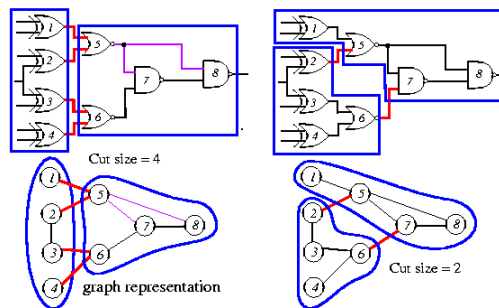


## Unit 3: Partitioning

- Course contents:
  - Kernighan & Lin heuristic
  - Fiduccia-Mattheyses heuristic
  - Multilevel circuit partitioning
  - Appendix: Network-flow based method

- Readings
  - S&Y: Chapter 2
  - Sherwani: Chapter 5



Unit 3

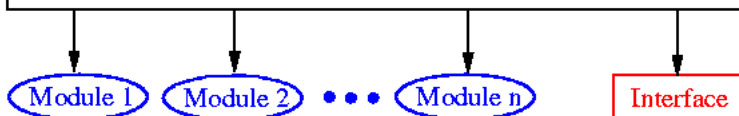
Y.-W. Chang

1

## 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 manageable size.



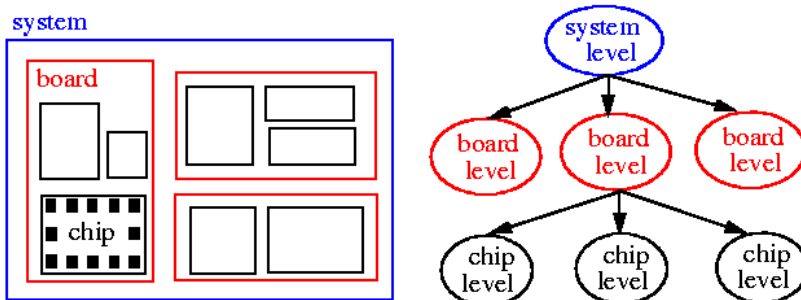
Unit 3

Y.-W. Chang

2

## Levels of Partitioning

- The levels of partitioning: system, board, chip.
- Hierarchical partitioning: higher costs for higher levels.



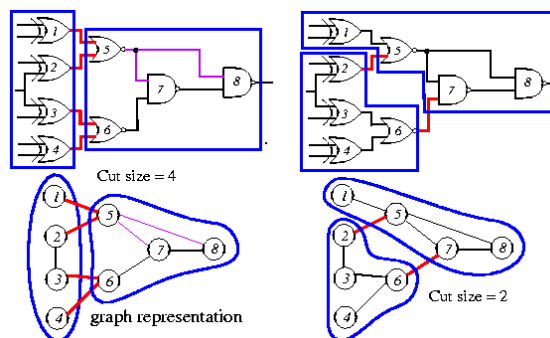
Unit 3

Y.-W. Chang

3

## 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.
  - More constraints are possible for some applications.
- Cutset? Cut size? Size of a component?



Unit 3

Y.-W. Chang

4

## Problem Definition: Partitioning

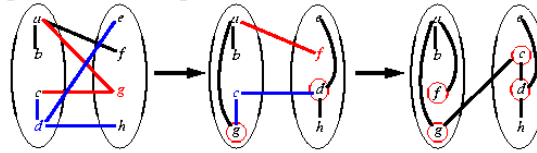
- **k-way partitioning:** Given a graph  $G(V, E)$ , where each vertex  $v \in V$  has a **size**  $s(v)$  and each edge  $e \in E$  has a **weight**  $w(e)$ , the problem is to divide the set  $V$  into  $k$  **disjoint subsets**  $V_1, V_2, \dots, V_k$  such that an objective function is optimized, 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.

## 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.
- Till 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

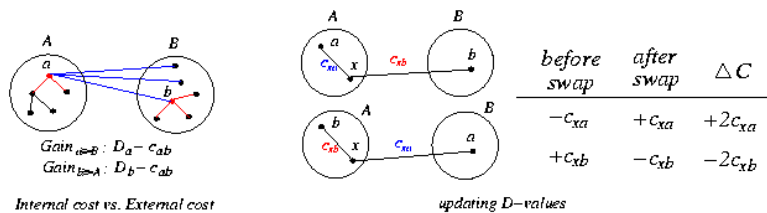
- Questions: How to compute cost reduction? What pairs to be swapped?
  - Consider the change of internal & external connections.

## Properties

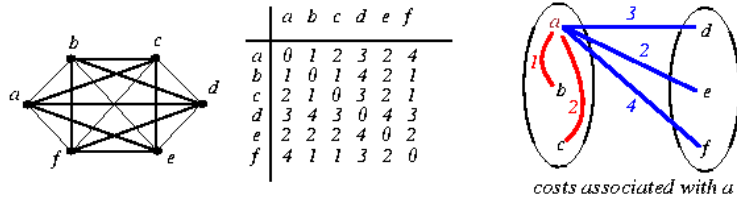
- Two sets  $A$  and  $B$  such that  $|A| = n = |B|$  and  $A \cap B = \emptyset$ .
- **External cost** of  $a \in A$ :  $E_a = \sum_{v \in B} c_{av}$
- **Internal cost** of  $a \in A$ :  $I_a = \sum_{v \in A} c_{av}$
- $D$ -value of a vertex  $a$ :  $D_a = E_a - I_a$  (cost reduction for moving  $a$ ).
- **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

$$D'_x = D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\}$$

$$D'_y = D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\}.$$



## Kernighan-Lin Heuristic: A Weighted Example



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

• Iteration 1:

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

Unit 3

Y.-W. Chang

9

## g-Value Computation

• Iteration 1:

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

•  $g_{xy} = D_x + D_y - 2c_{xy}$

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

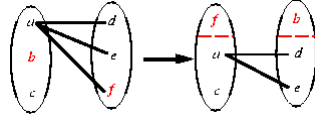
• Swap  $b$  and  $f$  ( $\hat{g}_1 = 4$ )

Unit 3

Y.-W. Chang

10

## D-Value Computation



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

$$D'_a = D_a + 2c_{ab} - 2c_{af} = 6 + 2 \times 1 - 2 \times 4 = 0$$

$$D'_c = D_c + 2c_{cb} - 2c_{cf} = 3 + 2 \times 1 - 2 \times 1 = 3$$

$$D'_d = D_d + 2c_{df} - 2c_{db} = 3 + 2 \times 3 - 2 \times 4 = 1$$

$$D'_e = D_e + 2c_{ef} - 2c_{eb} = 0 + 2 \times 2 - 2 \times 2 = 0$$

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

$$g_{ad} = D'_a + D'_d - 2c_{ad} = 0 + 1 - 2 \times 3 = -5$$

$$g_{ae} = D'_a + D'_e - 2c_{ae} = 0 + 0 - 2 \times 2 = -4$$

$$g_{cd} = D'_c + D'_d - 2c_{cd} = 3 + 1 - 2 \times 3 = -2$$

$$g_{ce} = D'_c + D'_e - 2c_{ce} = 3 + 0 - 2 \times 2 = -1 \text{ (maximum)}$$

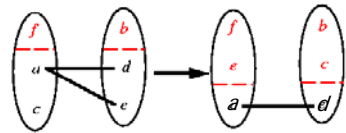
- Swap  $c$  and  $e!$  ( $\hat{g}_2 = -1$ )

Unit 3

Y.-W. Chang

11

## Swapping Pair Determination



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

$$D''_a = D'_a + 2c_{ac} - 2c_{ae} = 0 + 2 \times 2 - 2 \times 2 = 0$$

$$D''_d = D'_d + 2c_{de} - 2c_{dc} = 1 + 2 \times 4 - 2 \times 3 = 3$$

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

$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3 (\hat{g}_3 = -3)$$

- 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  $\max \sum_{i=1}^k \hat{g}_i = 4 \quad (k=1) \Rightarrow$  Swap  $b$  and  $f$ .

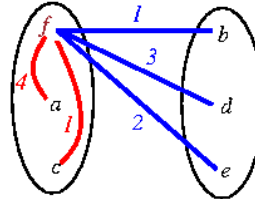
Unit 3

Y.-W. Chang

12

## Next Iteration

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
<i>a</i>	0	1	2	3	2	4
<i>b</i>	1	0	1	4	2	1
<i>c</i>	2	1	0	3	2	1
<i>d</i>	3	4	3	0	4	3
<i>e</i>	2	2	2	4	0	2
<i>f</i>	4	1	1	3	2	0



$$\text{Initial cut cost} = (1+3+2)+(1+3+2)+(1+3+2) = 18 \quad (22-4)$$

- 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 =  $\max \sum_{i=1}^k \hat{g}_i = 0$  ( $k=3$ )  $\Rightarrow$  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.

1 **begin**

2 Bipartition *G* into *A* and *B* such that  $|V_A| = |V_B|$ ,  $V_A \cap V_B = \emptyset$ ,  
and  $V_A \cup V_B = V$ .

3 **repeat**

4 Compute  $D_v, \forall v \in V$ .

5 **for**  $i=1$  **to**  $n$  **do**

6 Find a pair of unlocked vertices  $v_{a_i} \in V_A$  and  $v_{b_i} \in V_B$  whose exchange makes the largest decrease or smallest increase in cut cost;

7 Mark  $v_{a_i}$  and  $v_{b_i}$  as locked, store the gain  $\hat{g}_i$ , and compute the new  $D_v$  for all unlocked  $v \in V$ ;

8 Find  $k$ , such that  $G_k = \sum_{i=1}^k \hat{g}_i$  is maximized;

9 **if**  $G_k > 0$  **then**

10 Move  $v_{a_1}, \dots, v_{a_k}$  from  $V_A$  to  $V_B$  and  $v_{b_1}, \dots, v_{b_k}$  from  $V_B$  to  $V_A$ ;

11 Unlock  $v, \forall v \in V$ .

12 **until**  $G_k \leq 0$ ;

13 **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?

## Extensions of K-L Heuristic

---

- **Unequal sized subsets** (assume  $n_1 < n_2$ )
  1. Partition:  $|A| = n_1$  and  $|B| = n_2$ .
  2. Add  $n_2 - n_1$  dummy vertices to set  $A$ . Dummy vertices have no connections to the original graph.
  3. Apply the Kernighan-Lin algorithm.
  4. Remove all dummy vertices.
- **Unequal sized "vertices"**
  1. Assume that the smallest "vertex" has unit size.
  2. Replace each vertex of size  $s$  with  $s$  vertices which are fully connected with edges of infinite weight.
  3. Apply the Kernighan-Lin algorithm.
- **$k$ -way partition**
  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-partition.

## 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**,  $O(n^3)$ .

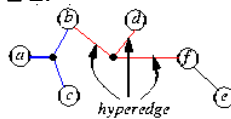
Unit 3

Y.-W. Chang

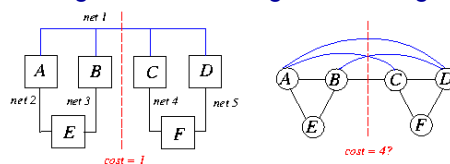
17

## 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  $|N_i| \geq 2$ .



- 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.



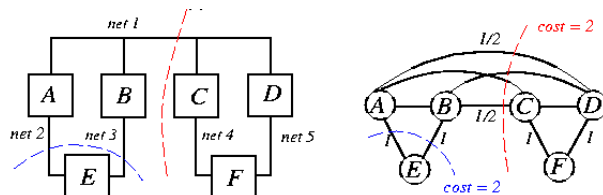
Unit 3

Y.-W. Chang

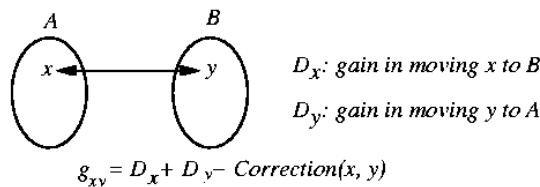
18

## 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.



Unit 3

Y.-W. Chang

19

## 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 cutsizes is extended to hypergraphs.
  - Only a **single vertex is moved** across the cut in a single move.
  - 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.

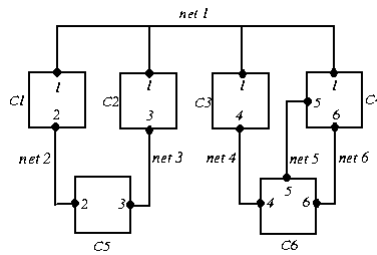
Unit 3

Y.-W. Chang

20

## 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;  $P = p(1) + \dots + p(C) = n(1) + \dots + n(N)$ .



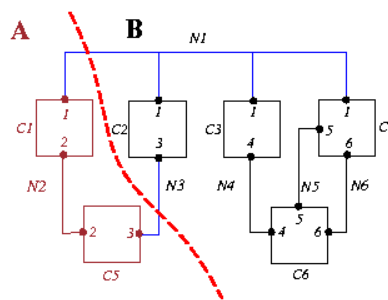
Unit 3

Y.-W. Chang

21

## 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.

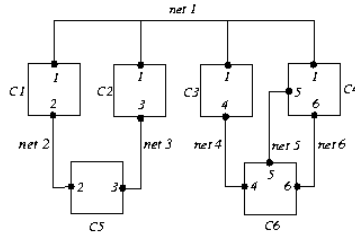


Unit 3

Y.-W. Chang

22

## 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

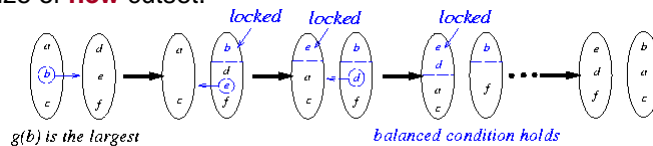
- Only move a cell at a time, preserving “balance.”

$$\frac{|A|}{|A| + |B|} \approx r$$

$$rW - S_{max} \leq |A| \leq rW + S_{max},$$

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

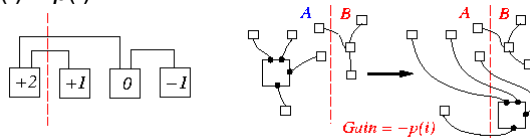
- $g(i)$ : gain in moving cell  $i$  to the other set, i.e., size of **old** cutset - size of **new** cutset.



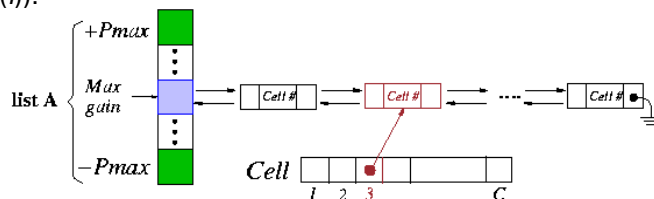
- Suppose  $\hat{g}_i$ 's:  $g(b), g(e), g(d), g(a), g(f), g(c)$  and the largest partial sum is  $g(b) + g(e) + g(d)$ . Then we should move  $b, e, d \Rightarrow$  two resulting sets:  $\{a, c, e, d, f\}, \{b, a\}$ .

## 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.

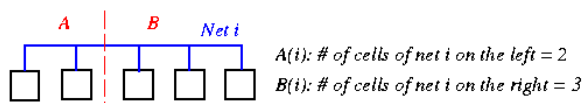
Unit 3

Y.-W. Chang

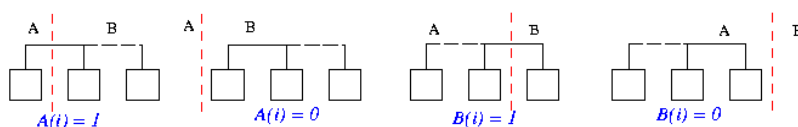
25

## 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.



- 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.**

Unit 3

Y.-W. Chang

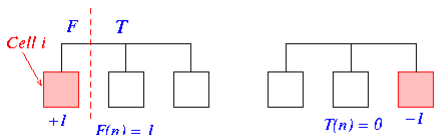
26

## Computing Cell Gains

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

```

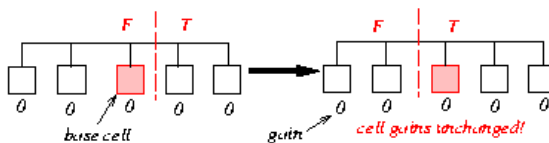
 $g(i) \leftarrow 0;$ 
 $F \leftarrow$  the "from block" of Cell  $i$ ;
 $T \leftarrow$  the "to block" of Cell  $i$ ;
for each net  $n$  on Cell  $i$  do
  if  $F(n)=1$  then  $g(i) \leftarrow g(i)+1;$ 
  if  $T(n)=0$  then  $g(i) \leftarrow g(i)-1;$ 
    
```



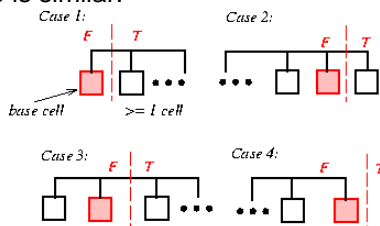
- Will show: Only need  $O(P)$  time to maintain all cell gains in one pass.

## Updating Cell Gains

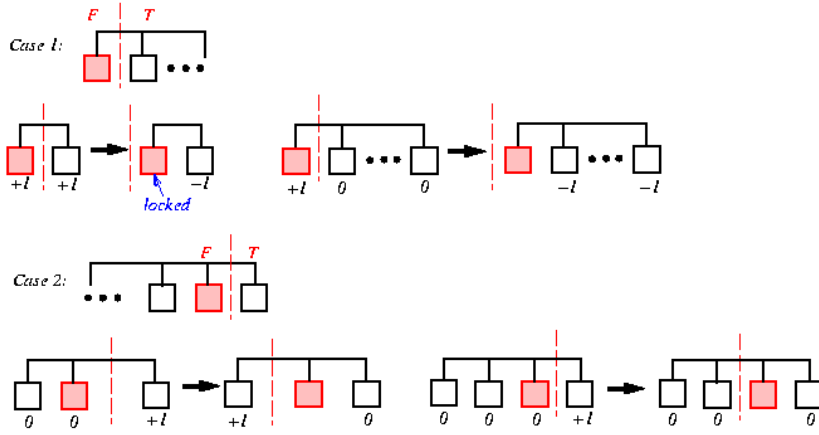
- To update the gains, we only need to look at those nets, connected to the base cell, which are critical **before** or **after** the move.
- Base cell:** The cell selected for movement from one set to the other.



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



## Updating Cell Gains (cont'd)

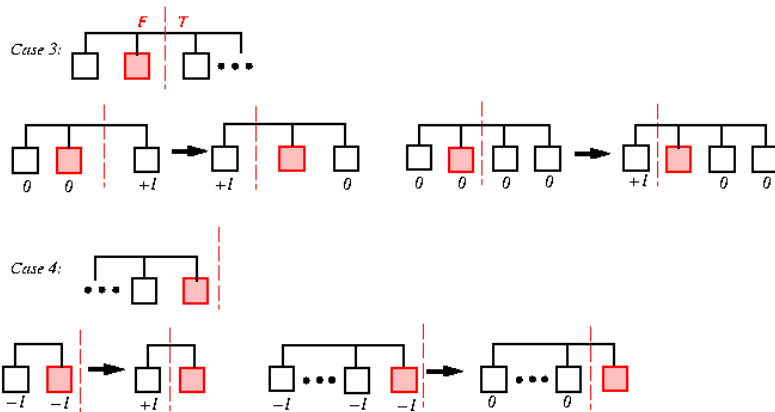


Unit 3

Y.-W. Chang

29

## Updating Cell Gains (cont'd)



Unit 3

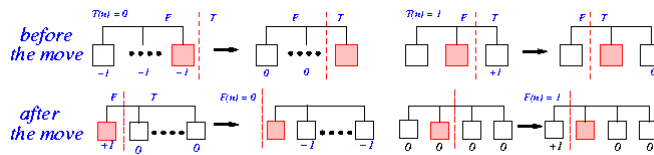
Y.-W. Chang

30

## Algorithm for Updating Cell Gains

```

Algorithm: Update_Gain
1 begin /* move base cells and update neighbors' gains */
2 F ← the From Block of the base cell;
3 T ← the To Block of the base cell;
4 Lock the base cell and complement its block;
5 for each net n on the base cell do
  /* check critical nets before the move */
6  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 */
7  F(n) ← F(n) - 1; T(n) ← T(n)+1;
  /* check for critical nets after the move */
8  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
9 end
    
```



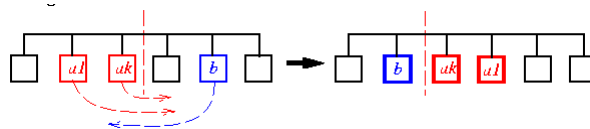
Unit 3

Y.-W. Chang

31

## 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  $a_1, a_2, \dots, a_k$  from left to right, and then move  $b$  from right to left  $\Rightarrow$  At most only moving  $a_1, a_2, \dots, a_k$  and  $b$  need updating!



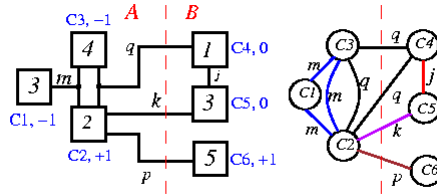
- To update the cell gains, it takes  $O(n(i))$  work for Net  $i$ .
- Total time =  $n(1)+n(2)+\dots+n(N) = O(P)$ .

Unit 3

Y.-W. Chang

32

## F-M Heuristic: An Example



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

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

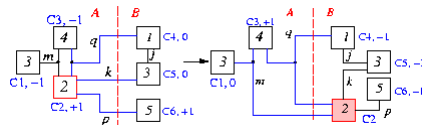
- Balanced criterion:  $r|V| - S_{max} \leq |A| \leq r|V| + S_{max}$ . Let  $r = 0.4 \Rightarrow |A| = 9$ ,  $|V| = 18$ ,  $S_{max} = 5$ ,  $r|V| = 7.2 \Rightarrow$  Balanced:  $2.2 \leq 9 \leq 12.2$ !
- maximum gain:  $c_2$  and balanced:  $2.2 \leq 9-2 \leq 12.2 \Rightarrow$  Move  $c_2$  from A to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

Unit 3

Y.-W. Chang

33

## 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
$c_5$									0	-2
$c_6$	-1			-1				-1	+1	-1

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

Unit 3

Y.-W. Chang

34

## Summary of the Example

Step	Cell	Max gain	A	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

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

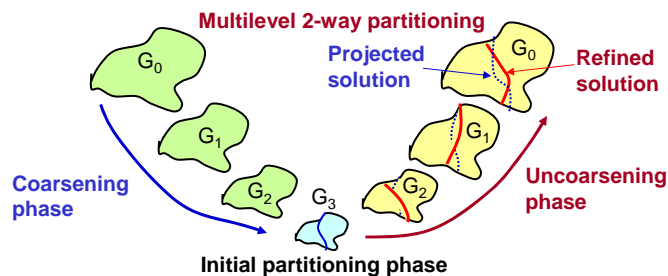
Unit 3

Y.-W. Chang

35

## 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).



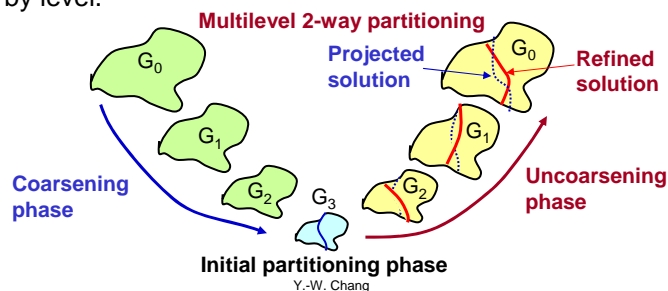
Unit 3

Y.-W. Chang

36

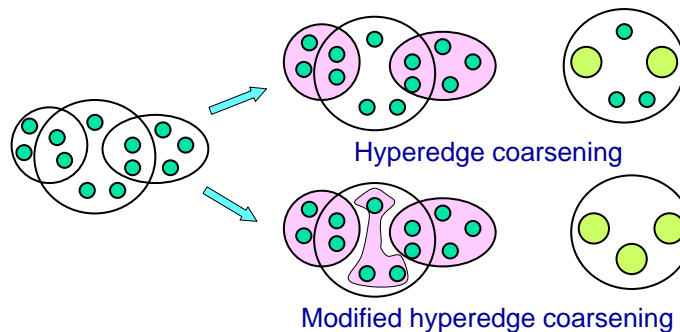
## hMetis: Multilevel 2-way Partitioner

- 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.



## 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.

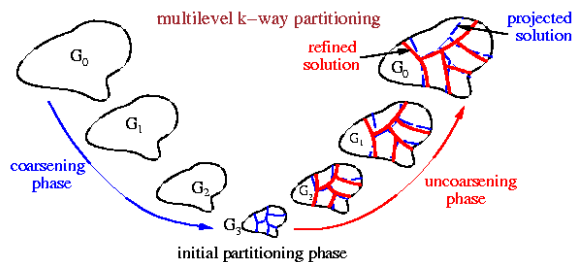
Unit 3

Y.-W. Chang

39

## Multilevel $k$ -way Partitioning

- Karypis & Kumar, "Multilevel  $k$ -way hypergraph partitioning," DAC-99 (**best published results so far**).
- **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).



Unit 3

40

## Summary: Partitioning

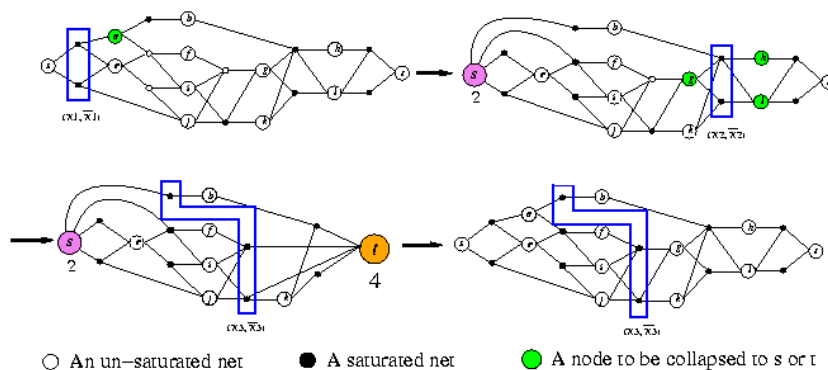
- Discussed methods: **group migration** (K-L, F-M) and **multilevel partitioning (hMetis)**.
- Other important partitioning approaches
  - **Network-flow method:** Yang and Wong, ICCAD-94, ICCAD-95.
  - **Spectral method:** Barnes, *SIAM J. Algebraic and Discrete Methods*, 1982; Boppana, FOCS-87; Alpert & Kahng, DAC-95, DAC-96, etc.
  - **Probabilistic approach:** Dutt & Deng, DAC-96; Chao, et al., ICCAD-99.
  - **Mathematical programming:** Shih & Kuh, DAC-93 (quadratic programming); Wu et al., TCAD, Oct. 2001 (ILP)
  - **Unified approach:** Network flow + Spectral, Li, et al., ICCAD-95.
- Clustering: Cong, et al., ICCAD-97; Chao, et al., ICCAD-99.
- Earlier survey: Alpert & Kahng, *Integration*, 1995.
- **Cong et al, ISPD-03: hMetis is almost “good enough.”**
- Recent work focuses on exact modeling between cut cost and wirelength (Chen et al., ICCAD-05; Chan et al., ISPD-06)
  - Apply to floorplanning and placement

Unit 3

Y.-W. Chang

41

## Appendix: Network Flow Based Partitioning



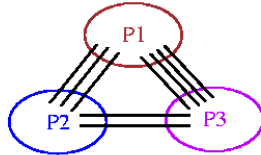
Unit 3

Y.-W. Chang

42

## 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.

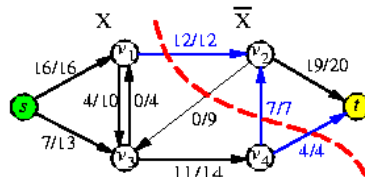
Unit 3

Y.-W. Chang

43

## 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  $|f| = 16 + 7 = 23$

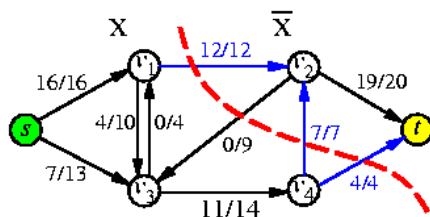
Unit 3

Y.-W. Chang

44

## Max-Flow Min-Cut

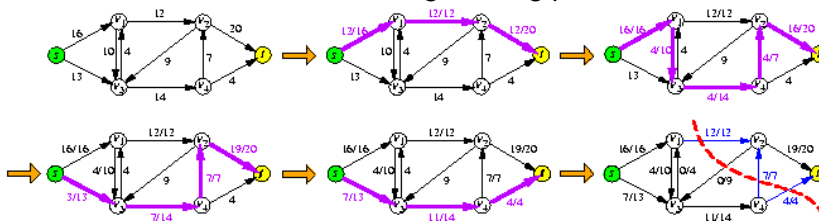
- A **cut**  $(X, \bar{X})$  of flow network  $G=(V, E)$  is a partition of  $V$  into  $X$  and  $\bar{X} = V - X$  such that  $s \in X$  and  $t \in \bar{X}$ .
  - **Capacity of a cut:**  $cap(X, \bar{X}) = \sum_{u \in X, v \in \bar{X}} c(u, v)$ . (Count only **forward** edges!)
- **Max-flow min-cut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

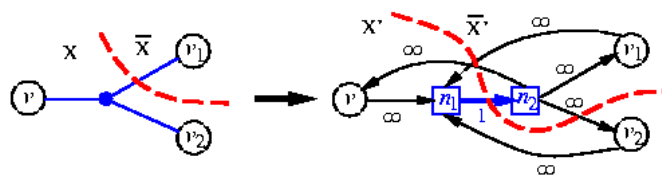
Unit 3

Y.-W. Chang

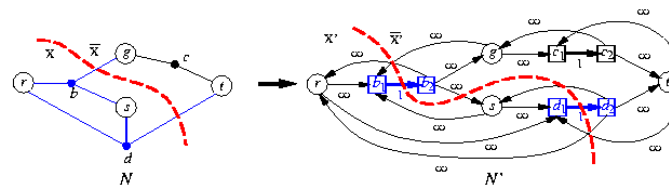
47

## Min-Net-Cut Bipartition

- Net modeling by flow network:

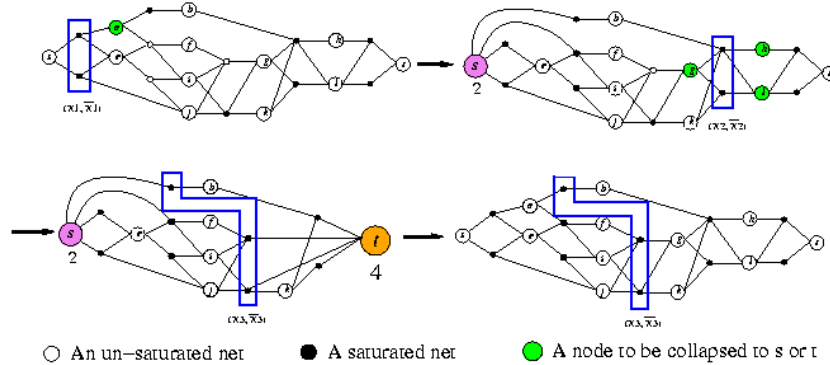


- A min-net-cut  $(x, \bar{x})$  in  $N \Leftrightarrow$  A min-capacity-cut  $(x', \bar{x}')$  in  $N'$ .
- Size of flow network:  $|V| \leq 3|V|$ ,  $|E| \leq 2|E| + 3|V|$ .
- Time complexity:  $O(\text{min-net-cut-size}) \times |E| = O(|V||E|)$ .



## Repeated Min-Cut for Balanced Bipartition (FBB)

- Allow component weights to deviate from  $(1 - \varepsilon)W/2$  to  $(1 + \varepsilon)W/2$ .



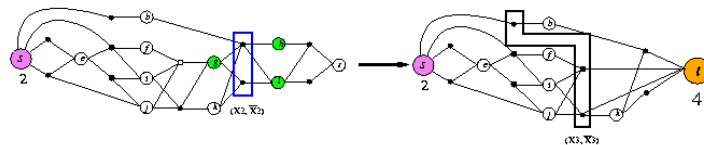
Unit 3

Y.-W. Chang

49

## 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.



Unit 3

Y.-W. Chang

50