASICs, gate arrays, and FPGAs**, but the principles are the same.

- A global router does not make any connections, **it just plans them.**

- Global route the whole chip (or large pieces if it is a large chip) **before detail routing the whole chip (or the pieces).**

Goals and Objectives

- **Input to routing**
  - **Floorplan** that includes the locations of all the fixed and flexible blocks;
  - **Placement information** for flexible blocks;
    - **Locations of all the logic cells.**
- **Goal** of global routing
  - **To provide complete instructions to the detailed router** on where to route every net.
- **Objectives** of global routing
  - ***Minimize the total interconnect length.***
  - ***Maximize the probability that the detailed router can complete the routing.***
  - ***Minimize the critical path delay.***

Detailed Routing

Goal:

- The goal of detailed routing is to **complete all the connections between logic cells.**

Objectives:

- The most common objective is to minimize one or more of the following:
  - The total **interconnect length and area**
  - The **number of layer changes** that the connections have to make
  - The **delay of critical paths**

- Minimizing the number of layer changes corresponds to minimizing the number of vias that add parasitic resistance and capacitance to a connection.

- The global routing step determines the channels to be used for each interconnect.
- Using this information the detailed router decides the exact location and layers for each interconnect.
- Figure shows typical metal rules. These rules determine the m1 routing pitch (track pitch , track spacing , or just pitch).
- We can set the m1 pitch to one of three values:
  - via-to-via (VTV) pitch (or spacing),
  - via-to-line (VTL or line-to-via) pitch, or
  - line-to-line (LTL) pitch.

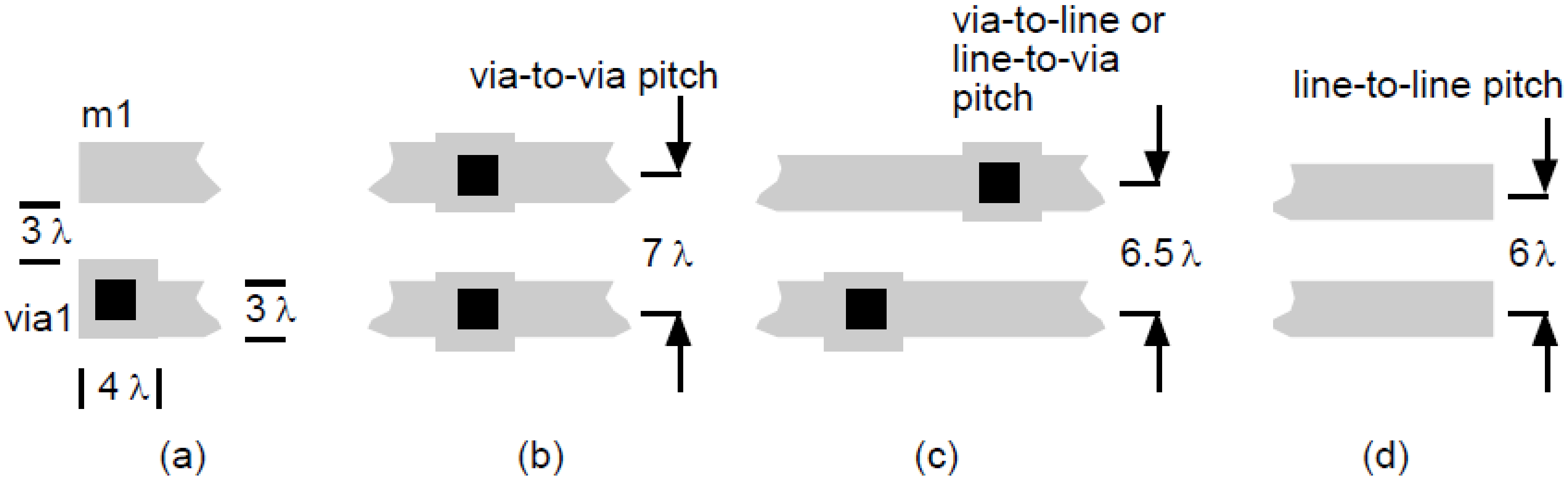
The metal routing pitch.

(a) An example of λ-based metal design rules for m1 and via1 (m1/m2 via).

(b) Via-to-via pitch for adjacent vias.

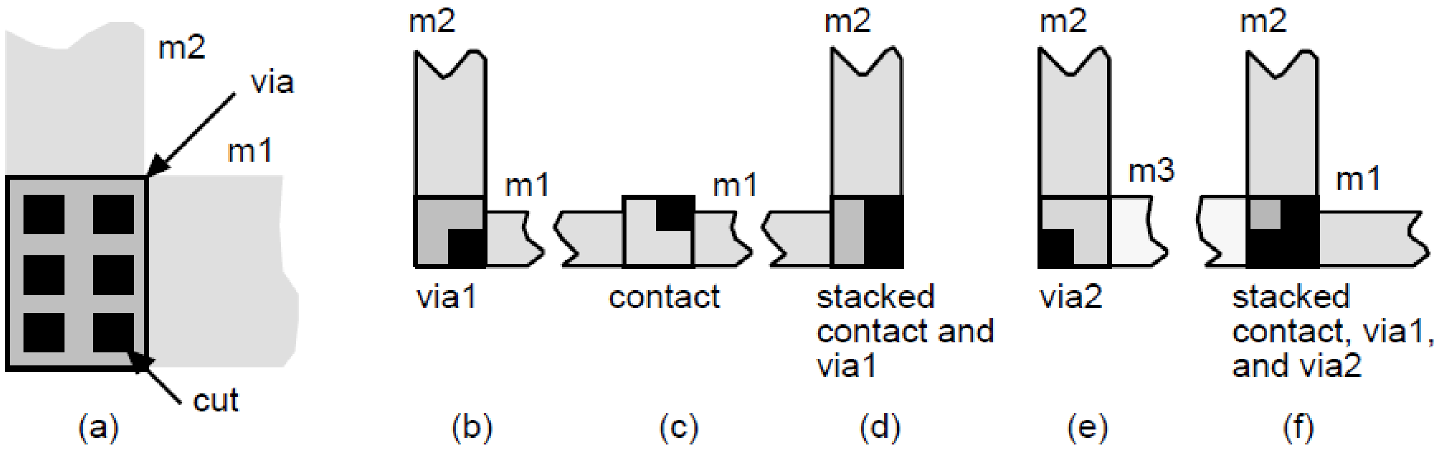
(c) Via-to-line (or line-to-via) pitch for nonadjacent vias.

(d) Line-to-line pitch with no vias.



- The same choices apply to the m2 and other metal layers if they are present.
- **Via-to-via spacing**
  - allows the router to place vias adjacent to each other.
  - easiest for a router to use and the most common.
- **Via-to-line spacing**
  - hard to use in practice because it restricts the router to nonadjacent vias.
- **line-to-line spacing**
  - prevents the router from placing a via at all without using jogs and is rarely used.
- Using either via-to-line or via-to-via spacing means that the routing pitch is larger than the minimum metal pitch

- Sometimes draw a distinction between a cut and a via when a large connections are present.
  - We split or stitch a **large via into identically sized cuts** (sometimes called a waffle via).
  - Because of the profile of the metal in a contact and the **way current flows into a contact**, often the total resistance of several **small cuts is less than that of one large cut**.
  - Using identically sized cuts also means the processing conditions during **contact etching,** which may vary with the area and perimeter of a contact, are the same for every cut on the chip.
- In a stacked via the contact cuts all overlap in a layout plot.
  - Figure (b–f) show an alternative way to draw contacts and vias.



(a) A large m1 to m2 via. The black squares represent the holes (or cuts) that are etched in the insulating material between the m1 and m2 layers.

(b) A m1 to m2 via (*a via1).*

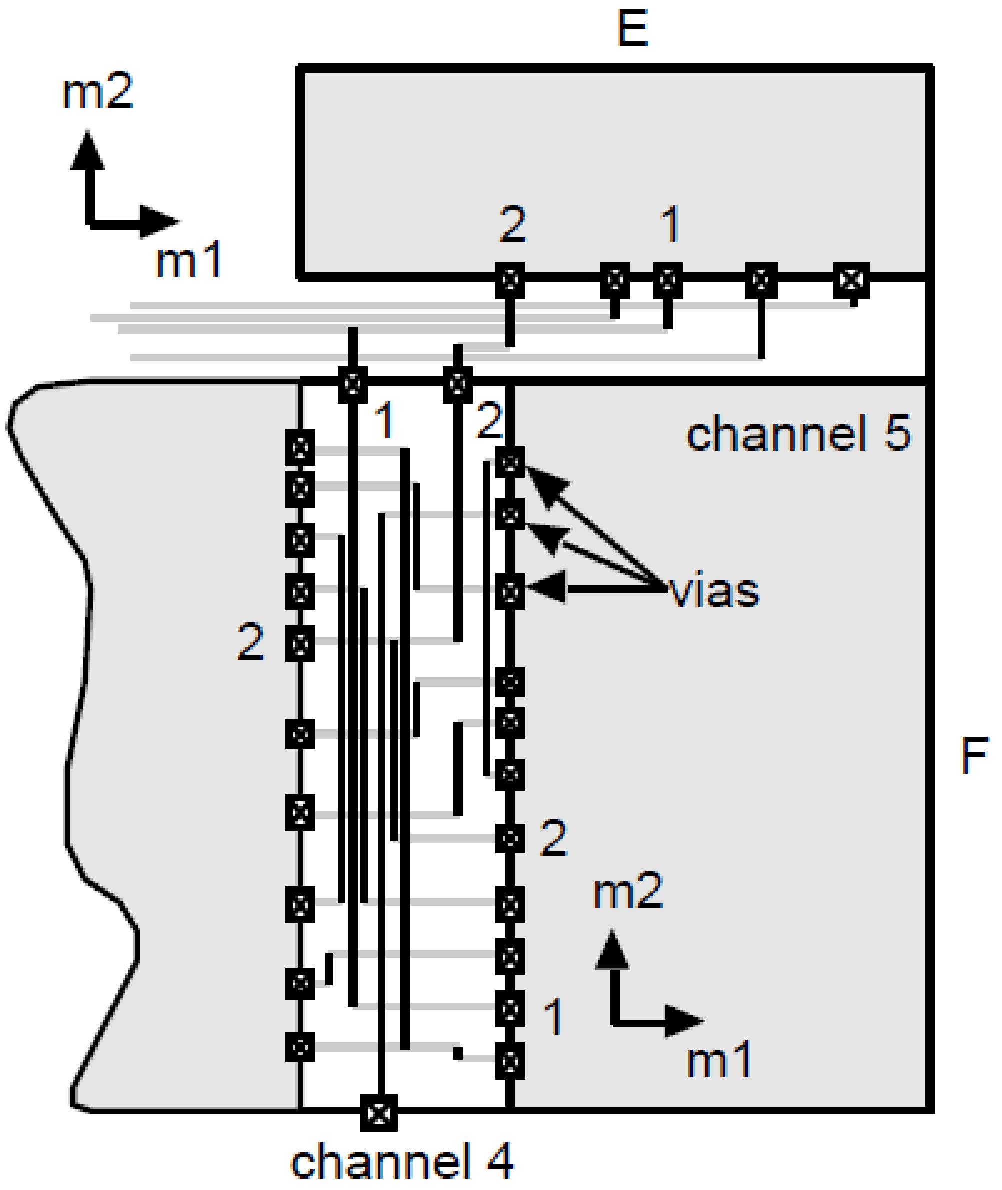
(c) A contact from m1 to diffusion or polysilicon (*a contact).*

(d) A via1 placed over (or stacked over) a contact.

(e) A m2 to m3 via *(a via2).*

(f) A via2 stacked over a via1 stacked over a contact.

- In a two-level metal CMOS ASIC technology we complete the wiring using the **two different metal layers** for the horizontal and vertical directions, one layer for each direction.
- Both channel 4 and channel 5 use m1 in the horizontal direction and m2 in the vertical direction.
- If the logic cell connectors are on m2 this requires vias to be placed at every logic cell connector in channel 4.



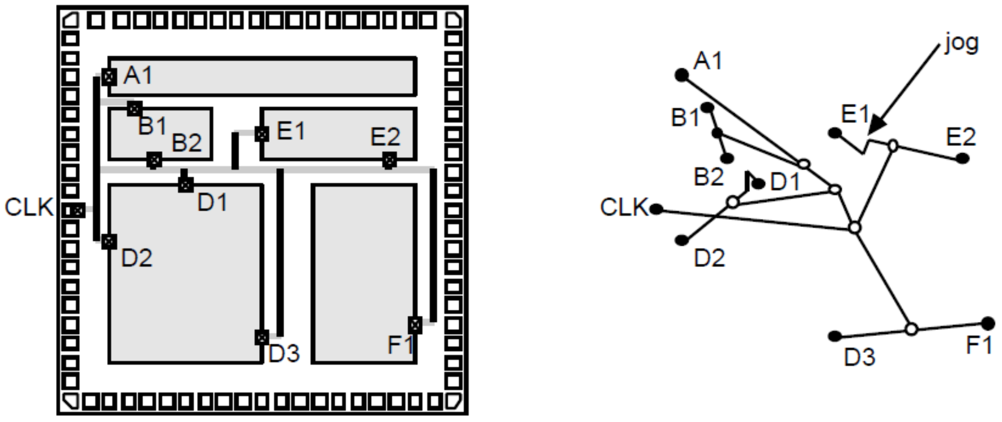
Special Routing

- The routing of nets that require special attention, clock and power nets.
- The architecture and structure of these nets is performed as part of floorplanning, but the sizing and topology of these nets is finalized as part of the routing step

1. **Clock Routing**
2. **Power Routing**

Clock Routing

- Gate arrays normally use a **clock spine (a regular grid)**, eliminating the need for special routing.
- The **clock distribution grid is designed** at the same time as the gate-array base **to ensure a minimum clock skew and minimum clock latency**—given power dissipation and clock buffer area limitations.
- Cell-based ASICs may use either a **clock spine, a clock tree, or a hybrid approach**.
- Figure shows how a clock router **may minimize clock skew in a clock spine** by making the path lengths, and thus net delays, to every leaf node equal—using jogs in the interconnect paths if necessary.
- More sophisticated clock routers perform
  - clock-tree synthesis (automatically choosing the depth and structure of the clock tree)
  - clock-buffer insertion (equalizing the delay to the leaf nodes by balancing interconnect delays and buffer delays).



(a)

(b)

Power Routing

- Each of the power buses has to be **sized according to the current it will carry.**
- Too much current in a power bus can lead to a failure through a mechanism known as **electromigration**
- The required **power-bus widths can be estimated**
  - automatically from library information, from a separate power simulation tool
  - by entering the power-bus widths to the routing software by hand
- Many routers use a **default power-bus width** so that it is **quite easy to complete routing** of an ASIC without even knowing about this problem.

Gate Array ASICs:

- Gate arrays normally use a **regular power grid** as part of the gate-array base.
- The gate-array logic cells **contain two fixed-width power buses** inside the cell, running horizontally on m1.
- The horizontal m1 power buses are then strapped in a vertical direction by m2 buses, which run vertically across the chip.

Cell-based ASIC

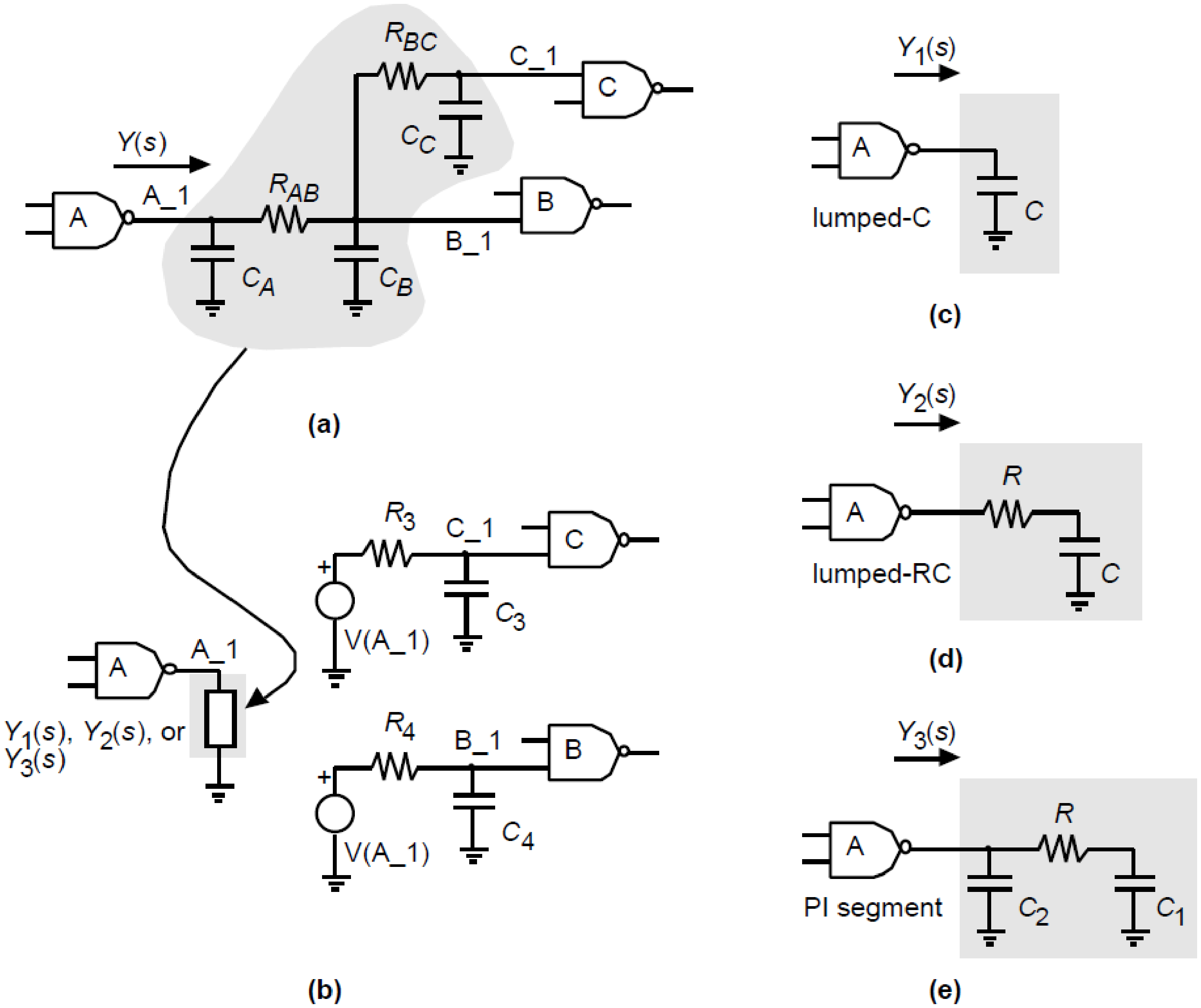
- Standard cells are constructed in a **similar fashion to gate-array cells**, with power buses running horizontally in m1 at the top and bottom of each cell.
- A row of standard cells uses **end-cap cells that connect to the VDD and VSS power buses** placed by the power router.
- Power routing of cell-based ASICs may **include the option to include vertical m2 straps at a specified intervals**.
- In a **three-level metal process**, power routing is similar to two-level metal ASICs.

Circuit Extraction and DRC

Circuit Extraction:

- After detailed routing is complete, **the exact length and position of each interconnect for every net is known.**
- Now the **parasitic capacitance and resistance** associated with each interconnect, via, and contact **can be calculated**.
- This data is generated by a **circuit-extraction tool** in one of the formats of SPF.
- **standard parasitic format (SPF):** describes interconnect delay and loading due to parasitic resistance and capacitance.
- There are three different forms of SPF:
  1. regular SPF
  2. reduced SPF
  3. detailed SPF
- **two of them** (regular SPF and reduced SPF) contain the same information, but in different formats, and **model the behavior of interconnect**; the **third form of SPF** (detailed SPF) **describes the actual parasitic resistance and capacitance components of a net.**

The load at the output of gate A is represented by **one of three models**: lumped-C, lumped- RC, or PI segment.



The regular and reduced standard parasitic format (SPF) models for interconnect.

(a) An example of an interconnect network with fanout. The driving-point admittance of the interconnect network is $Y(s)$.

(b) The SPF model of the interconnect.

(c) The lumped-capacitance interconnect model.

(d) The lumped-RC interconnect model.

(e) The PI segment interconnect model (notice the capacitor nearest the output node is labeled C_2 rather than C_1).

The values of C, R, C_1, and C_2 are calculated so that $Y_1(s)$, $Y_2(s)$, and $Y_3(s)$ are the first-, second-, and third-order Taylor-series approximations to $Y(s)$.

The key features of

Regular and reduced SPF are as follows:

- The **loading effect of a net** as seen by the driving gate is represented by choosing one of three different RC networks: lumped-C, lumped-RC, or PI segment (selected when generating the SPF)
- The **pin-to-pin delays of each path in the net** are modeled by a simple RC delay (one for each path). This can be the Elmore constant for each path.
- The **reduced SPF (RSPF) contains the same information as regular SPF, but uses the** SPICE format.

Detailed SPF:

- The **detailed SPF (DSPF) shows the resistance and capacitance of each segment in a** net, again in a SPICE format. There are **no models or assumptions** on calculating the net delays in this format.

Design Checks:

- ASIC designers **perform two major checks** before fabrication.
  1. design-rule check (DRC)
  2. layout versus schematic (LVS) check

Design-rule check (DRC):

- To ensure that nothing has gone wrong in the process of assembling the logic cells and routing.
- The DRC may be performed at two levels.
  - the **first level of DRC** is a **phantom-level DRC** , which checks for shorts, spacing violations, or other design-rule problems between logic cells.
    - This is principally a check of the detailed router.
  - If we have access to the **real library-cell layouts** (sometimes called hard layout), we can instantiate the phantom cells and perform a **second-level DRC** at the transistor level.
    - This is principally a check of the correctness of the library cells.

Layout Versus Schematic (LVS) check:

- To ensure that what is about to be committed to silicon is what is really wanted.
- An electrical schematic is extracted from the physical layout and compared to the netlist.
- This closes a loop between the logical and physical design processes and ensures that both are the same.