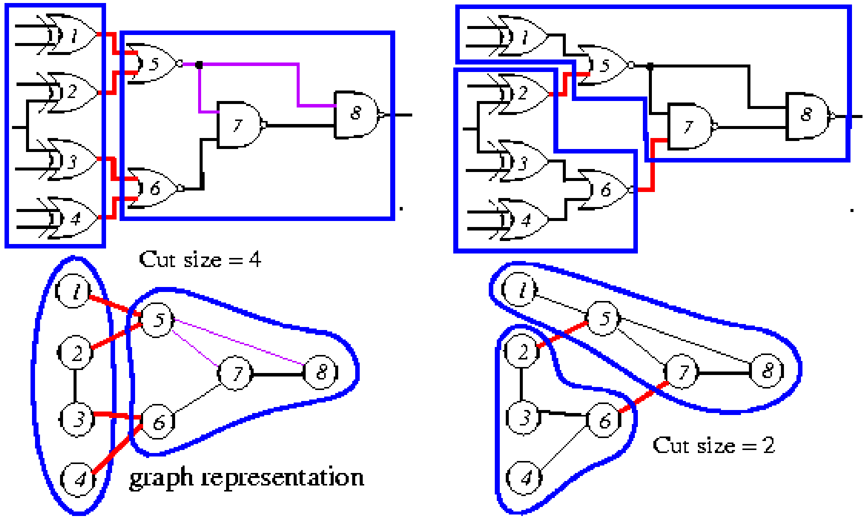
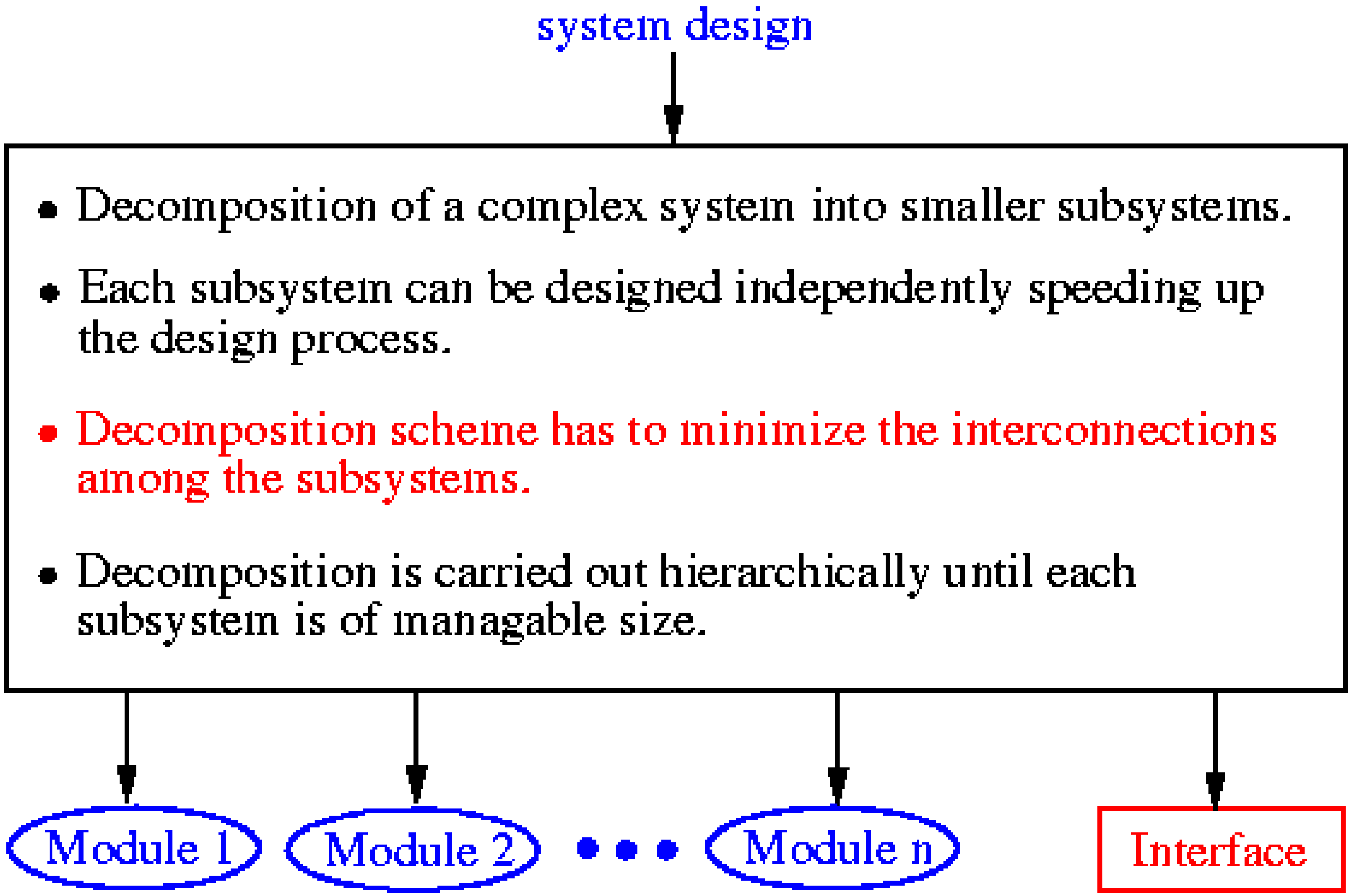
Unit 3: Partitioning

- Course contents
  - Kernighagn & Lin heuristic
  - Fiduccia-Mattheyses heuristic
  - Multilevel circuit partitioning
  - Exact net-weight modeling for wirelength minimization
- Appendices
  - Hypergraph cut cost modeling
  - Network-flow based method
  - Multi-2.5D FPGA-based system partitioning
- Readings
  - W&C&C: Chapter 11.3.1
  - S&Y: Chapter 2



Partitioning



system design

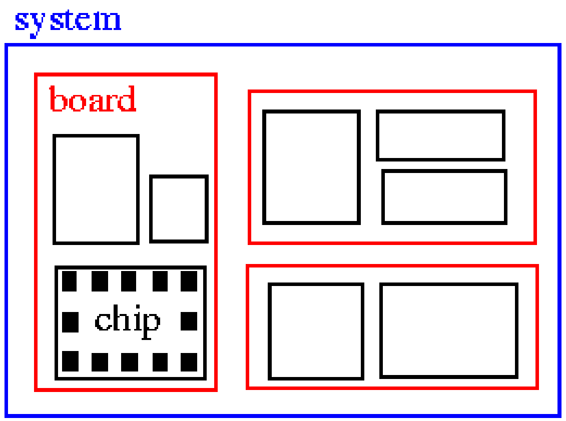
- Decomposition of a complex system into smaller subsystems.
- Each subsystem can be designed independently speeding up the design process.
- Decomposition scheme has to minimize the interconnections among the subsystems.
- Decomposition is carried out hierarchically until each subsystem is of managable size.

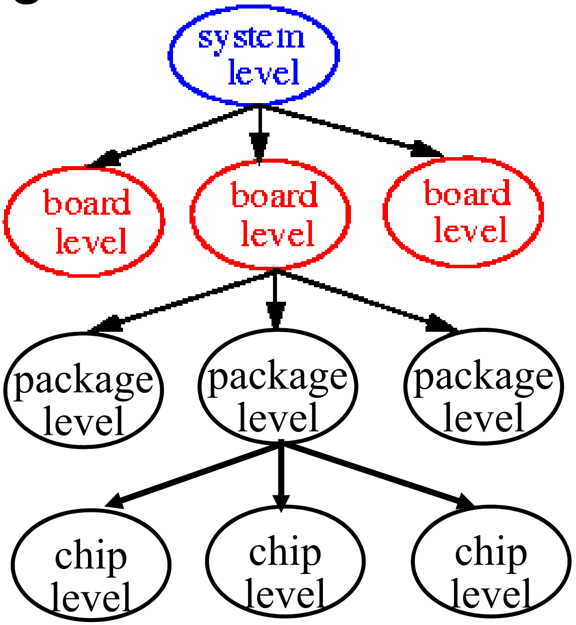
Module 1 Module 2 • • • Module n

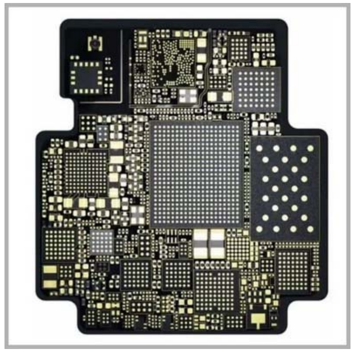
Interface

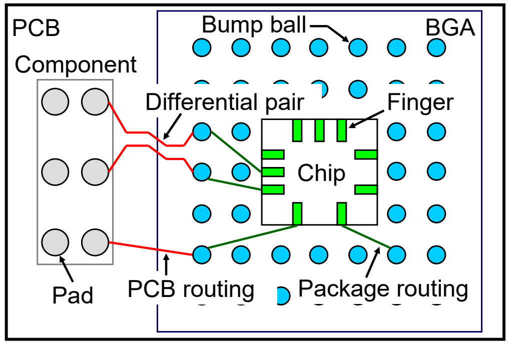
Levels of Partitioning

- Levels of partitioning: system, board, package, chip.
- Higher partitioning costs for higher levels.

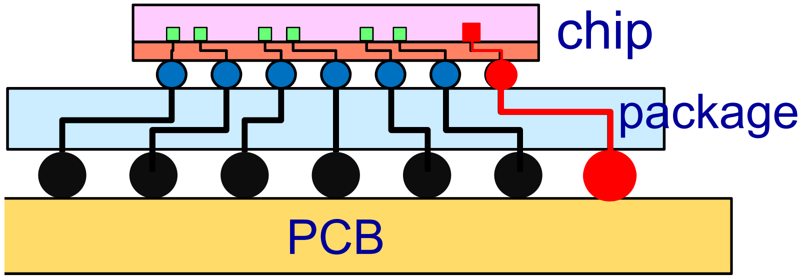






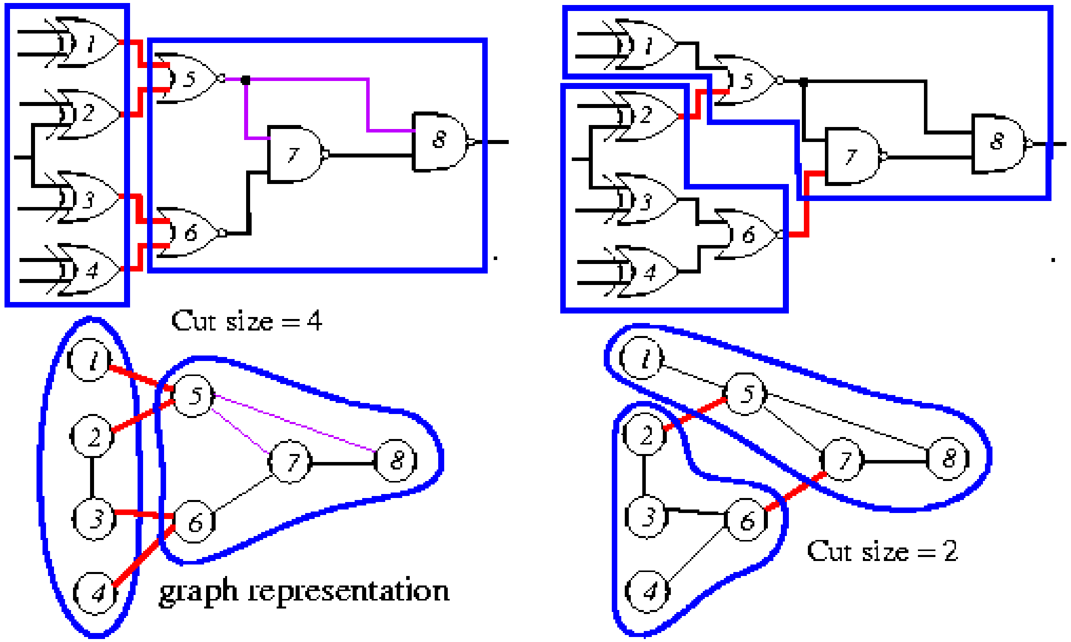


Board, package, chip co-design?
Partitioning cost?



Circuit Partitioning

- **Objective:** Partition a circuit into parts such that every component is within a prescribed range and the # of connections among the components is minimized.
  - **Two issues: cut size, balanced partitions**
  - More constraints are possible for some applications.
- Cutset? Cut size? Size of a component?



Partitioning Problem

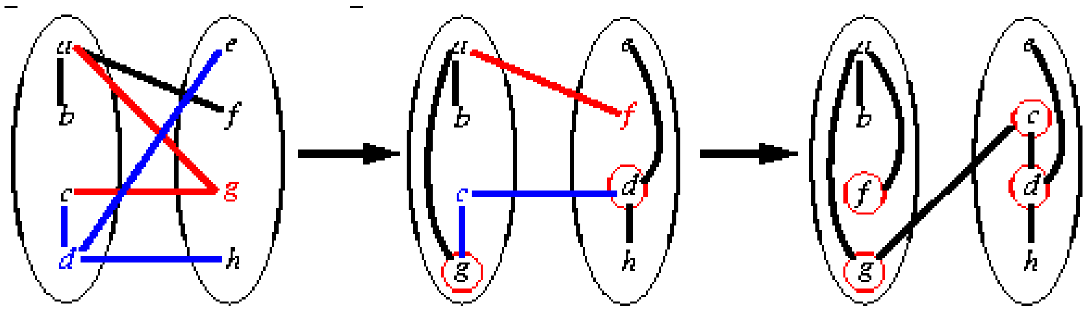
- ***k*-way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k **disjoint subsets** $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **Bounded size constraint:** The size of the *i*-th subset is bounded by B_i ($\sum_{v \in V_i} s(v) \leq B_i$).
  - Is the partition balanced?
- **Min-cut cost between two subsets:** Minimize $\sum_{\forall e=(u,v) \wedge p(u) \neq p(v)} w(e)$, where $p(u)$ is the partition # of node u.
- The 2-way, balanced partitioning problem is NP-complete, even in its simple form with identical vertex sizes and unit edge weights.
- **Key issues: min-cut cost & balanced partitions!!**

Kernighan-Lin Heuristic

- Kernighan and Lin, “An efficient heuristic procedure for partitioning graphs,” *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- An **iterative**, **2-way**, **balanced** partitioning (bi-sectioning) heuristic.
- Minimize cut cost while keeping partitions balanced
- **While** the cut size keeps decreasing
  - Vertex pairs which give the largest decrease **or the smallest increase** in cut size are exchanged.
  - These vertices are then **locked** (and thus are prohibited from participating in any further exchanges).
  - This process continues until all the vertices are locked.
  - Find the set with the **largest partial sum** for swapping.
  - Unlock all vertices.

Kernighan-Lin Heuristic: A Simple Example

- Each edge has a unit weight.



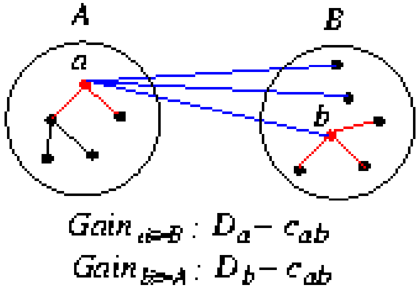
Step #	Vertex pair	Cost reduction	Cut cost
0	-	0	5
1	{d, g}	3	2
2	{c, f}	1	1
3	{b, h}	-2	3
4	{a, e}	-2	5

- Questions: How to compute cost reduction? What pairs to be swapped?
  - Consider the change of internal & external connections.

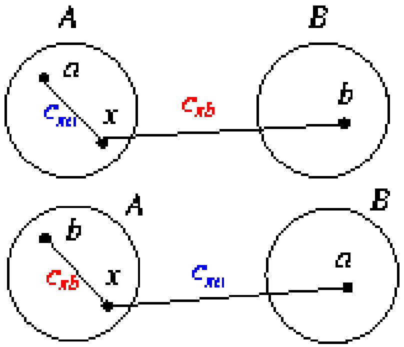
Properties

- Two sets *A* and *B* such that $|A| = n = |B|$ and $A \cap B = \emptyset$.
- **External cost** of $a \in A$: $E_a = \sum_{v \in B} c_{av}$.
- **Internal cost** of $a \in A$: $I_a = \sum_{v \in A} c_{av}$.
- *D*-value of a vertex *a*: $D_a = E_a - I_a$ (cost reduction for moving *a*).
- Cost reduction (gain) for swapping *a* and *b*: $g_{ab} = D_a + D_b - 2c_{ab}$.
- If $a \in$ A and $b \in B$ are interchanged, then the new *D*-values, *D*', are given by

$$\begin{aligned} D'_x &= D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\} \\ D'_y &= D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\}. \end{aligned}$$



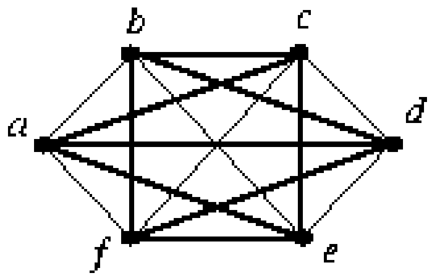
Internal cost vs. External cost

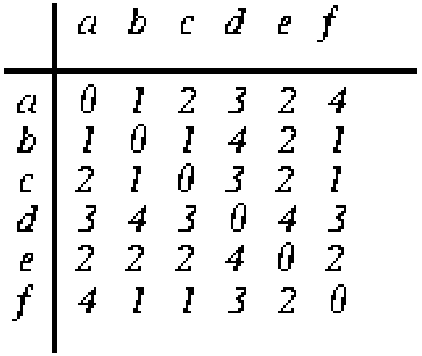


before swap	after swap	$\triangle C$
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

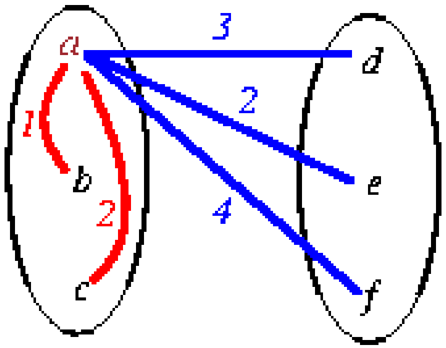
updating D−values

Kernighan-Lin Heuristic: A Weighted Example





	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



costs associated with a

Initial cut cost = (3+2+4)+(4+2+1)+(3+2+1) = 22

- 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-Value Computation

- Iteration 1:

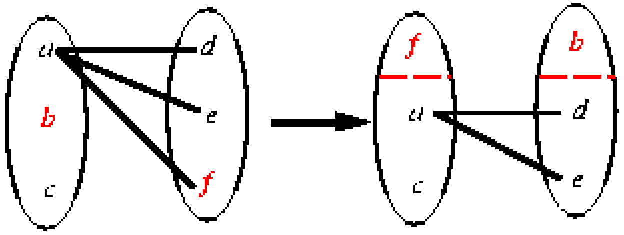
$$
\begin{array}{lll}
I_a = 1 + 2 = 3; & E_a = 3 + 2 + 4 = 9; & D_a = E_a - I_a = 9 - 3 = 6 \\
I_b = 1 + 1 = 2; & E_b = 4 + 2 + 1 = 7; & D_b = E_b - I_b = 7 - 2 = 5 \\
I_c = 2 + 1 = 3; & E_c = 3 + 2 + 1 = 6; & D_c = E_c - I_c = 6 - 3 = 3 \\
I_d = 4 + 3 = 7; & E_d = 3 + 4 + 3 = 10; & D_d = E_d - I_d =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

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

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

- Swap *b* and *f* and lock them! $(\hat{g}_1 = 4)$

D-Value Computation



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

$$\begin{aligned} D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0 \\ D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3 \end{aligned}$$

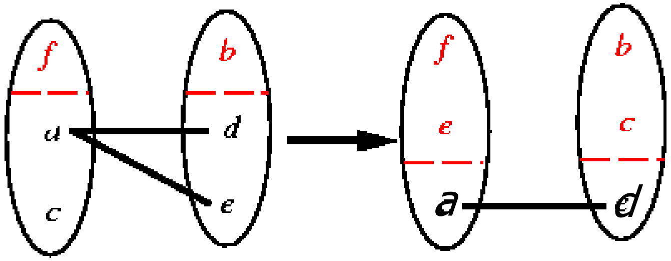
$$\boxed{\begin{aligned} D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1 \\ D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0 \end{aligned}}$$

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

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

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

Swapping Pair Determination



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

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

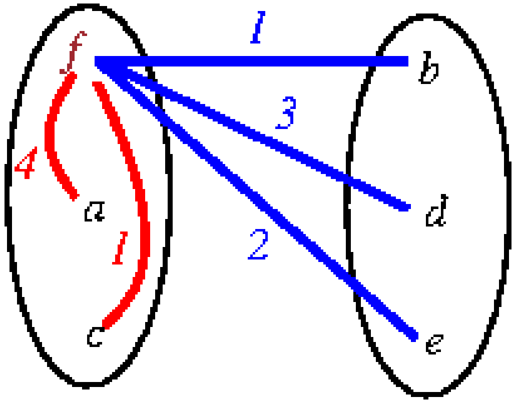
- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

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

- 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*.

Next Iteration

	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



Initial cut cost = (1+3+2)+(1+3+2)+(1+3+2) = 18 (22−4)

- Iteration 2: Repeat what we did at Iteration 1 (Initial cost = 22-4 =18).
- Summary: $\hat{g_1} = g_{ce} = -1,\ \hat{g_2} = g_{ab} = -3,\ \hat{g_3} = g_{fd} = 4.$
- Largest partial sum = $\max \sum_{i=1}^{k} \hat{g_i} = 0$ (*k*=3) ⇒ Stop!

Kernighan-Lin Heuristic

```
Algorithm: Kernighan-Lin($G$)
Input: $G = (V, E)$, $|V| = 2n$.
Output: Balanced bi-partition $A$ and $B$ with "small" cut cost.
begin
Bipartition $G$ into $A$ and $B$ such that $|V_A| = |V_B|$, $V_A \cap V_B = \emptyset$,
  and $V_A \cup V_B = V$;
repeat
    Compute $D_v$, $\forall\, v \in V$;
    for $i$ =1 to $n$ do
        Find a pair of unlocked vertices $v_{ai} \in V_A$ and $v_{bi} \in V_B$ whose
        exchange makes the largest decrease or smallest increase in cut
        cost;
        Mark $v_{ai}$ and $v_{bi}$ as locked, store the gain $\widehat{g}_i$ , and compute the new
        $D_v$, for all unlocked $v \in V$;
    Find $k$, such that $G_k = \sum_{i=1}^{k} \widehat{g}_i$ is maximized;
    if $G_k > 0$ then
        Move $v_{a1}, \ldots, v_{ak}$ from $V_A$ to $V_B$ and $v_{b1}, \ldots, v_{bk}$ from $V_B$ to $V_A$;
    Unlock $v$, $\forall\, v \in V$;
until $G_k \leq 0$;
end
```

Time Complexity

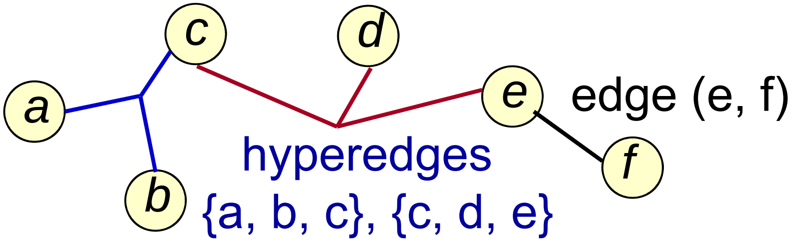
- Line 4: Initial computation of D: $O(n^2)$
- Line 5: The **for**-loop: $O(n)$
- The body of the loop: $O(n^2)$.
  - Lines 6--7: Step i takes $(n-i+1)^2$ time.
- Lines 4--11: Each pass of the repeat loop: $O(n^3)$.
- Suppose the repeat loop terminates after r passes.
- The total running time: $O(rn^3)$.
  - Polynomial-time algorithm?
- k-way partitioning
  1. Partition the graph into k equal-sized sets.
  2. Apply the Kernighan-Lin algorithm for each pair of subsets.
  3. Time complexity? Can be reduced by recursive bi-partitioning

Drawbacks of the Kernighan-Lin Heuristic

- The K-L heuristic **handles only unit vertex weights**.
  - Vertex weights might represent block sizes, different from blocks to blocks.
  - Reducing a vertex with weight $w(v)$ into a clique with $w(v)$ vertices and edges with a high cost increases the size of the graph substantially.
- The K-L heuristic **handles only exact bisections**.
  - Need dummy vertices to handle the unbalanced problem.
- The K-L heuristic **cannot handle hypergraphs.**
  - Need to handle multi-terminal nets directly.
- The **time complexity of a pass is high**, $O(n^3)$.

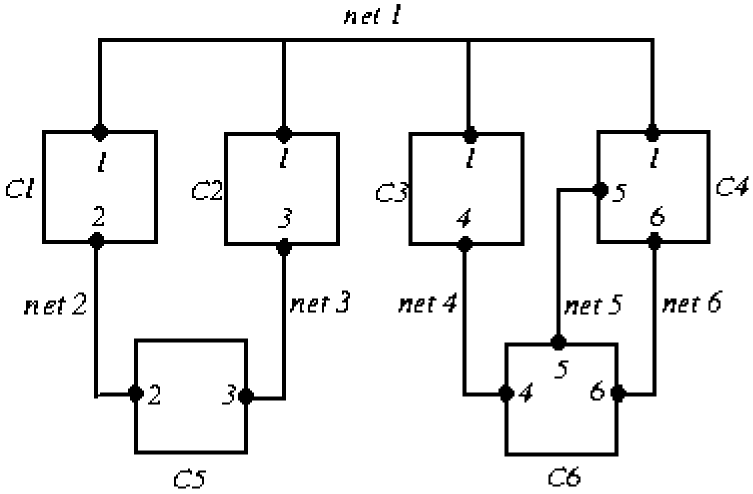
Fiduccia-Mattheyses Heuristic

- Fiduccia and Mattheyses, "A linear time heuristic for improving network partitions," DAC-82.
- New features to the K-L heuristic:
  - Aims at **reducing net-cut costs**; the concept of cutsize is extended to hypergraphs.
  - Only a **single vertex is moved** across the cut each time.
  - Vertices are weighted.
  - Can handle "unbalanced" partitions; a balance factor is introduced.
  - A special data structure is used to select vertices to be moved across the cut to improve running time.
  - **Time complexity $O(P)$,** where P is the total # of terminals.
- Hypergraph $H = (N, L)$ consists of a set N of vertices and a set L of hyperedges, where each hyperedge is a **subset** N_i of distinct vertices with $|N_i| \geq 2$



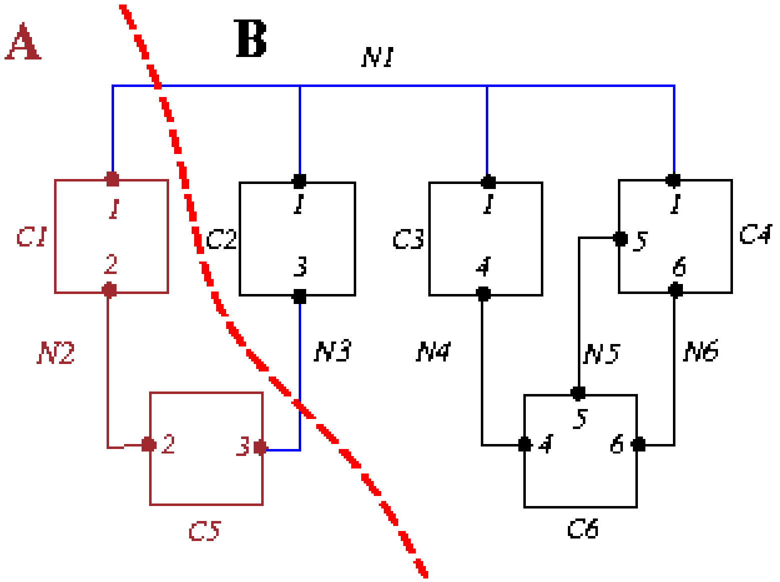
F-M Heuristic: Notation

- $n(i)$: # of cells in Net i; e.g., $n(1) = 4$.
- $s(i)$: size of Cell i.
- $p(i)$: # of pin terminals in Cell i; e.g., $p(6)=3$.
- C: total # of cells; e.g., $C=6$.
- N: total # of nets; e.g., $N=6$.
- P: total # of pins; $P = p(1) + \dots + p(C) = n(1) + \dots + n(N)$.

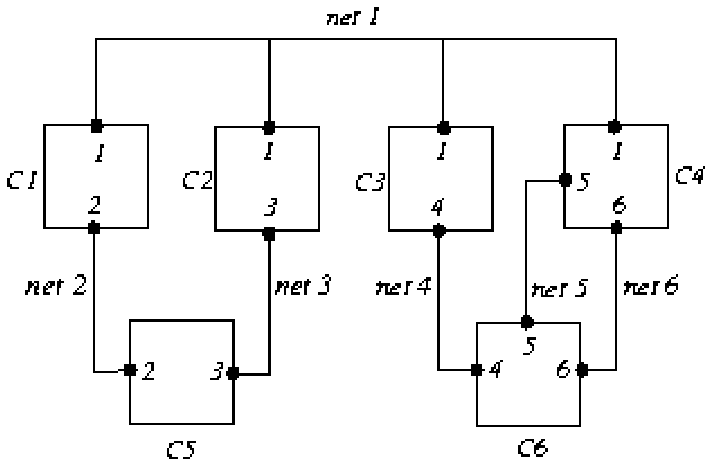


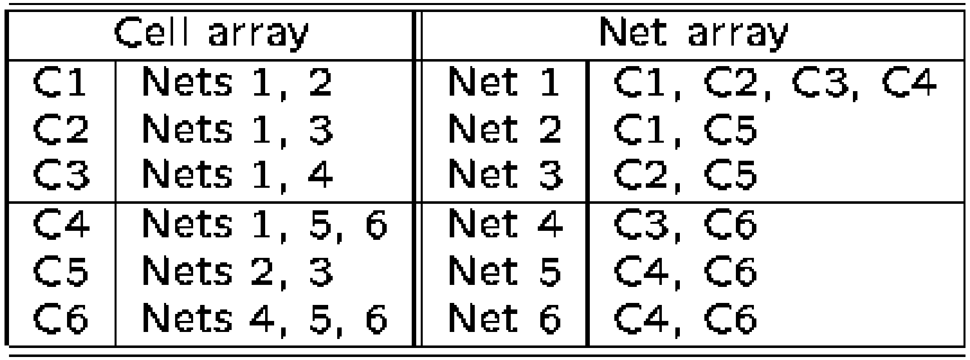
Cut

- **Cutstate** of a net:
  - Net 1 and Net 3 are **cut** by the partition.
  - Net 2, Net 4, Net 5, and Net 6 are **uncut**.
- **Cutset** = {Net 1, Net 3}.
- $|A|$ = size of A = $s(1)+s(5)$; $|B| = s(2)+s(3)+s(4)+s(6)$.
- **Balanced 2-way partition:** Given a fraction r, $0 < r < 1$, partition a graph into two sets A and B such that
  - $\frac{|A|}{|A|+|B|} \approx r$.
  - Size of the cutset is minimized.



Input Data Structures





Cell array		Net array	
C1	Nets 1, 2	Net 1	C1, C2, C3, C4
C2	Nets 1, 3	Net 2	C1, C5
C3	Nets 1, 4	Net 3	C2, C5
C4	Nets 1, 5, 6	Net 4	C3, C6
C5	Nets 2, 3	Net 5	C4, C6
C6	Nets 4, 5, 6	Net 6	C4, C6

- Size of the network: $P = \sum_{i=1}^{6} n(i) = 14$
- Construction of the two arrays takes *O*(*P*) time.

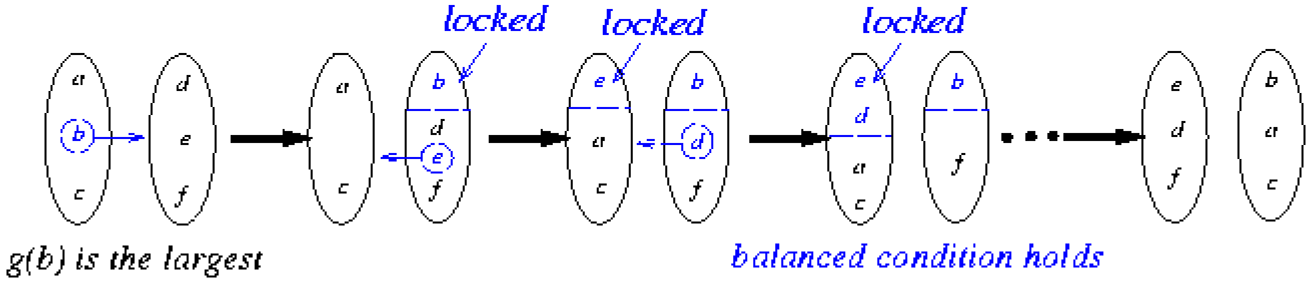
Basic Ideas: Balance and Movement

- Only move a cell at a time, preserving "balance."

$$\begin{aligned} \frac{|A|}{|A|+|B|} &\approx r \\ rW - S_{max} &\leq |A| \leq rW + S_{max}, \end{aligned}$$

where $W=|A|+|B|$; $S_{\max}=\max_i s(i)$.

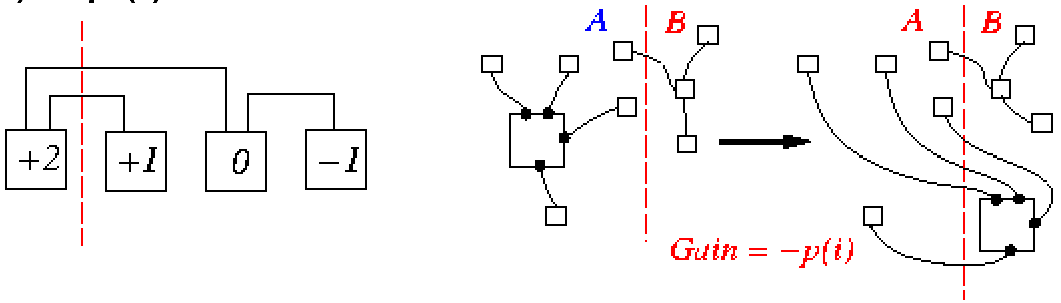
- $g(i)$: gain in moving cell i to the other set, i.e., size of **old** cutset - size of **new** cutset.



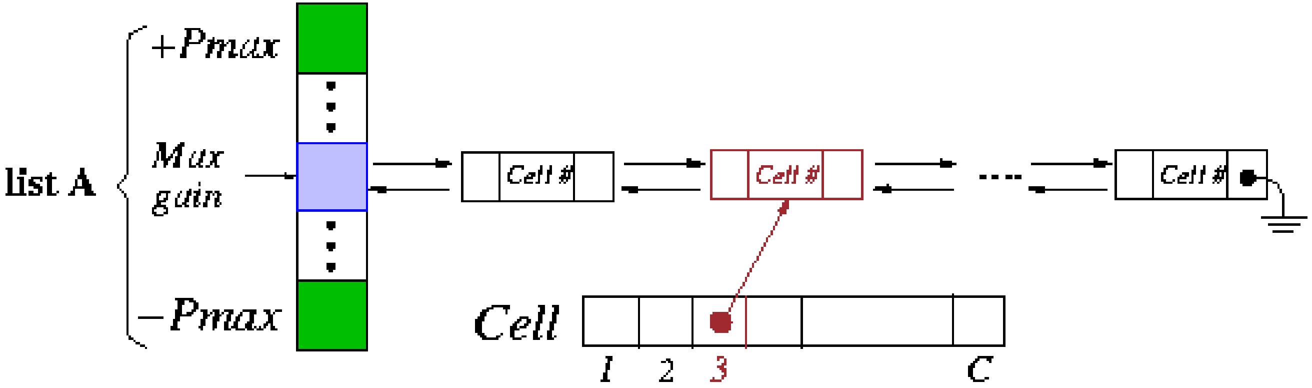
- Suppose $\hat{g}_i$'s: $g(b)$, $g(e)$, $g(d)$, $g(a)$, $g(f)$, $g(c)$ and the largest partial sum is $g(b)+g(e)+g(d)$. Then we should move b, e, d two resulting sets: {a, c, e, d}, {b, f}.

Cell Gains and Data Structure Manipulation

- $-p(i) \leq g(i) \leq p(i)$



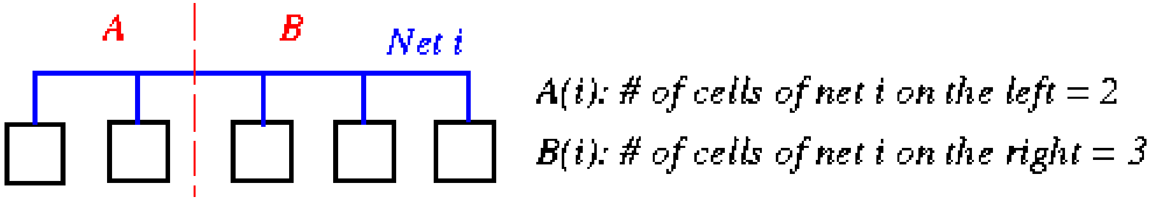
- Two “bucket list” structures, one for set *A* and one for set *B* (P_{max} = $\max_i p(i)$).



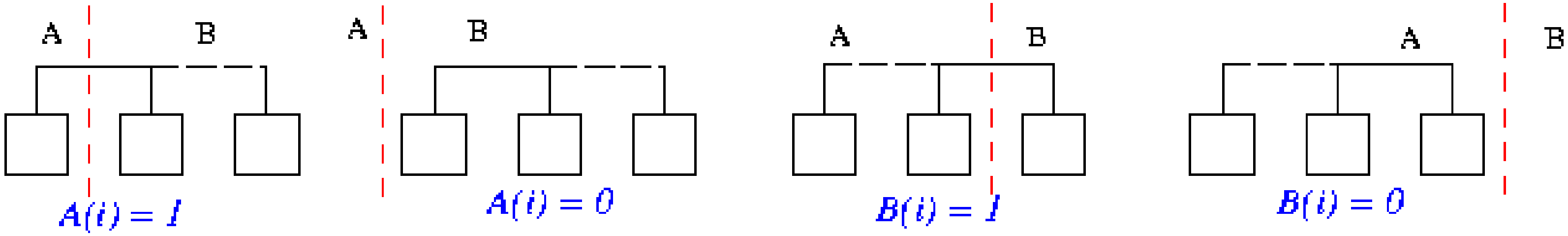
- **O(1)-time operations:** find a cell with Max Gain, remove Cell *i* from the structure, insert Cell *i* into the structure, update $g(i)$ to $g(i)+\Delta$, and update the Max Gain pointer.

Net Distribution and Critical Nets

- Distribution of Net *i*: (*A*(*i*), *B*(*i*)) = (2, 3).
  - (*A*(*i*), *B*(*i*)) for all *i* can be computed in *O*(*P*) time.


A
B
Net i
A(i): # of cells of net i on the left = 2
B(i): # of cells of net i on the right = 3

- **Critical Nets**: A net is critical if it has a cell which if moved will change its cutstate.
  - 4 cases: *A*(*i*) = 0 or 1, *B*(*i*) = 0 or 1.

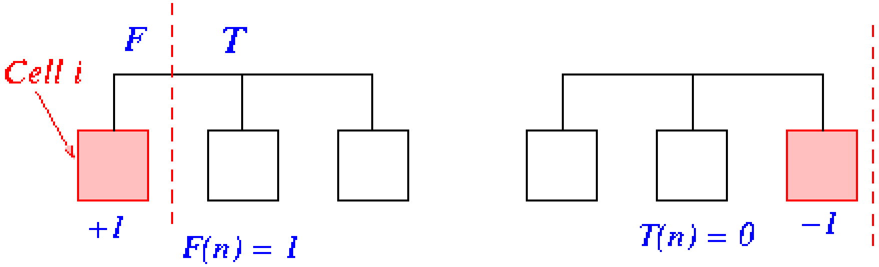


- **Gain of a cell depends only on its critical nets.**

Computing Cell Gains

- Initialization of all cell gains requires $O(P)$ time:

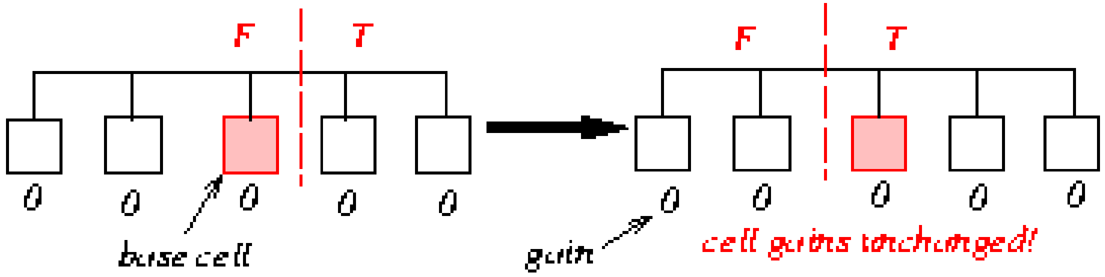
```
$g(i) \leftarrow 0$;
$F \leftarrow$ the "from block" of Cell $i$;
$T \leftarrow$ the "to block" of Cell $i$;
for each net $n$ on Cell $i$ do
    if $F(n) = 1$ then $g(i) \leftarrow g(i) + 1$;
    if $T(n) = 0$ then $g(i) \leftarrow g(i) - 1$;
```



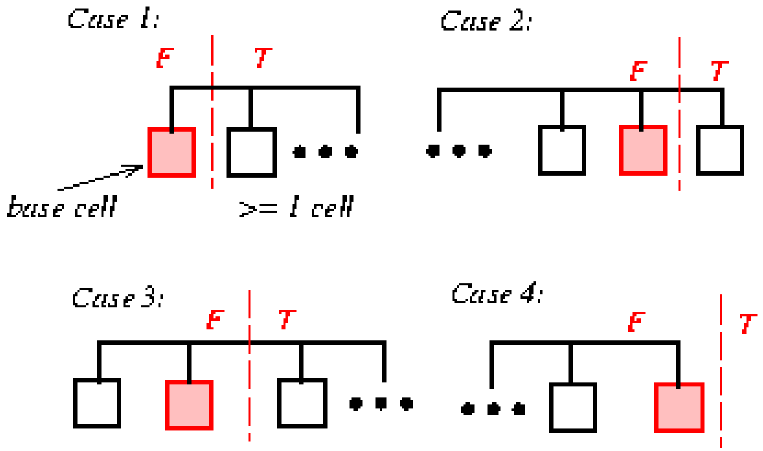
- Will show: Only need $O(P)$ time to maintain all cell gains in one pass.

Updating Cell Gains

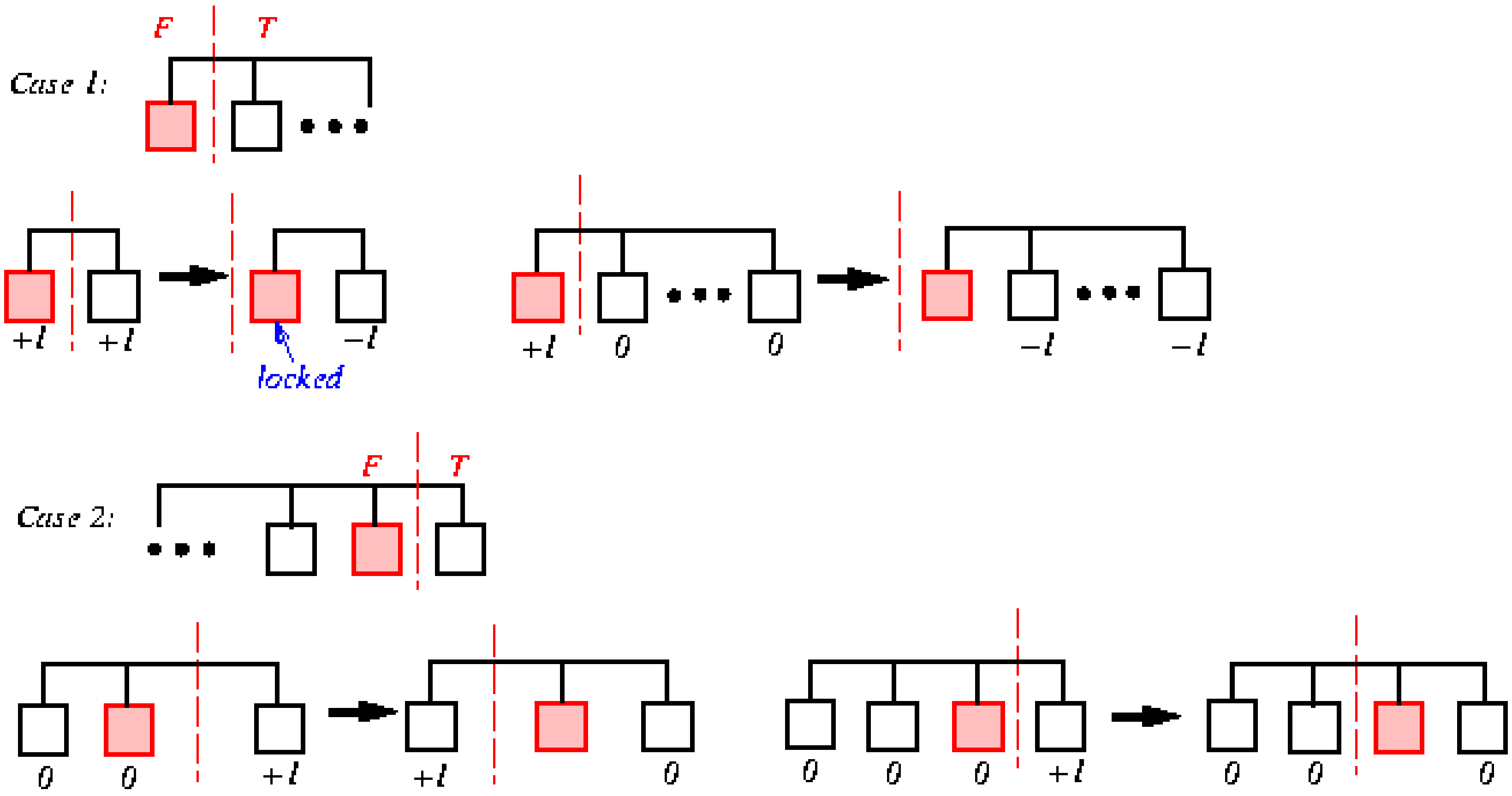
- To update the gains, we only need to look at those nets, connected to the base cell, which are critical **before** or **after** the move.
- **Base cell**: The cell selected for movement from one set to the other.



- Consider only the case where the base cell is in the left partition. The other case is similar.

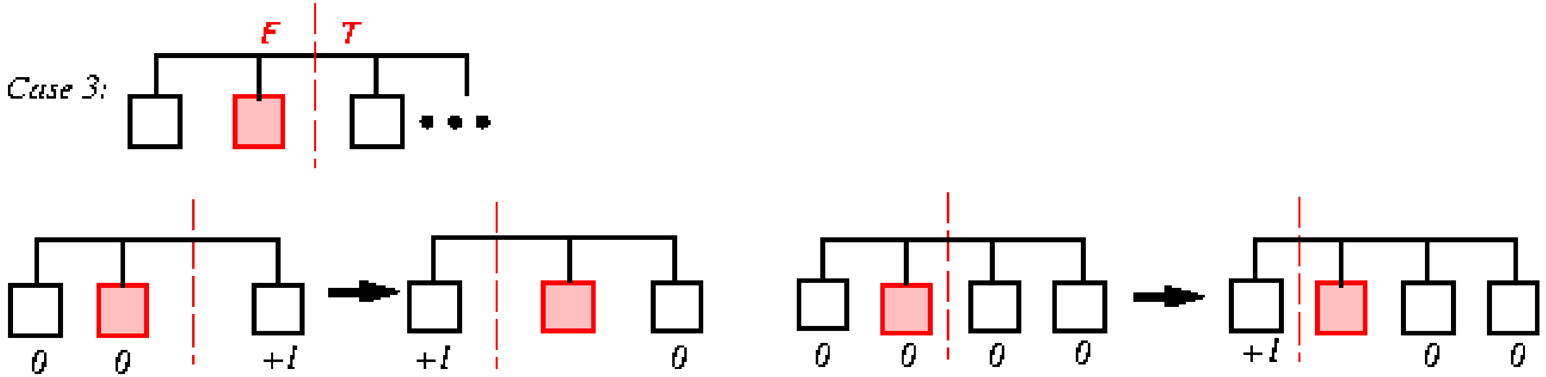


Updating Cell Gains (cont'd)

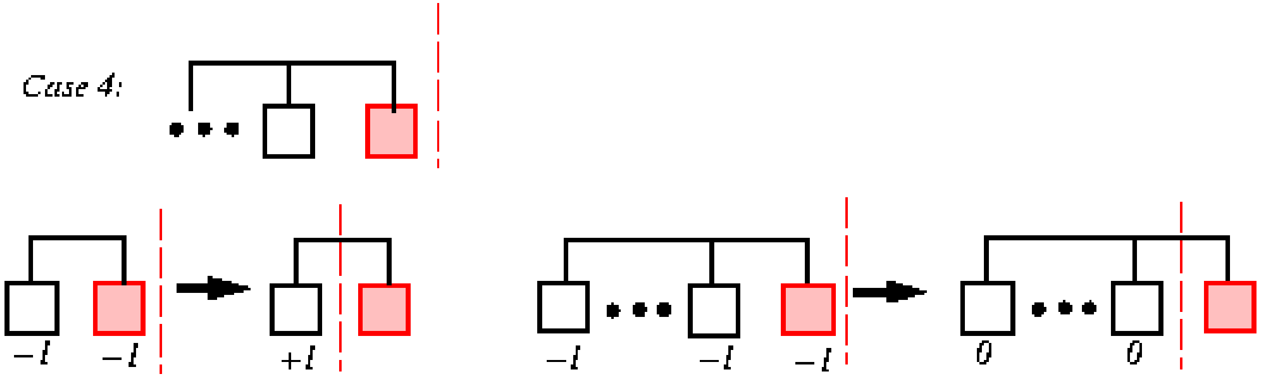


Updating Cell Gains (cont'd)

Case 3:

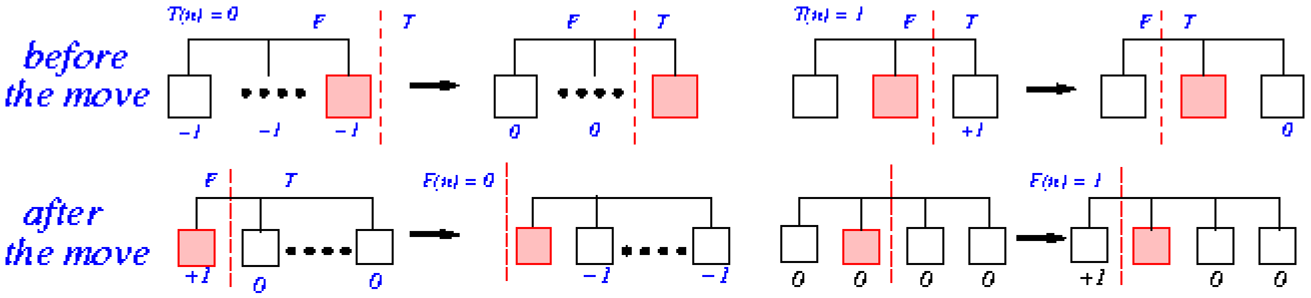


Case 4:



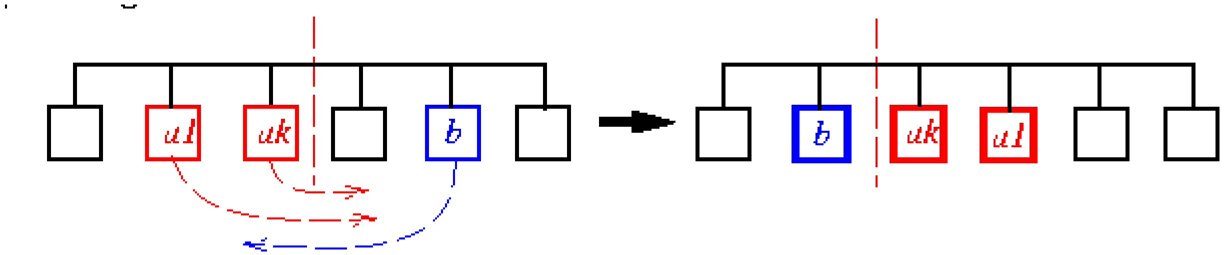
Algorithm for Updating Cell Gains

```
Algorithm: Update_Gain
begin  /* move base cells and update neighbors' gains */
F ← the From Block of the base cell;
T ← the To Block of the base cell;
Lock the base cell and complement its block;
for each net n on the base cell do
  /* check critical nets before the move */
    if T(n) = 0 then increment gains of all free cells on n
    else if T(n) = 1 then decrement gain of the only T cell on n,
    if it is free
     /* change F(n) and T(n) to reflect the move */
     F(n) ← F(n) - 1; T(n) ← T(n) + 1;
    /* check for critical nets after the move */
    if F(n) = 0  then decrement gains of all free cells on n
    else if F(n) = 1 then increment gain of the only F cell on n,
    if it is free
end
```



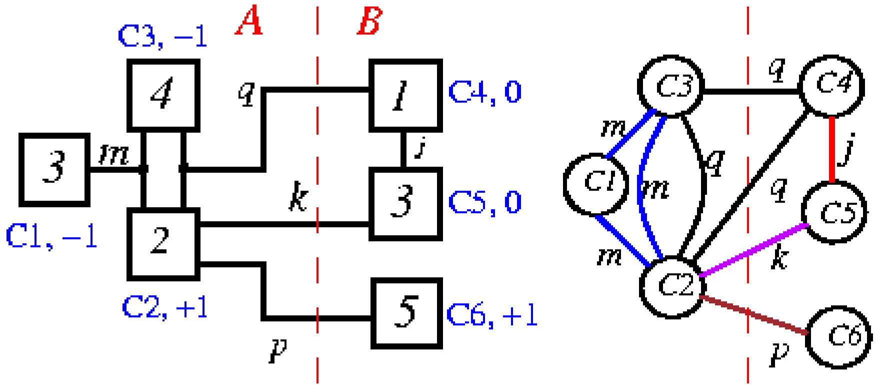
Complexity of Updating Cell Gains

- Once a net has “locked” cells at both sides, the net will remain cut from now on.
- Suppose we move a_1, a_2, …, a_k from left to right, and then move *b* from right to left At most only moving a_1, a_2, …, a_k and *b* need updating!



- To update the cell gains, it takes $O(n(i))$ work for Net *i*.
- Total time = $n(1)+n(2)+\ldots+n(N) = O(P)$.

F-M Heuristic: An Example

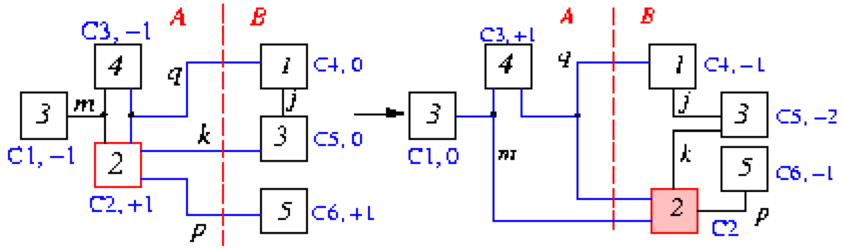


- Computing cell gains: $F(n) = 1$ $g(i) + 1$; $T(n) = 0$ $g(i) - 1$

Cell	m		q		k		p		j		$g(i)$
	F	T	F	T	F	T	F	T	F	T	
c1	0	−1									−1
c2	0	−1	0	0	+1	0	+1	0			+1
c3	0	−1	0	0							−1
c4			+1	0					0	−1	0
c5					+1	0			0	−1	0
c6							+1	0			+1

- Balanced criterion: $r|V| - S_{max} \leq |A| \leq r|V| + S_{max}$. Let $r = 0.4$ $|A| = 9$, $|V| = 18$, $S_{max} = 5$, $r|V|=7.2$ Balanced: $2.2 \leq 9 \leq 12.2$!
- maximum gain: c_2 and balanced: $2.2 \leq 9-2 \leq 12.2$ Move c_2 from A to B (use size criterion if there is a tie).

F-M Heuristic: An Example (cont'd)



- Changes in net distribution:

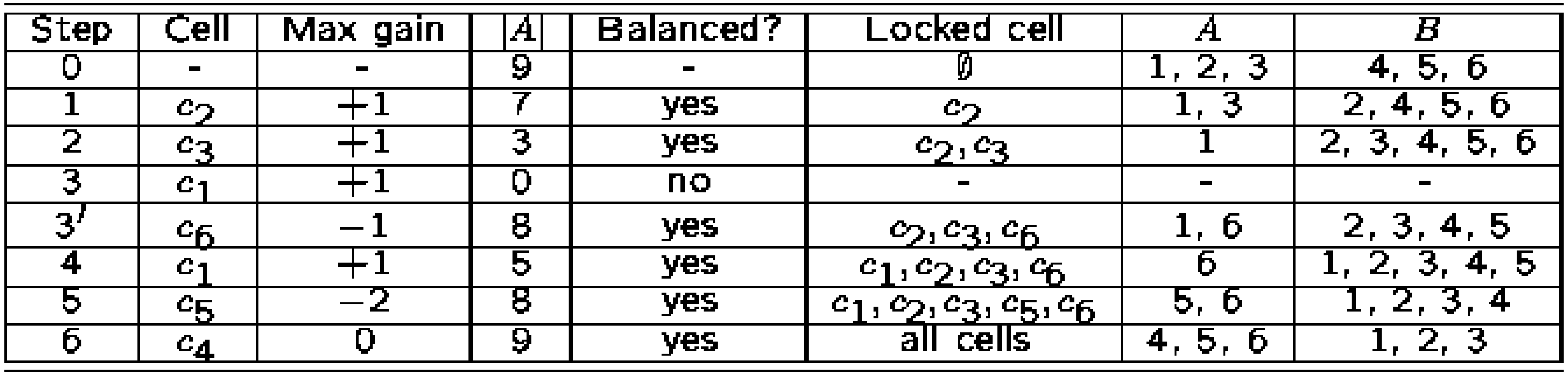
Net	Before move		After move	
	F	T	F'	T'
k	1	1	0	2
m	3	0	2	1
q	2	1	1	2
p	1	1	0	2

- Updating cell gains on critical nets (run Algorithm Update_Gain):

Cells	Gains due to $T(n)$				Gain due to $F(n)$				Gain changes	
	k	m	q	p	k	m	q	p	Old	New
c_1		$+1$							-1	0
c_3		$+1$					$+1$		-1	$+1$
c_4			-1						0	-1
c_5	-1				-1				0	-2
c_6				-1				-1	$+1$	-1

- Maximum gain: c_3 and balanced! ($2.2 \leq 7\text{-}4 \leq 12.2$) → Move c_3 from *A* to *B* (use size criterion if there is a tie).

Summary of the Example

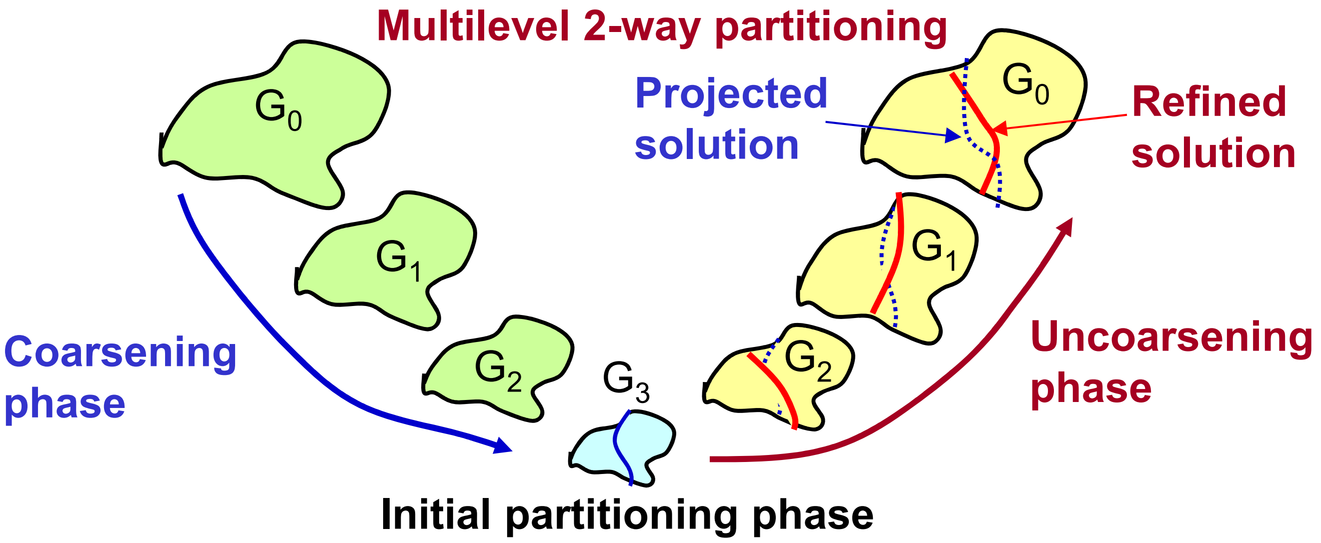


Step	Cell	Max gain	$\lvert A \rvert$	Balanced?	Locked cell	A	B
0	-	-	9	-	$\emptyset$	1, 2, 3	4, 5, 6
1	c_2	$+1$	7	yes	c_2	1, 3	2, 4, 5, 6
2	c_3	$+1$	3	yes	c_2, c_3	1	2, 3, 4, 5, 6
3	c_1	$+1$	0	no	-	-	-
$3'$	c_6	-1	8	yes	c_2, c_3, c_6	1, 6	2, 3, 4, 5
4	c_1	$+1$	5	yes	c_1, c_2, c_3, c_6	6	1, 2, 3, 4, 5
5	c_5	-2	8	yes	c_1, c_2, c_3, c_5, c_6	5, 6	1, 2, 3, 4
6	c_4	0	9	yes	all cells	4, 5, 6	1, 2, 3

- $\hat{g}_1 = 1, \hat{g}_2 = 1, \hat{g}_3 = -1, \hat{g}_4 = 1, \hat{g}_5 = -2, \hat{g}_6 = 0$ Maximum partial sum G_k = +2, k = 2 or 4.
- Since k=4 results in a better balanced partition Move c_1, c_2, c_3, c_6 A={6}, B={1, 2, 3, 4, 5}.
- **Repeat the whole process until new $G_k \leq 0$.**

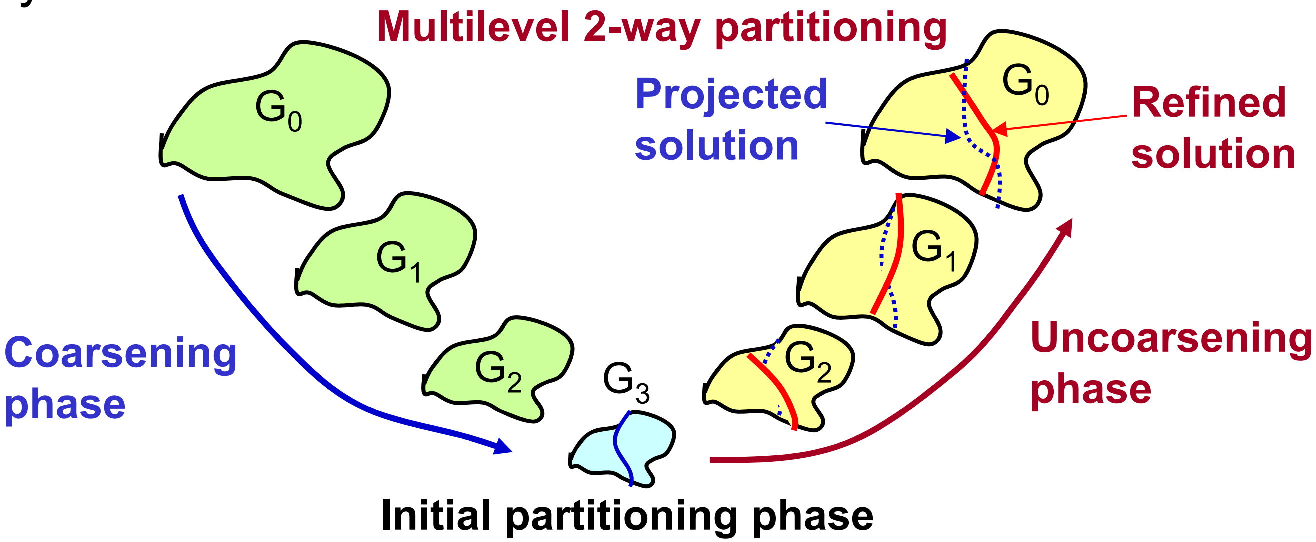
Large-scale Circuit Partitioning

- Keys for large-scale circuits: **clustering, multilevel framework.**
- **Clustering:** Reduce the problem size by grouping highly connected components and treat them as a super node.
- **Multilevel partitioning**
  - **Coarsening:** Recursively clusters the instance until its size is smaller than a given threshold.
  - **Uncoarsening:** Declusters the instance while applying a partitioning refinement algorithm (e.g., F-M).



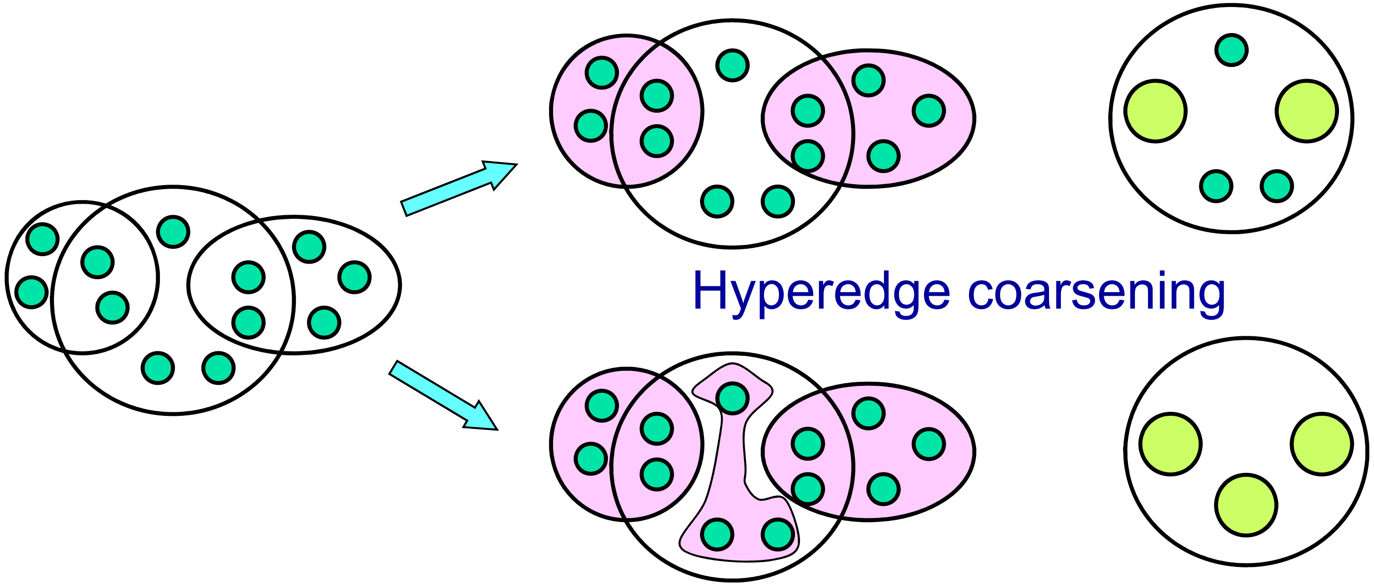
hMetis: Multilevel 2-way Partitioner

- Karypis *et al*., “Multilevel hypergraph partitioning: application in VLSI domain,” DAC-97.
- **Coarsening:** Recursively groups together vertices based on some connectivity metrics (each vertex is highly connected with at least one other vertex in the group) until the number of vertices is less than *ck* (say, *c* =100, *k* = 2).
- **Initial partitioning:** Balanced random bisection (could also apply F-M to obtain an initial partitioning of the coarsest hypergraph).
- **Uncoarsening:** Declusters the instance while applying a partitioning refinement algorithm (e.g., FM) to improve the quality level by level.



Hyperedge Coarsening

- **Hyperedge coarsening:** An independent set of hyperedges is selected and the vertices that belong to these hyperedges are contracted together.
  - Give preference to the hyperedges with larger weights and smaller sizes.
- **Modified hyperedge coarsening:** After the hyperedge coarsening, the vertices of each uncontracted hyperedge are matched to be contracted together.



Hyperedge coarsening

Modified hyperedge coarsening

Refinement

- **Early-exit FM:** Repeatedly move vertices between partitions to improve the cut by early-exit FM
  - Limit the max # of passes to only two.
  - Stop each pass after performing *p* vertex moves that did not improve the cut

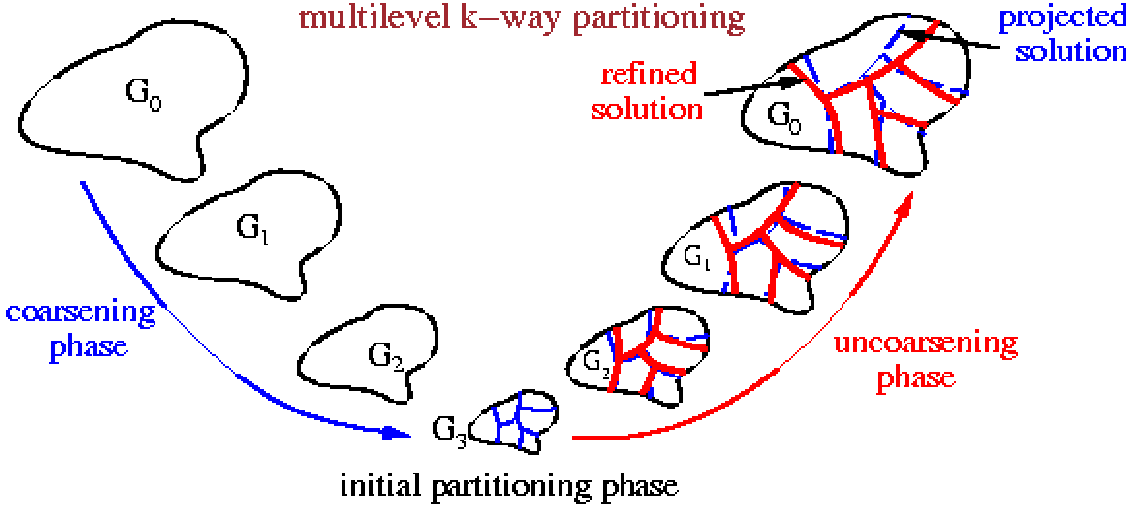
- **Hyperedge Refinement:** Move groups of vertices between two partitions so that an entire hyperedge is moved from the cut

- Empirically, early-exit FM performs slightly better than hyperedge refinement by about 1-2% in cut size, but needs 50% longer running time.

- My note: Due to the computer performance improvement, it should be feasible nowadays to perform full FM during the refinement to further improve the solution quality.

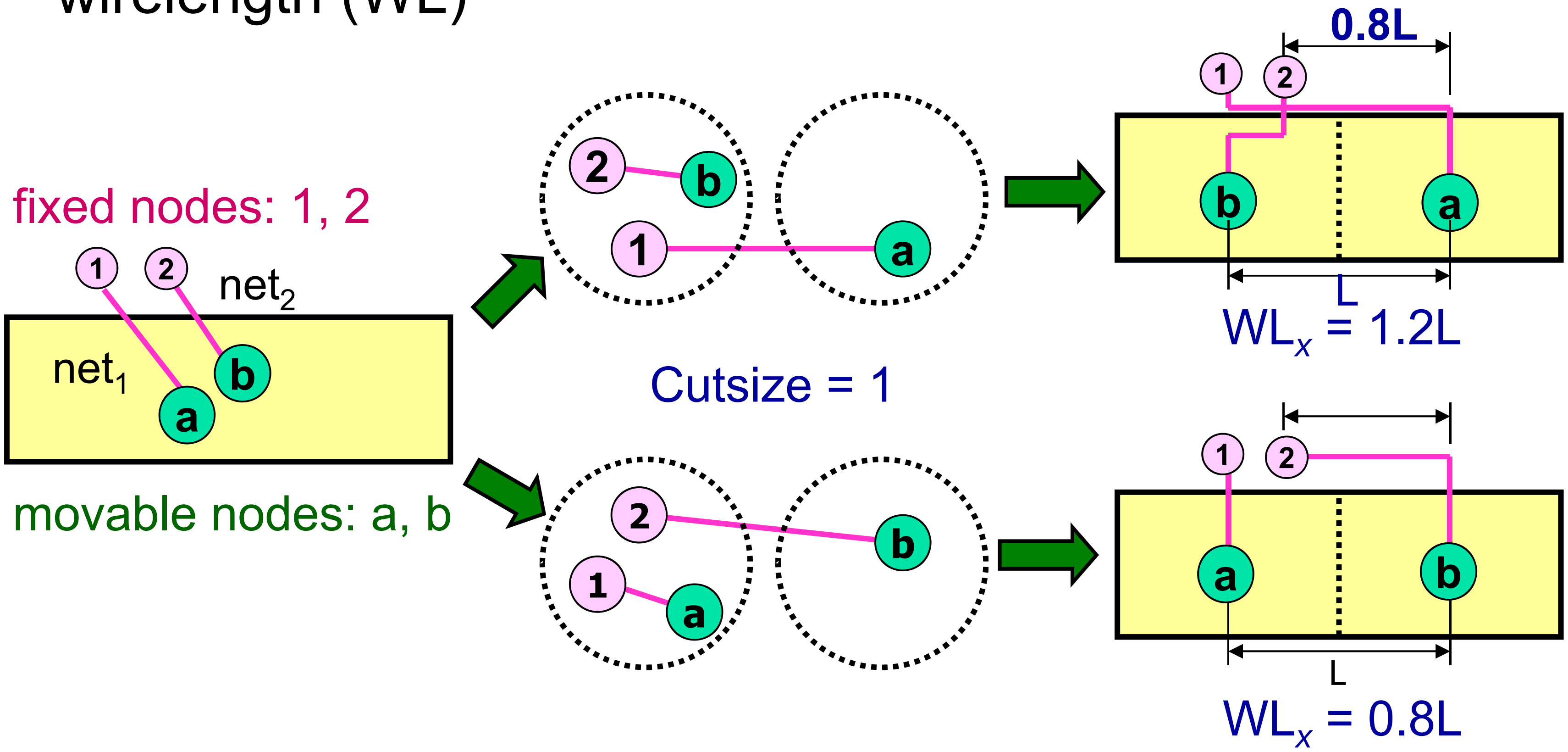
Multilevel *k*-way Partitioning

- Karypis & Kumar, "Multilevel *k*-way hypergraph partitioning," DAC-99.
- **Coarsening:** Recursively groups together vertices (each vertex is highly connected with at least one other vertex in the group) until the number of vertices is less than *ck* (say, *c* =100).
- **Initial partitioning:** Compute a k-way partitioning of the coarsest hypergraph (e.g., by a multilevel bisection algorithm) s.t. the balance constraint is satisfied and the objective is optimized.
- **Uncoarsening:** Declusters the instance while applying an iterative greedy refinement algorithm (those vertices at the boundary of a partition are moved if the moves result in better solutions).



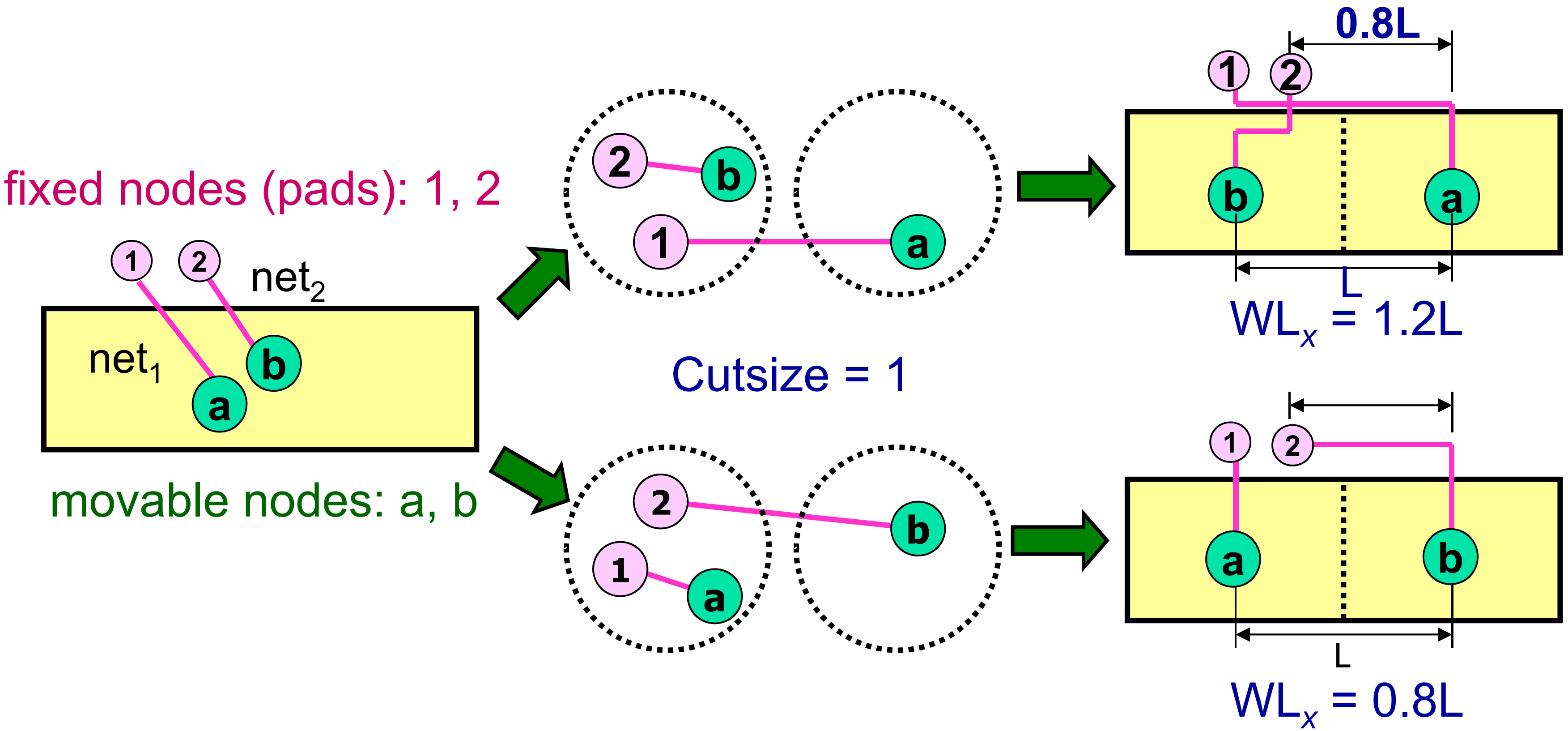
Partitioning for Direct Wirelength Minimization

- Chen, Chang, Lin, "IMF: Interconnect-driven floorplanning for large-scale building-module designs," ICCAD-05
- Minimizing cut size is ***not*** equivalent to minimizing wirelength (WL)



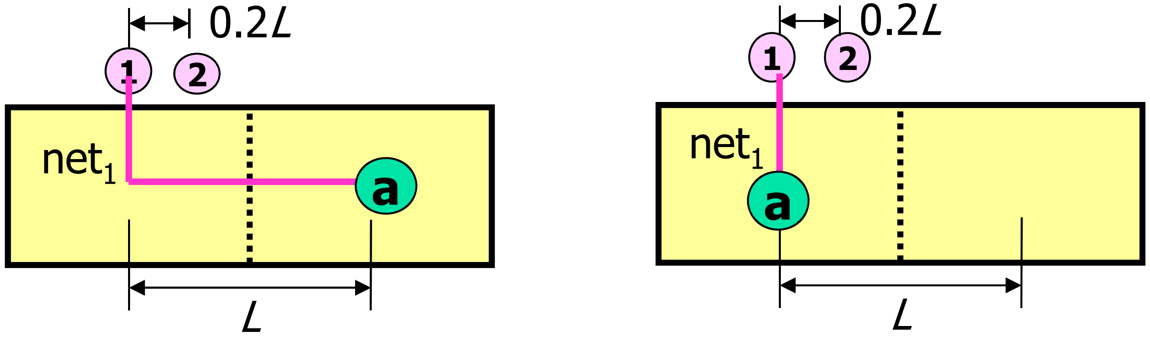
Problem with Min-Cut

- Problem: hyperedge weight is a **constant** value!
  - Shall map the min-cut cost to wirelength (WL) change
  - Shall assign the hyperedge weight as the value of the wirelength contribution if the hyperedge is cut

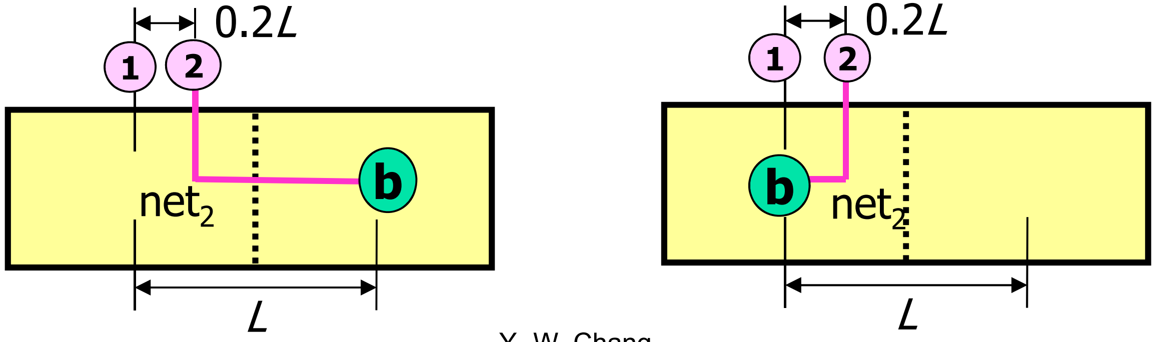


Net Weight Assignment

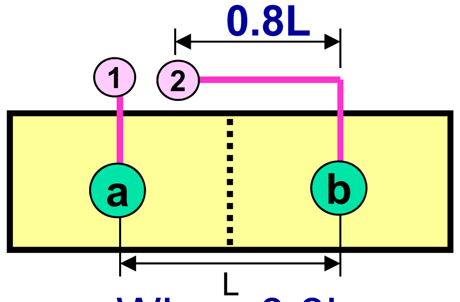
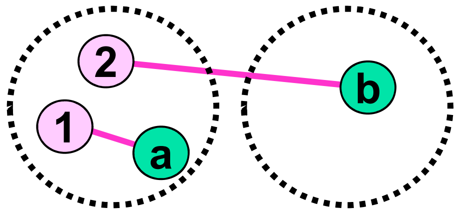
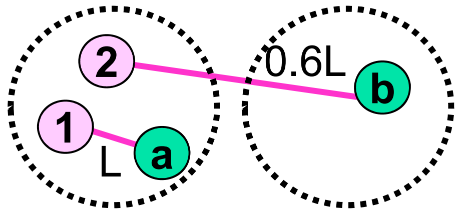
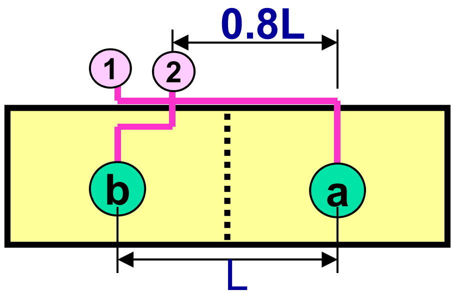
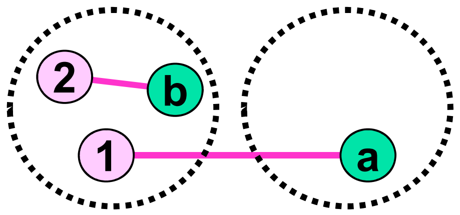
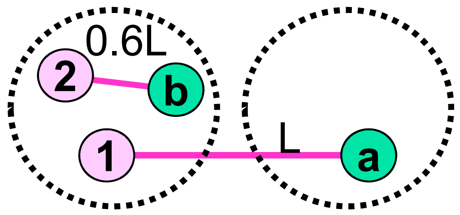
- net_1 connects a movable node ***a*** and a fixed node **1**.
  Weight(net_1) = WL(net_1 is cut) – WL(net_1 is not cut)
  = L – 0L = L



- net_2 connects a movable node ***b*** and a fixed node **2**.
  Weight(net_2) = WL(net_2 is cut) – WL(net_2 is not cut)
  = 0.8L – 0.2L = 0.6L



Model Comparisons

Physical Partitions	Traditional Terminal Propagation	Exact Net-Weight Modeling
 WL_x = 0.8L	 Cutsize = 1	 Cut weight = 0.6L
 WL_x = 1.2L	 Cutsize = 1	 Cut weight = 1.0L

Cut weight is proportional to the wirelength (WL)!

WL = Cut weight + 0.2L

(0.2L is the WL lower bound: placing a & b in the left side)

Relationship Between WL and Cut Weight

- Theorem: $\mathrm{WL}_i = w_{1,i} + n_{\mathrm{cut},i}$
  - $n_{\mathrm{cut},i}$: cut weight for net i
  - $w_{1,i}$: the wirelength lower bound for net i

- Then, we have

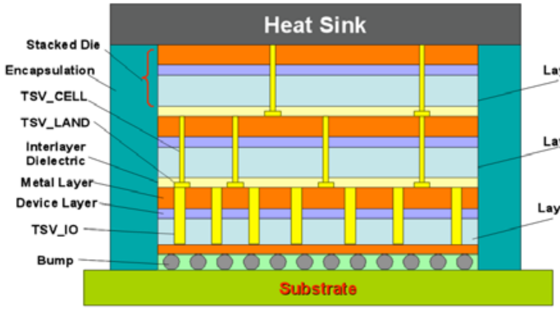
Constant!

$$\min\left(\sum WL_i\right) = \min\left(\sum (w_{1,i} + n_{cut,i})\right) = \sum w_{1,i} + \min\left(\sum n_{cut,i}\right)$$

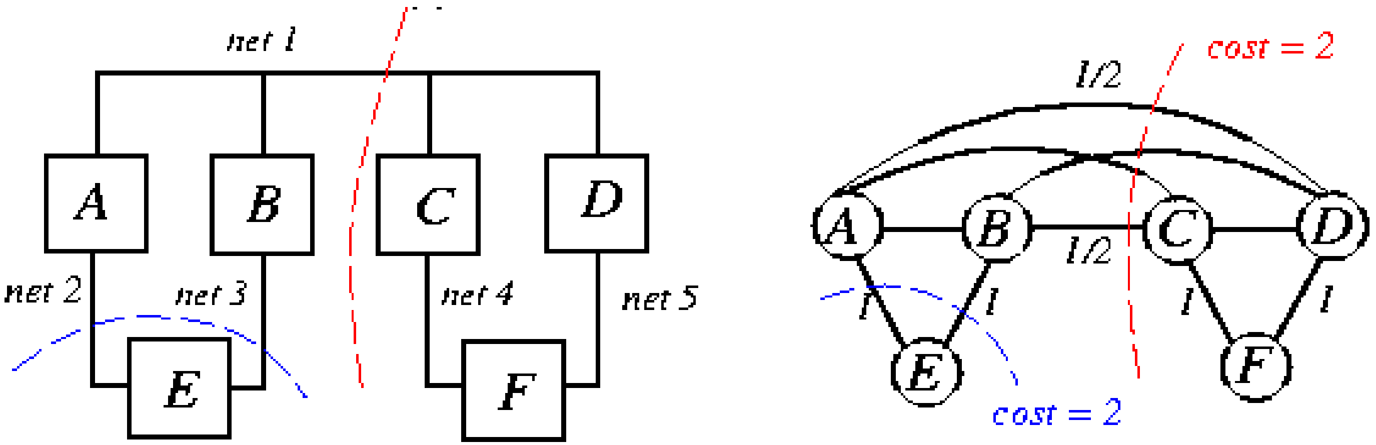
Finding the minimum wirelength is equivalent to finding the minimum cut weight!!

Summary: Partitioning

- Discussed methods: **group migration** (K-L, F-M) and **multilevel partitioning (hMetis)**
  - hMetis is almost “good enough” (Cong, et al, ISPD-03)
- Other important partitioning approaches
  - **Network-flow method:** Yang and Wong, ICCAD-94, ICCAD-95
  - **Spectral method:** Barnes, *SIAM JADM*; Boppana, FOCS-87; Alpert & Kahng, DAC-95, DAC-96, etc.
  - **Probability:** Dutt & Deng, DAC-96; Chao, et al., ICCAD-99
  - **Mathematical programming:** Quadratic programming (Shih & Kuh, DAC-93); ILP (Wu et al., TCAD, 2001)
  - **Unified approach:** Network flow + Spectral, Li, et al., ICCAD-95
- Clustering: Cong, et al., ICCAD-97; Chao, et al., ICCAD-99
- **Multi-FPGA pin assignment & partitioning**
  - Kuo, et al., ISPD-18; Chen et al., ICCAD-18
- Cost model for partitioning for 3D (2.5D) IC’s? TSV cost? Thermal issue?

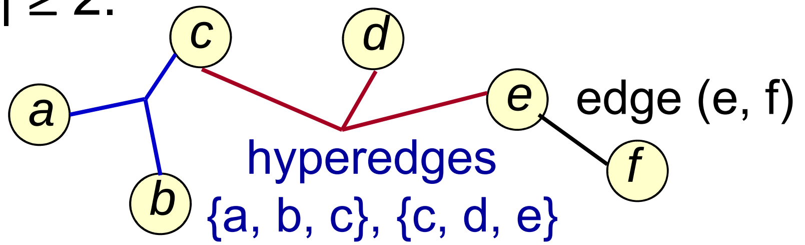


Appendix A:
Hypergraph Cut Cost Modeling



Coping with Hypergraph

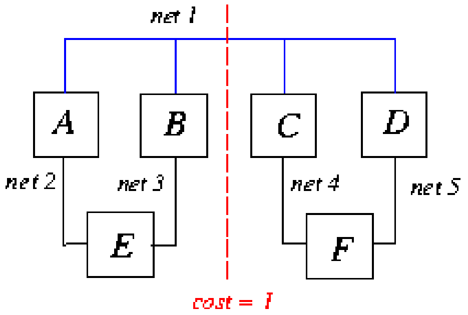
- A hypergraph H=(N, L) consists of a set N of vertices and a set L of hyperedges, where each hyperedge corresponds to a **subset** N_i of distinct vertices with $|N_i| \geq 2$.

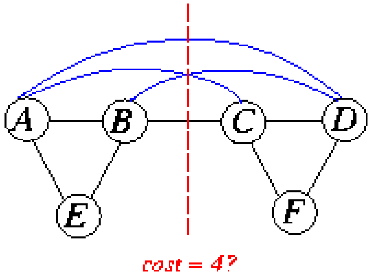


edge (e, f)

hyperedges {a, b, c}, {c, d, e}

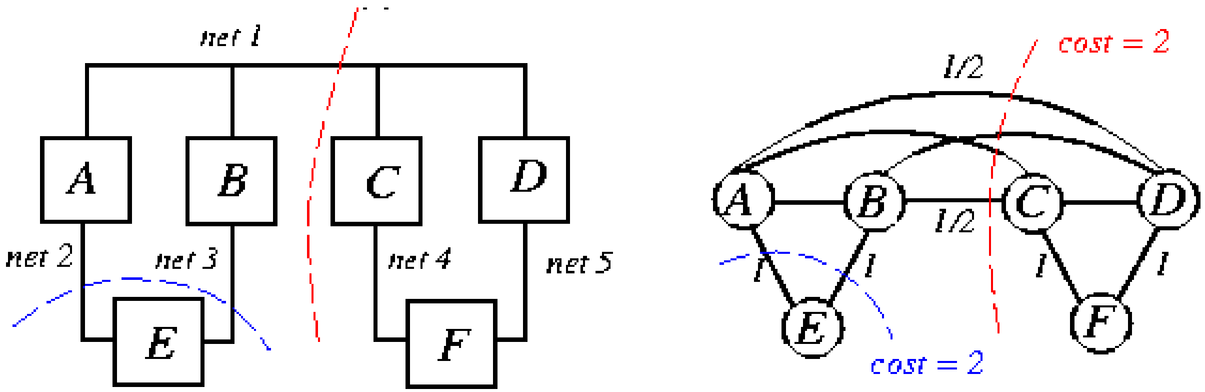
- Schweikert and Kernighan, “A proper model for the partitioning of electrical circuits,” 9th Design Automation Workshop, 1972.
- For multi-terminal nets, **net cut** is a more accurate measurement for cut cost (i.e., deal with hyperedges).
  - {A, B, E}, {C, D, F} is a good partition.
  - Should not assign the same weight for all edges.



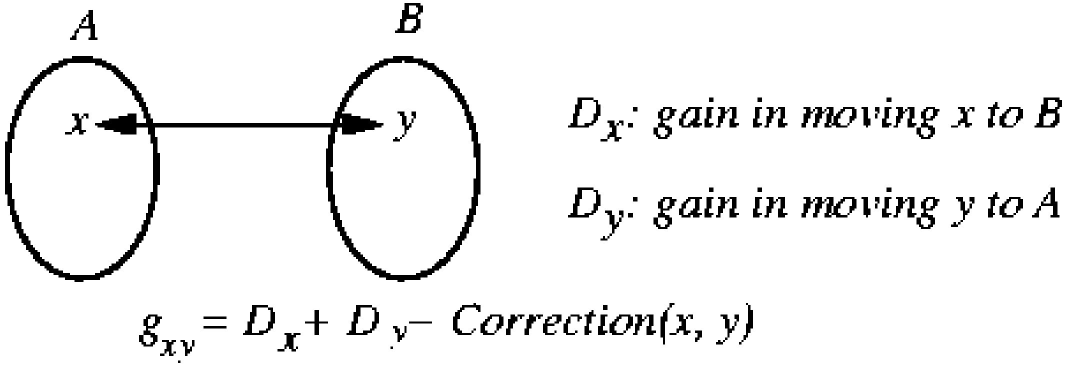


Net-Cut Model

- Let $n(i)$ = # of cells associated with Net i.
- Edge weight $w_{xy} = \frac{2}{n(i)}$ for an edge connecting cells x and y.



- Easy modification of the K-L heuristic.



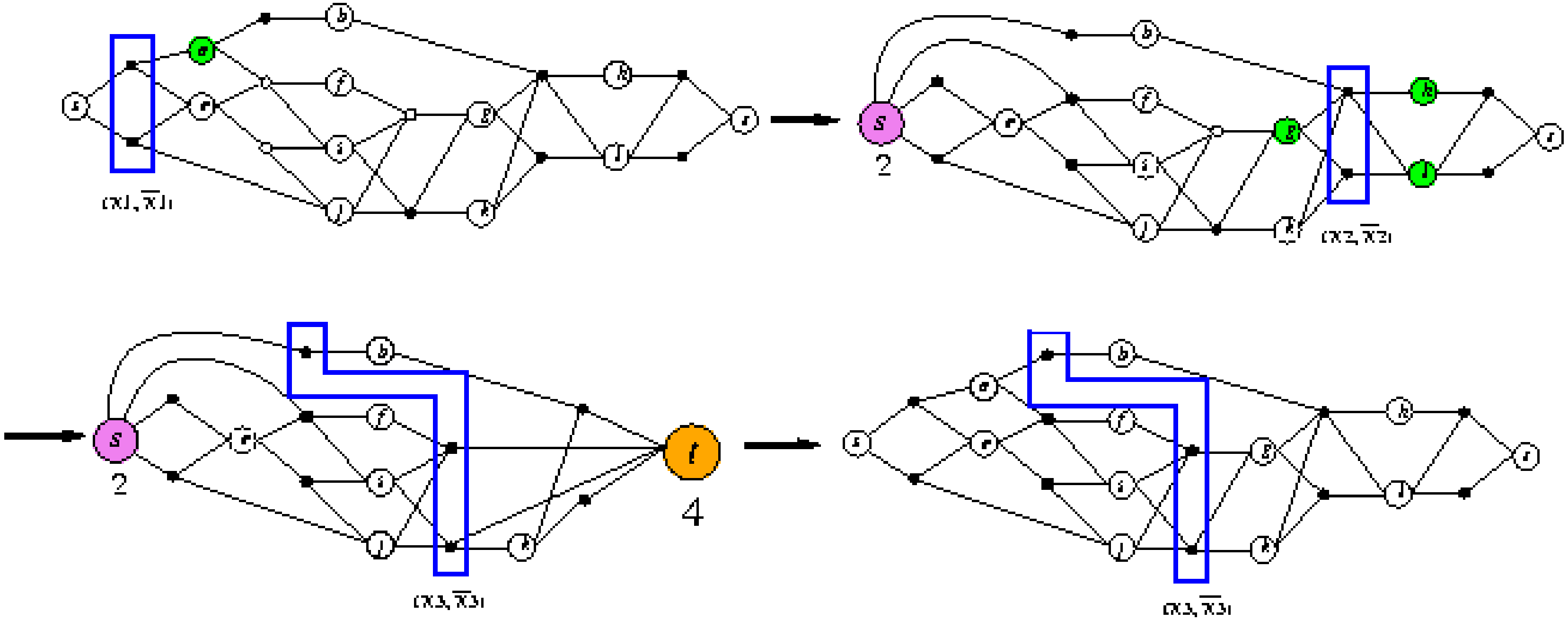
D_x: gain in moving x to B

D_y: gain in moving y to A

$$g_{xy} = D_x + D_y - Correction(x, y)$$

Appendix B:

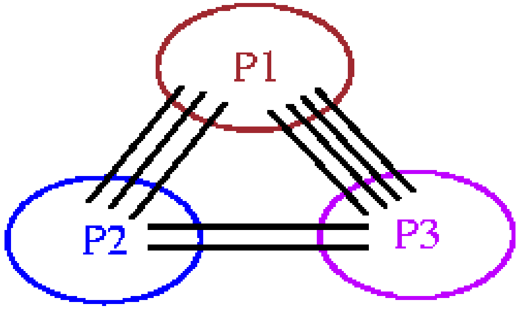
Network Flow Based Partitioning



○ An un–saturated net ● A saturated net 🟢 A node to be collapsed to s or t

Network Flow Based Partitioning

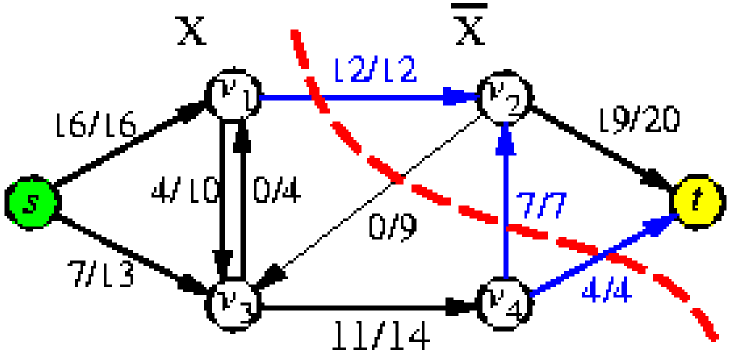
- Yang and Wong, "Efficient network-flow based min-cut balanced partitioning," ICCAD-94.
  - Based on max-flow min-cut theorem.



- Gate replication for partitioning: Yang and Wong, ICCAD-95.
- Performance-driven multiple-chip partitioning: Yang and Wong, FPGA-94, ED&TC-95.
- Multi-way partitioning with area and pin constraints: Liu and Wong, ISPD-97.
- Multi-resource partitioning: Liu, Zhu, and Wong, FPGA-98.
- Partitioning for time-multiplexed FPGAs: Liu and Wong, ICCAD-98.

Flow Networks

- A **flow network** $G = (V, E)$ is a **directed** graph in which each edge $(u, v) \in E$ has a **capacity** $c(u, v) > 0$.
- There is exactly one node with no incoming (outgoing) edges, called the **source** s (**sink** t).
- A **flow** f: $V \times V \rightarrow R$ satisfies
  - **Capacity constraint: $f(u, v) \leq c(u, v)$, $\forall$ $u, v \in V$.**
  - **Skew symmetry: $f(u, v) = -f(v, u)$, $\forall$ $u, v \in V$.**
  - **Flow conservation:** $\sum_{v \in V} f(u,v) = 0$ **, $\forall$ u $\in$ V-{s, t}.**
- The **value** of a flow f: $|f|$ = $\sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$
- **Maximum-flow problem:** Given a flow network G with source s and sink t, find a flow of maximum value from s to t.

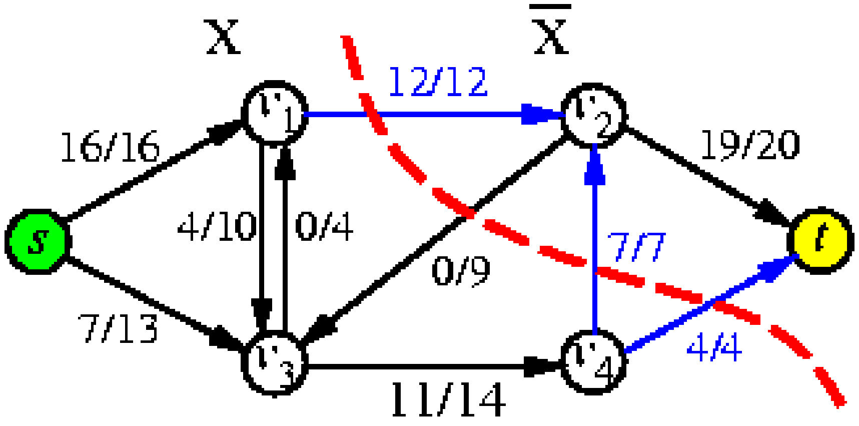


flow/capacity

max flow |f| = 16 + 7 = 23

Max-Flow Min-Cut

- A **cut** $(X, \bar{X})$ of flow network G=(V, E) is a partition of V into X and $\bar{X}$ = V - X such that $s \in X$ and $t \in \bar{X}$.
  - **Capacity of a cut:** *cap*(X, $\bar{X}$) = $\sum_{u \in X, v \in \bar{X}} c(u,v)$. (Count only **forward** edges!)
- **Max-flow min-cut theorem:** Ford & Fulkerson, 1956.
  - f is a max-flow ⇔ |f| = *cap*(X, $\bar{X}$) for some min-cut (X, $\bar{X}$).



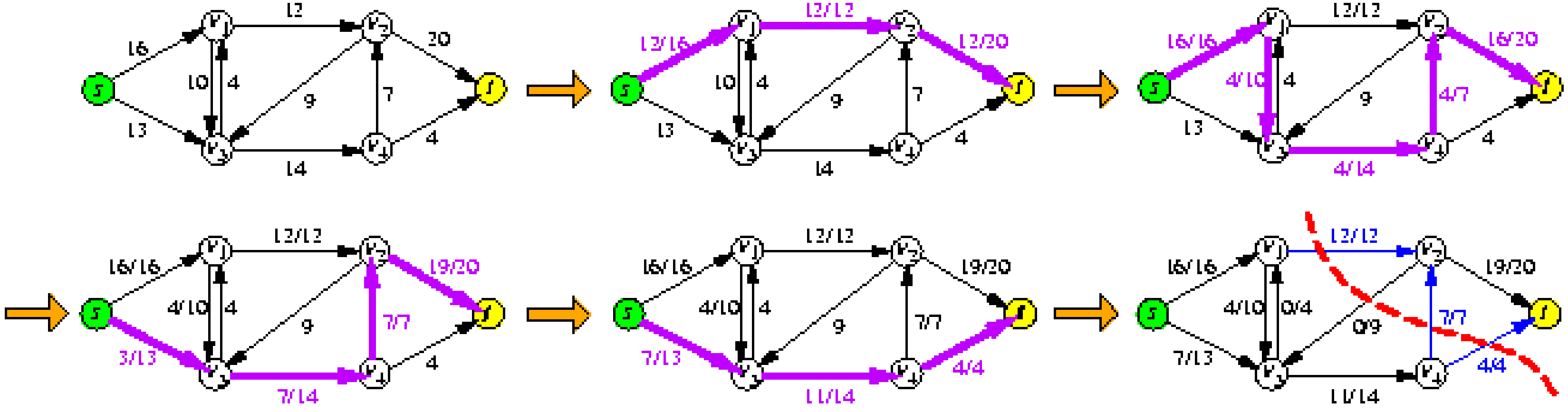
flow/capacity

max flow |f| = 16 + 7 = 23

cap(X, $\bar{X}$) = 12 + 7 + 4 = 23

Network Flow Algorithms

- An **augmenting path** *p* is a simple path from *s* to *t* with the following properties:
  - For every edge $(u, v) \in E$ on *p* in the **forward** direction (a **forward edge**), we have $f(u, v) < c(u, v)$.
  - For every edge $(u, v) \in E$ on *p* in the **reverse** direction (a **backward edge**), we have $f(u, v) > 0$.
- *f* is a max-flow ⇔ no more augmenting path.



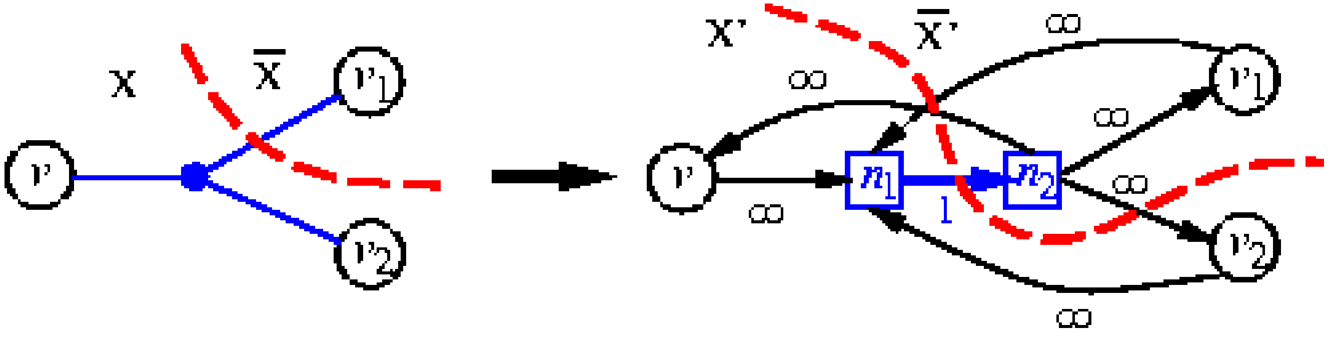
- First algorithm by Ford & Fulkerson in 1959: $O(E|f|)$; First **polynomial-time** algorithm by Edmonds & Karp in 1969: $O(E^2V)$; Goldberg & Tarjan in 1985: $O(EV \lg(V^2E))$, etc.

Network Flow Based Partitioning

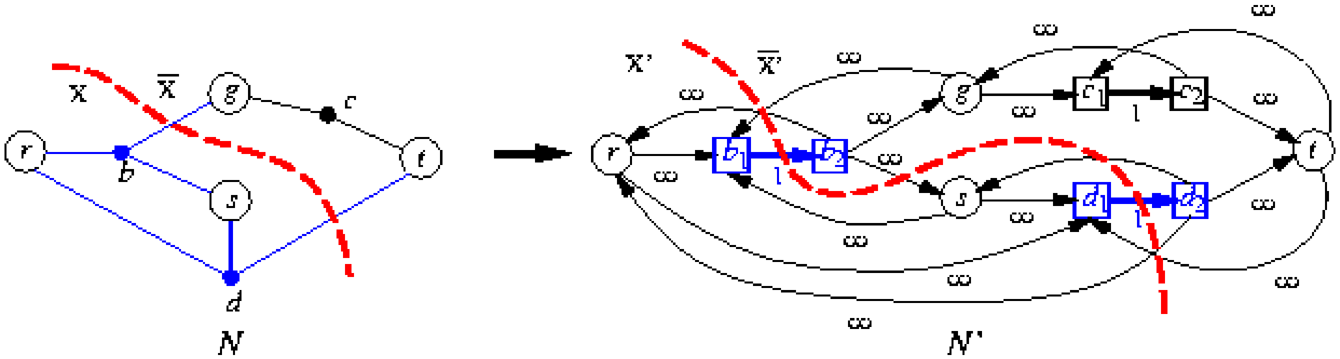
- Why was the technique not wisely used in partitioning?
  - Works on graphs, not hypergraphs.
  - Results in unbalanced partitions; repeated min-cut for balance: |*V*| max-flows, time-consuming!
- Yang & Wong, ICCAD-94.
  - Exact **net** modeling by flow network.
  - Optimal algorithm for min-net-cut bipartition (unbalanced).
  - Efficient implementation for repeated min-net-cut: same asymptotic time complexity as **one** max-flow computation.

Min-Net-Cut Bipartitioning

- Net modeling by flow network:

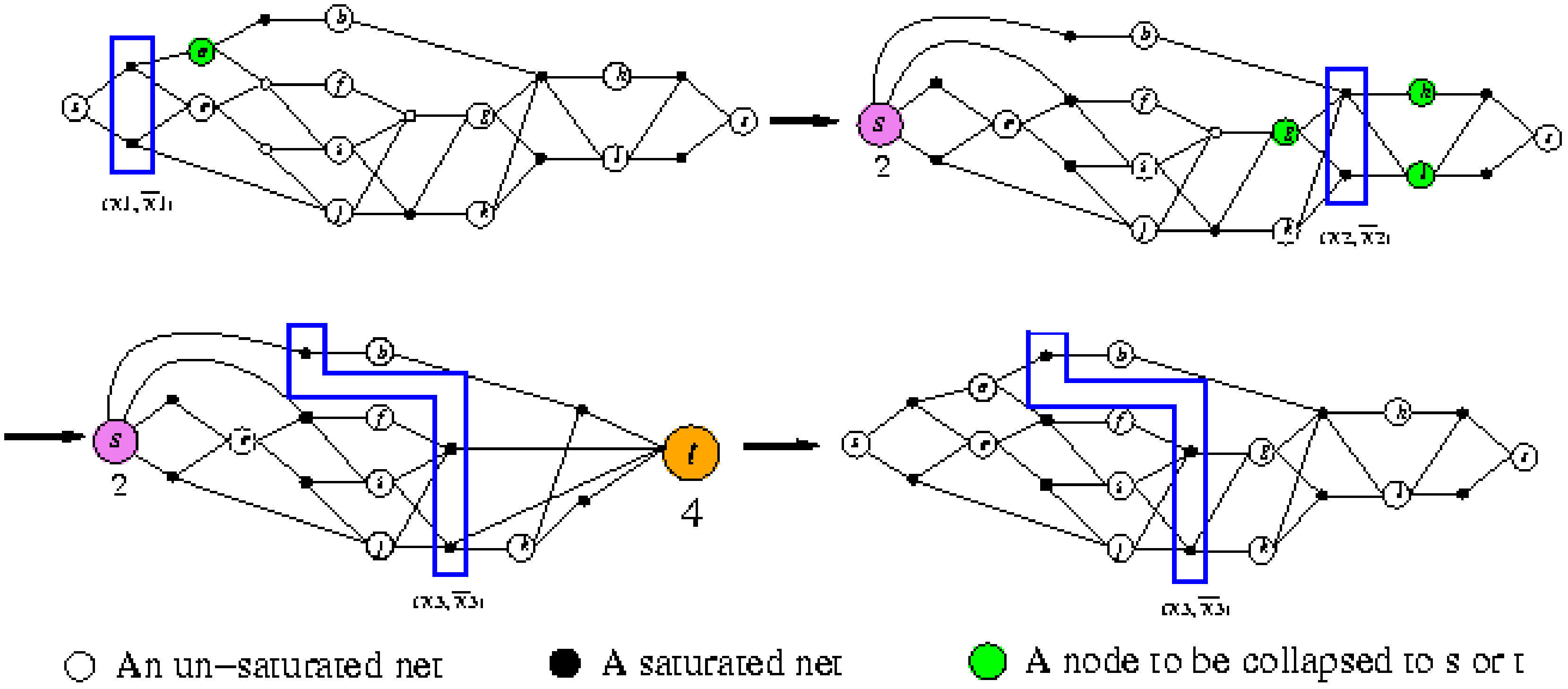


- A min-net-cut $(X, \bar{X})$ in *N* ⇔ A min-capacity-cut $(X', \bar{X}')$ in *N*'.
- Size of flow network: |*V*'| ≤ 3|*V*|, |*E*'| ≤ 2|*E*| + 3|*V*|.
- Time complexity: O(min-net-cut-size) × |*E*| = *O*(|*V*||*E*|).



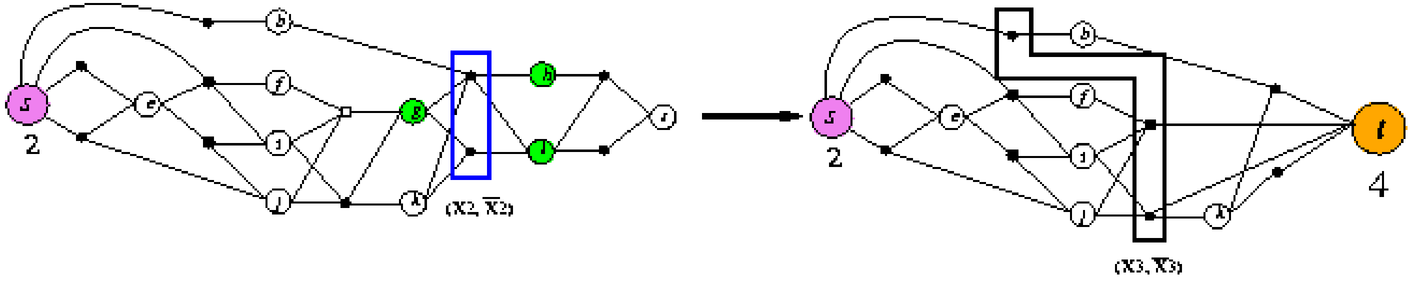
Repeated Min-Cut for Balanced Bipartition (FBB)

- Allow component weights to deviate from (1 - ε)*W*/2 to (1 + ε)*W*/2.



Incremental Flow

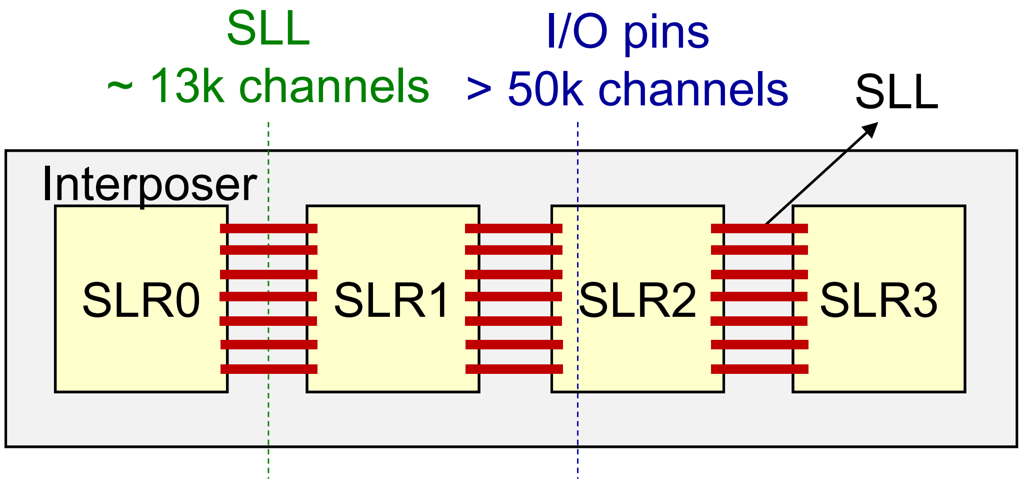
- Repeatedly compute max-flow: very time-consuming.
- No need to compute max-flow from scratch in each iteration.
- Retain the flow function computed in the previous iteration.
- Find additional flow in each iteration. Still correct.
- FBB time complexity: $O(|V||E|)$, the same as **one** max-flow.
  - At most $2|V|$ augmenting path computations.
    - At each augmenting path computation, either (1) an augmenting path is found, or (2) a new cut is found, and at least 1 node is collapsed to *s* or *t*.
    - At most $|f| \leq |V|$ augmenting paths will be found, since bridging edges have unit capacity.
  - An augmenting path computation: $O(|E|)$ time.



Appendix C:

Multi-2.5D FPGA-Based System Partitioning

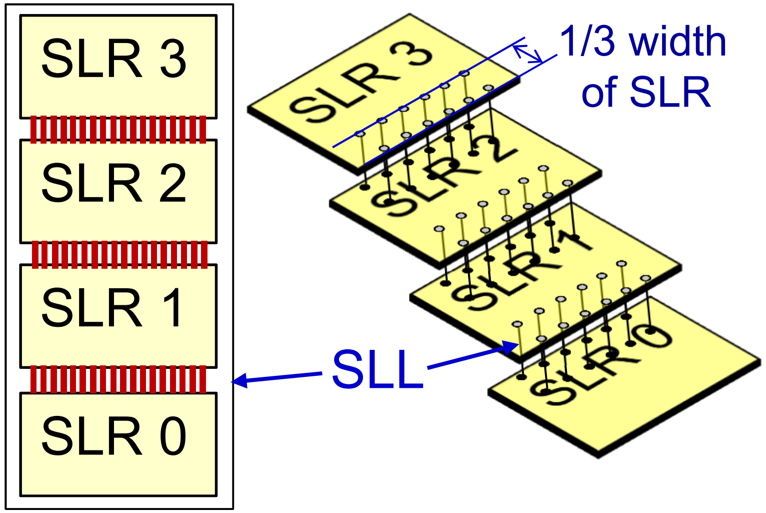
Chen, Sun, and Chang, "Simultaneous partitioning and signals grouping for time-division multiplexing in 2.5D FPGA-based systems," ICCAD-18.



Xilinx Virtex-7 2000T

Multi-2.5D FPGA-Based System

- Super logic region (SLR) for dies
- SLRs are connected by super long lines (SLLs)
- #I/O pins for SLLs are limited
- SLL delay is significant



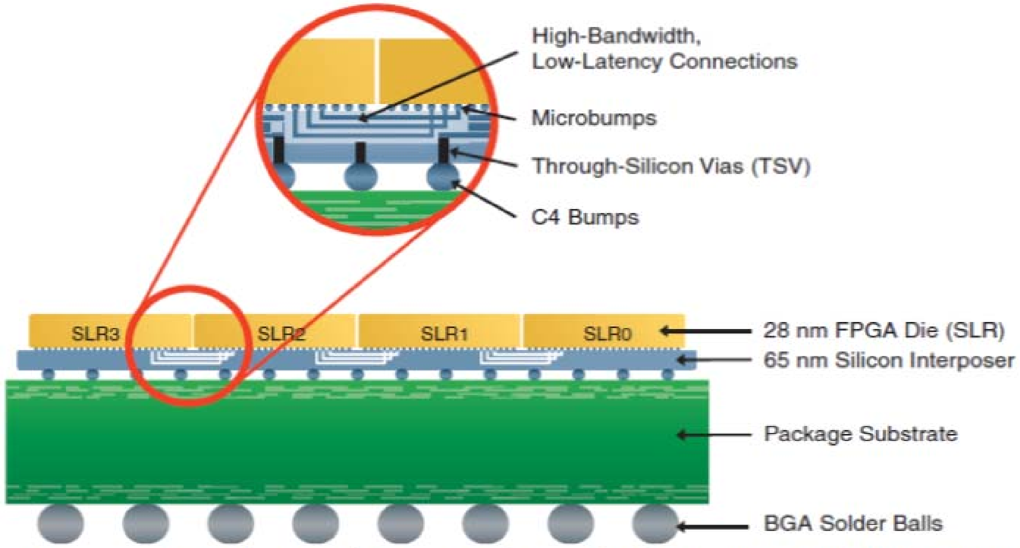
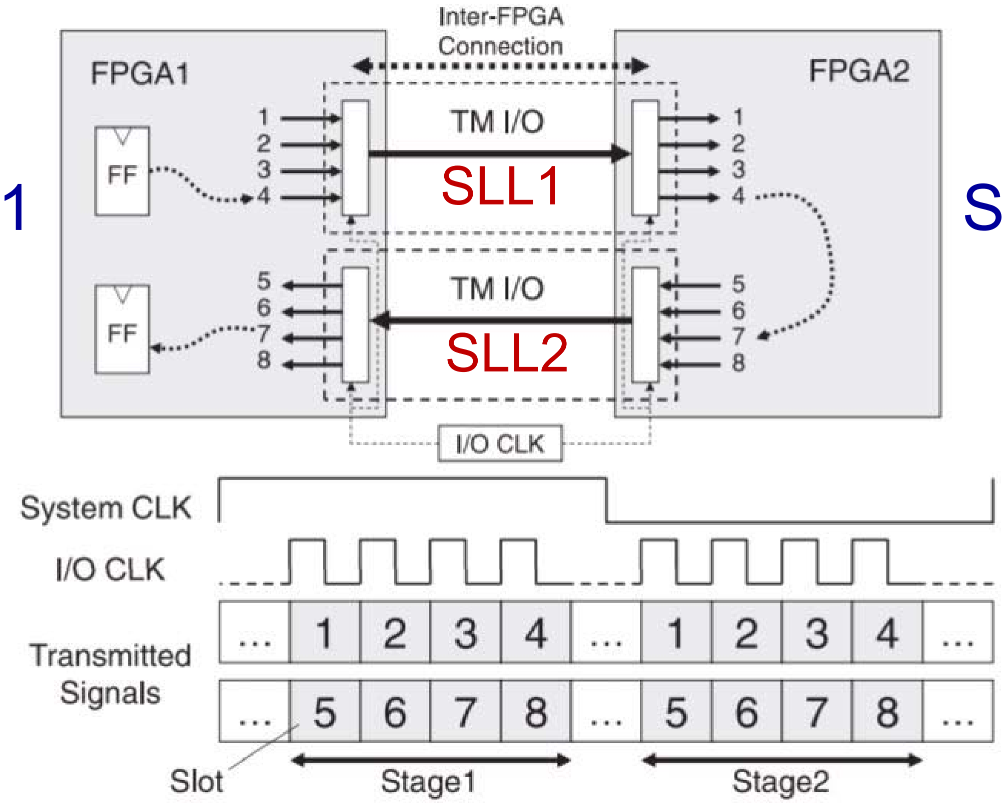


Figure 1. Schematic X-section of Virtex®-7 2000T FPGA Assembled with Interposer with TSVs

Time-Division-Multiplexing (TDM) Technique

- I/O time-division-multiplexing (TDM) is used to let multiple signal nets share one physical routing channel/pin at different times
  - All 8 routing signals can be transmitted in one system clock period.
- Pros: increase the signal capability and routability
- Con: incur longer inter-FPGA signal delay

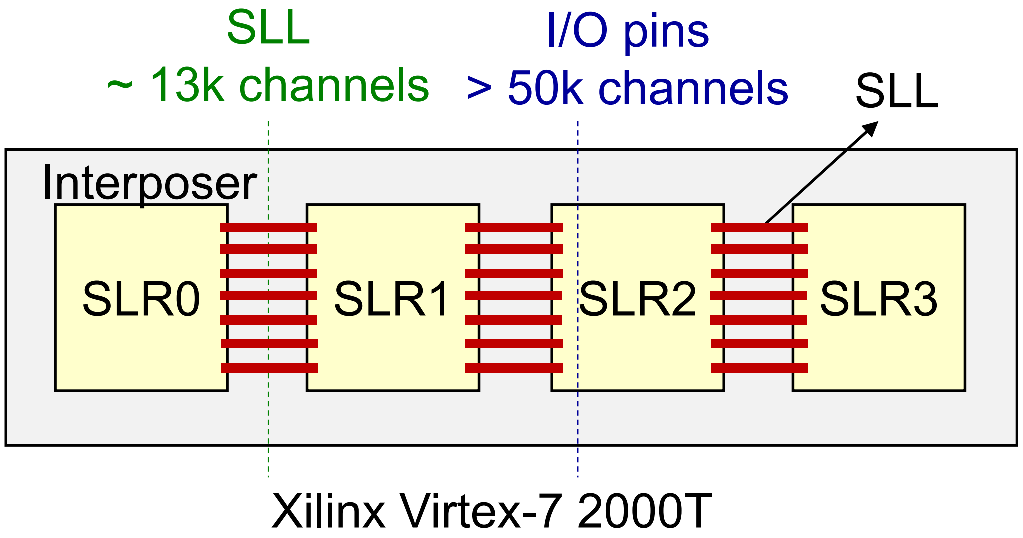
SLR1



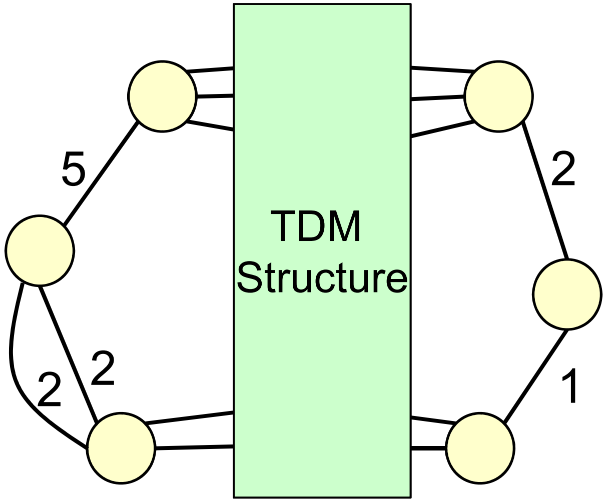
SLR2

Time-division Multiplexing in 2.5D FPGA Systems

- Chen, Sun, Chang, “Simultaneous partitioning and signals grouping for time-division multiplexing in 2.5D FPGA-based systems,” ICCAD’18
- SLLs are typically fewer than the routing nets across SLRs
- I/O TDM is used to share the crossing nets
  - Inter-SLR nets are grouped together to go through a single SLL
- TDM wire delay grows linearly with #shared nets

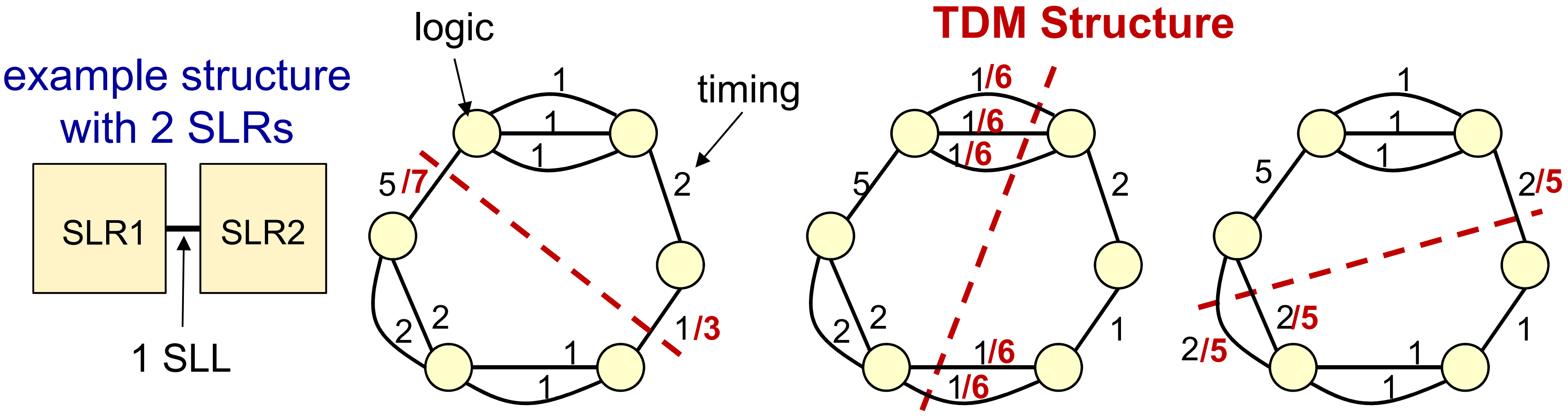


Xilinx Virtex-7 2000T



Balanced Partitioning and Signal Grouping

- Two-stage flow of balanced partitioning + signal grouping might just produce an undesired solution for timing optimization
- Need to consider partitioning & signal grouping simultaneously



example structure with 2 SLRs

TDM Structure

Cut size	**2**	5	3
Total timing cut weight	6	**5**	6
Timing criticality	7	6	**5**

Grouping penalty = # shared nets

Timing criticality = timing + grouping penalty