

BACKEND DESIGN

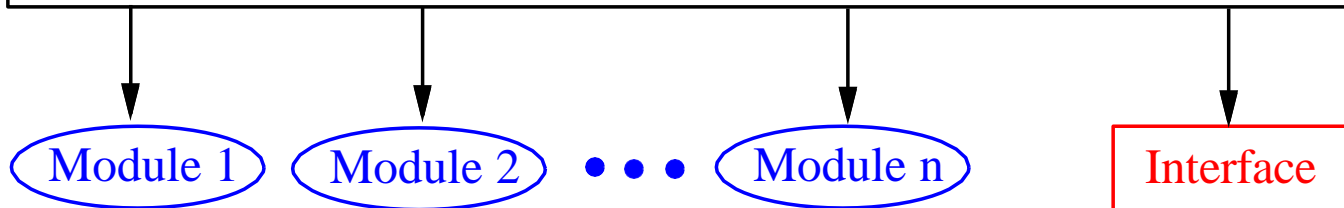
Circuit Partitioning

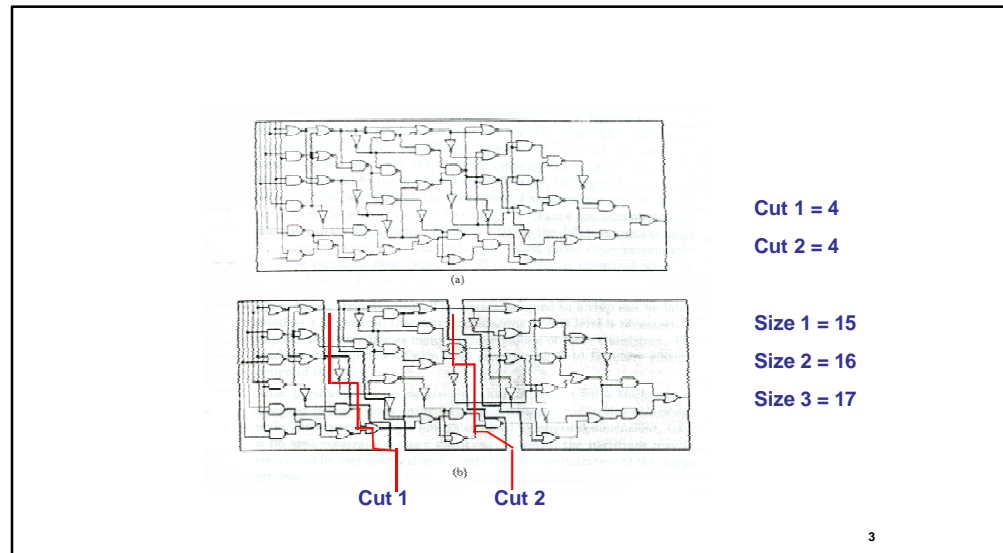
Partitioning

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.

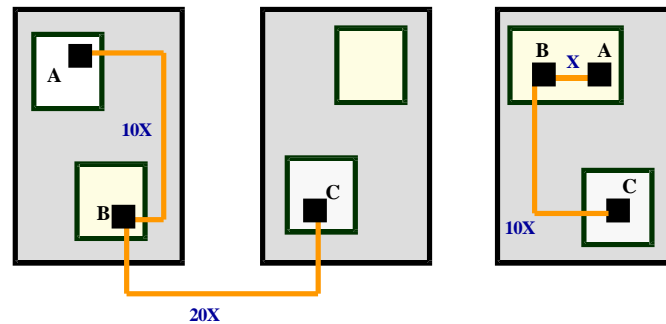




Partitioning at Different Levels

- Can be done at multiple levels:
 - System level
 - Board level
 - Chip level
- Delay implications are different:
 - Intrachip → X
 - Intraboard → 10X
 - Interboard → 20X

Different Delays in a Chip



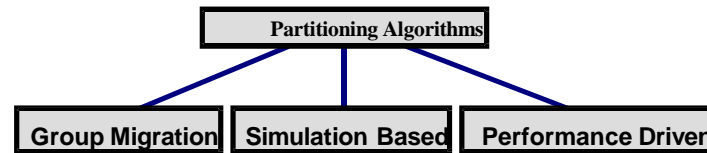
5

Problem Formulation

- Partition a given netlist into smaller netlists such that:
 1. Interconnection between partitions is minimized.
 2. Delay due to partitioning is minimized.
 3. Number of terminals is less than a predetermined maximum value.
 4. The area of each partition remains within specified bounds.
 5. The number of partitions also remains within specified bounds.

6

Classification of Partitioning Algorithms



Kernighan-Lin

Fiduccia-Mattheyses

Goldberg-Burstein

Simulated Annealing

Simulated Evolution

7

Group Migration Algorithms

- Kernighan-Lin
 - An iterative improvement algorithm for balanced two-way partitioning.
- Goldberg-Burstein
 - Uses properties of graphs to improve the performance of K-L algorithm.
- Fiduccia-Mattheyses
 - Considers multi-pin nets.
 - Can generate partitions of unequal sizes.
 - Uses efficient data structure to represent nodes.

8

Partitioning

system design

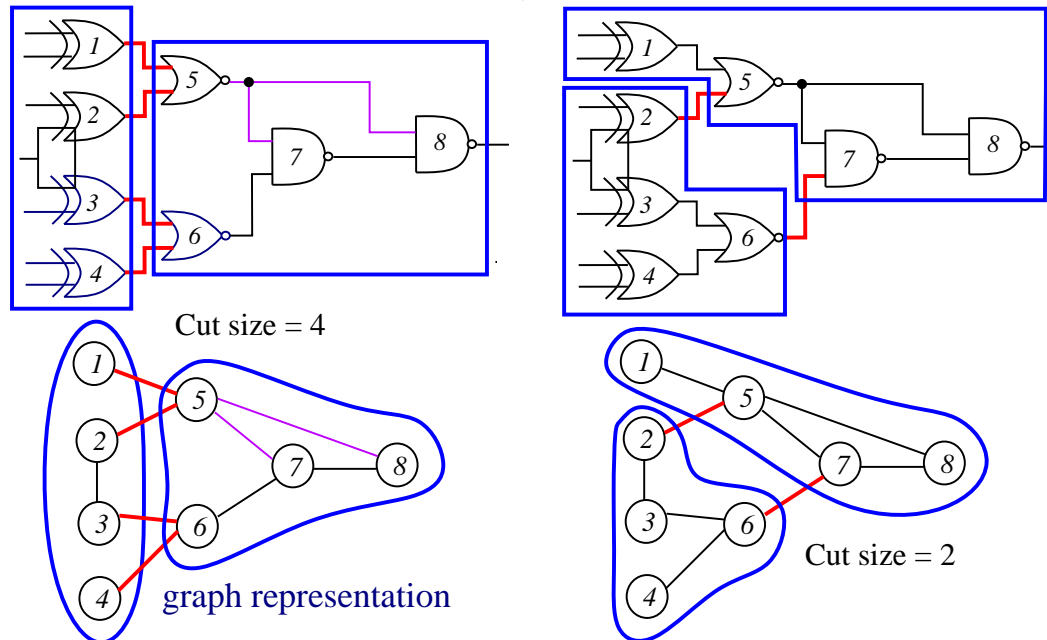


- Decomposition of a complex system into smaller subsystems.
- Each subsystem can be designed independently speeding up the design process.
- Decomposition scheme has to minimize the interconnections among the subsystems.
- Decomposition is carried out hierarchically until each subsystem is of manageable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



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?



Problem Definition: Partitioning

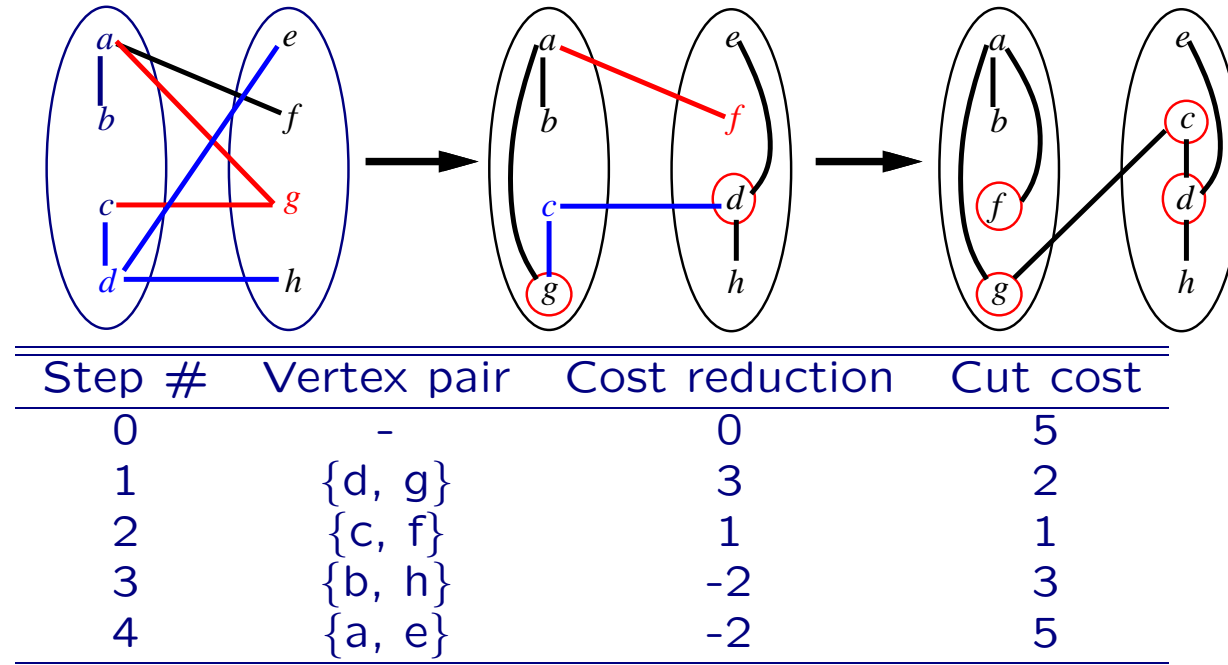
- **k -way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets V_1, V_2, \dots, V_k , such that an objective function is optimized, subject to certain constraints.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

- 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.

Kernighan-Lin Algorithm: A Simple Example

- Each edge has a unit weight.

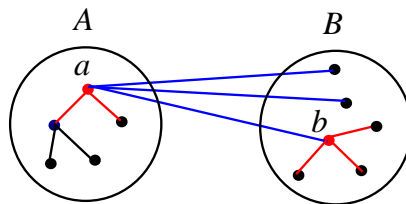


- Questions: How to compute cost reduction? What pairs to be swapped?
 - Consider the change of internal & external connections.

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

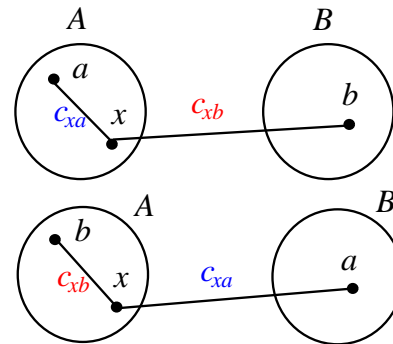
$$\begin{aligned} D'_x &= D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\} \\ D'_y &= D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\}. \end{aligned}$$



$\text{Gain}_{a \Rightarrow B}: D_a - c_{ab}$

$\text{Gain}_{b \Rightarrow A}: D_b - c_{ab}$

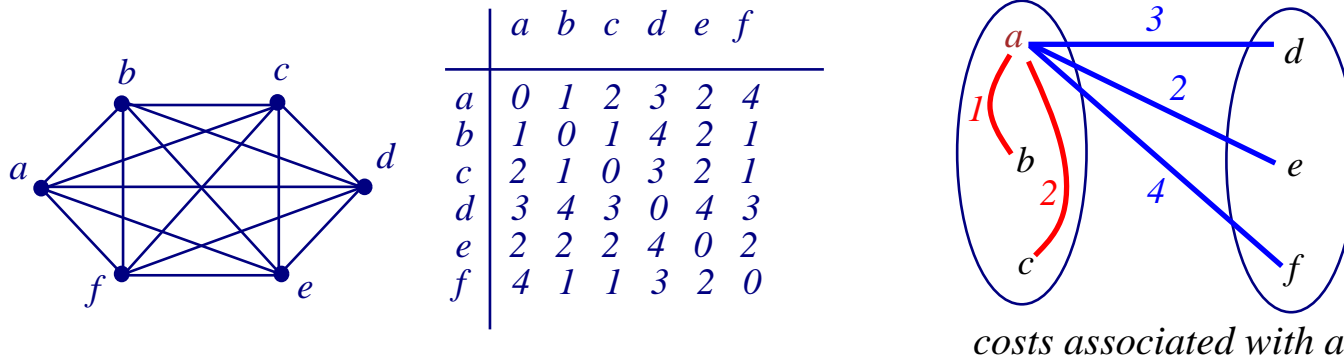
Internal cost vs. External cost



updating D -values

before swap	after swap	ΔC
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

Kernighan-Lin Algorithm: A Weighted Example



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

- Iteration 1:

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

Weighted Example (cont'd)

- Iteration 1:

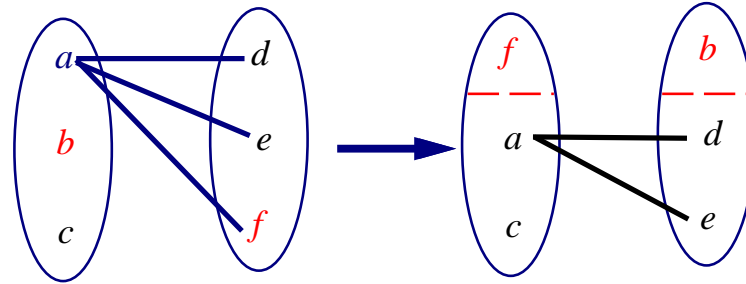
$$\begin{array}{lll}
 I_a = 1 + 2 = 3; & E_a = 3 + 2 + 4 = 9; & D_a = E_a - I_a = 9 - 3 = 6 \\
 I_b = 1 + 1 = 2; & E_b = 4 + 2 + 1 = 7; & D_b = E_b - I_b = 7 - 2 = 5 \\
 I_c = 2 + 1 = 3; & E_c = 3 + 2 + 1 = 6; & D_c = E_c - I_c = 6 - 3 = 3 \\
 I_d = 4 + 3 = 7; & E_d = 3 + 4 + 3 = 10; & D_d = E_d - I_d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

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

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

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

Weighted Example (cont'd)



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

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

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

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

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

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

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

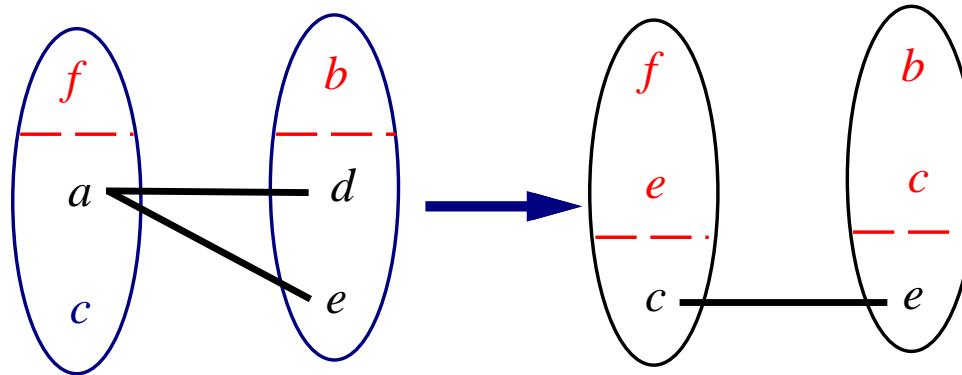
$$g_{ae} = D'_a + D'_e - 2c_{ae} = 0 + 0 - 2 \times 2 = -4$$

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

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

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

Weighted Example (cont'd)



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

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

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

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

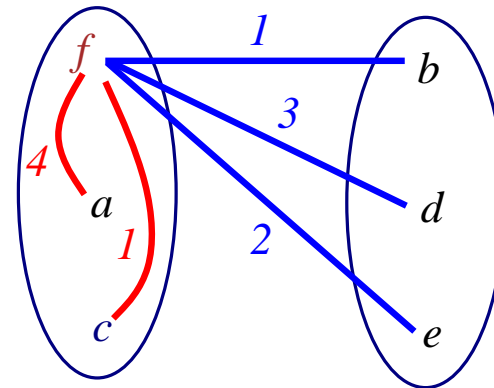
$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2 \times 3 = -3 (\hat{g}_3 = -3)$$

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

Weighted Example (cont'd)

	<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



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

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  $v_{ai} \in V_A$  and  $v_{bi} \in V_B$  whose
       exchange makes the largest decrease or smallest increase in
       cut cost;
7     Mark  $v_{ai}$  and  $v_{bi}$  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  $v_{a1}, \dots, v_{ak}$  from  $V_A$  to  $V_B$  and  $v_{b1}, \dots, v_{bk}$  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: $O(rn^3)$.

Extensions of K-L Algorithm

- **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.

A “Better” Implementation of K-L Algorithm

- Sort the D -values in a non-increasing order:

$$D_{a_1} \geq D_{a_2} \geq \dots \geq D_{a_n}$$

$$D_{b_1} \geq D_{b_2} \geq \dots \geq D_{b_n}$$

- Start with a_1 , compute $g_{a_1, b_i}, \forall b_i$
 Start with a_2 , compute $g_{a_2, b_i}, \forall b_i$

\vdots

whenever $D_{a_i} + D_{b_j} \leq$ Maximum gain found so far (Quit!).

- Partition $A = \{a, b, c\}$: $D_a = 6$; $D_b = 5$; $D_c = 3$;
 Partition $B = \{d, e, f\}$: $D_d = 3$; $D_f = 1$; $D_e = 0$;

Compute g 's

$$g_{ad} = 3 \rightarrow g_{af} = -1 \rightarrow g_{ae} = 2$$

$$g_{bd} = 0 \rightarrow g_{bf} = 4 \rightarrow g_{be} = 1$$

$$g_{cd} = 0 \rightarrow \text{No need to compute } g_{cf} \text{ (Quit!)}$$

$$\text{since } D_c + D_f \leq g_{bf} = 4.$$

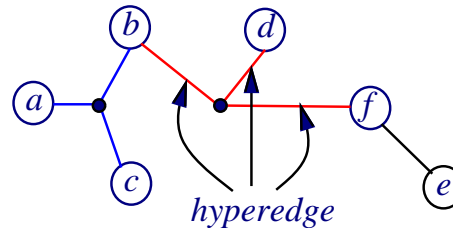
- Note that the overall time complexity remains $O(rn^3)$.

Drawbacks of the Kernighan-Lin Heuristic

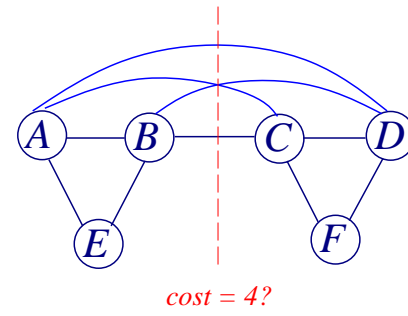
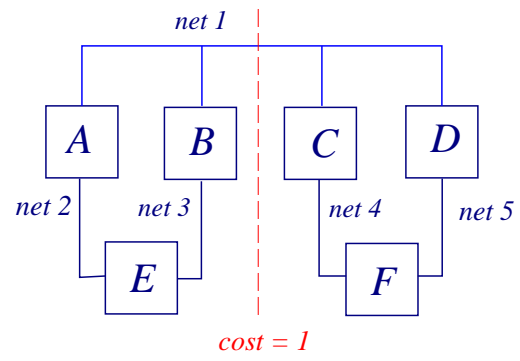
- The K-L heuristic **handles only unit vertex weights**.
 - Vertex weights might represent block sizes, different from blocks to blocks.
 - Reducing a vertex with weight $w(v)$ into a clique with $w(v)$ vertices and edges with a high cost increases the size of the graph substantially.
- The K-L heuristic **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**, $O(n^3)$.

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 $|N_i| \geq 2$.

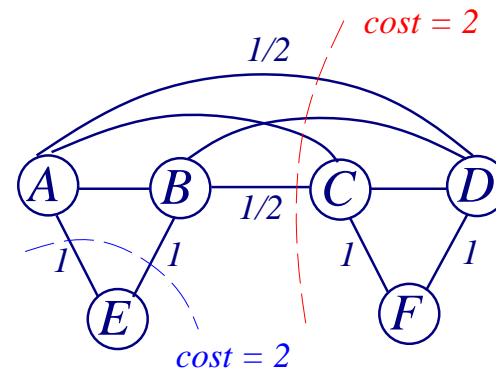
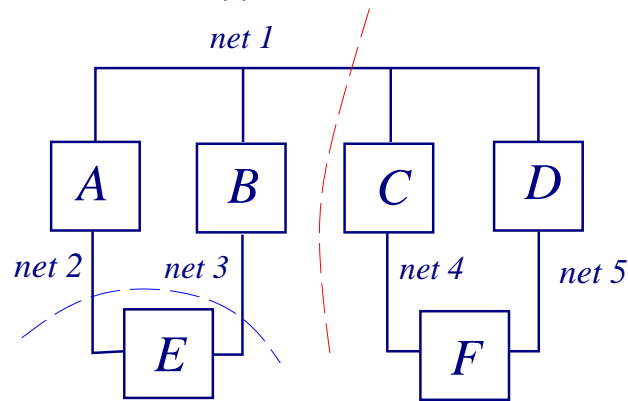


- Schweikert and Kernighan, "A proper model for the partitioning of electrical circuits," 9th Design Automation Workshop, 1972.
- For multi-terminal nets, **net cut** is a more accurate measurement for cut cost (i.e., deal with hyperedges).
 - $\{A, B, E\}, \{C, D, F\}$ is a good partition.
 - Should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

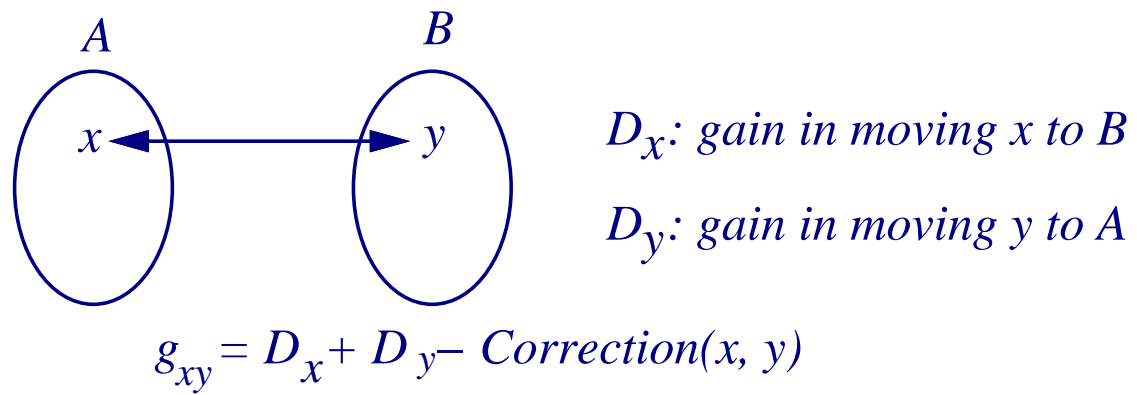


Net-Cut Model

- 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 .

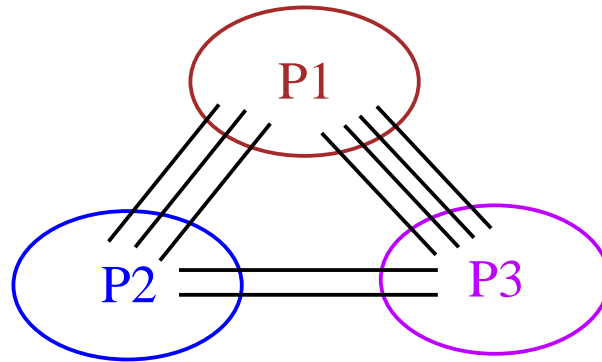


- Easy modification of the K-L heuristic.



Network Flow Based Partitioning

- Min-cut balanced partitioning: Yang and Wong, ICCAD-94.
 - Based on max-flow min-cut theorem.

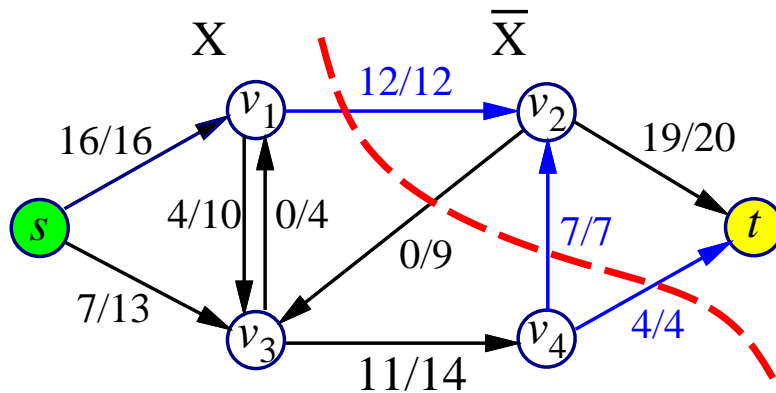


- Gate replication for partitioning: Yang and Wong, ICCAD-95.
- Performance-driven multiple-chip partitioning: Yang and Wong, FPGA'94, ED&TC-95.
- Multi-way partitioning with area and pin constraints: Liu and Wong, ISPD-97.
- Multi-resource partitioning: Liu, Zhu, and Wong, FPGA-98.
- Partitioning for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

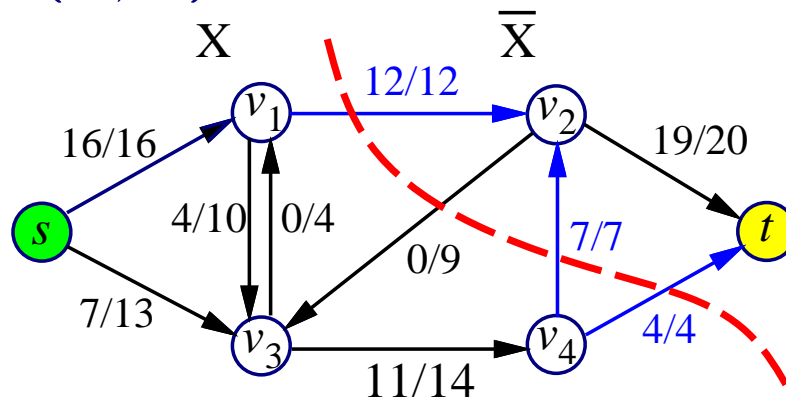


flow/capacity

$$\text{max flow } |f| = 16 + 7 = 23$$

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 $\iff |f| = cap(X, \bar{X})$ for some min-cut (X, \bar{X}) .

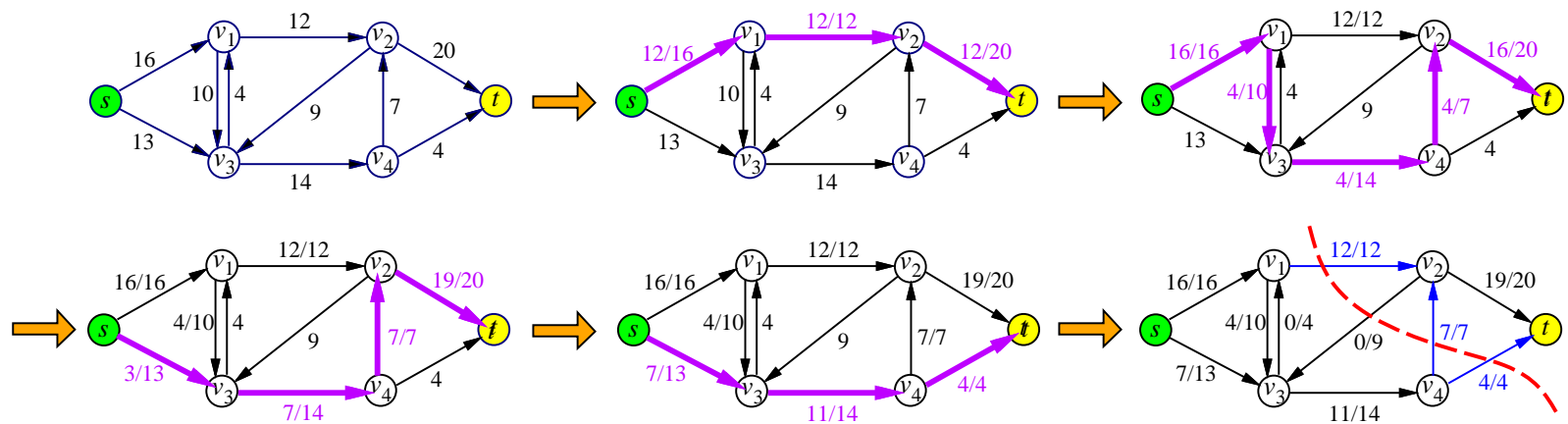


flow/capacity

$$\begin{aligned} \text{max flow } |f| &= 16 + 7 = 23 \\ \text{cap}(X, \bar{X}) &= 12 + 7 + 4 = 23 \end{aligned}$$

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 \iff no more augmenting path.



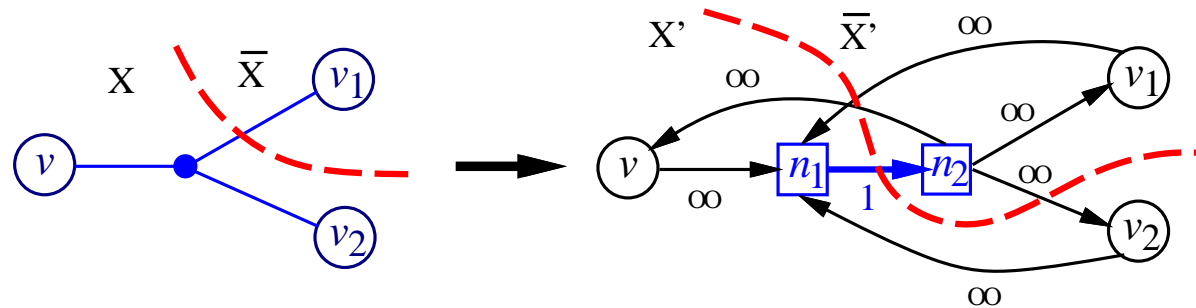
- First algorithm by Ford & Fulkerson in 1959: $O(|E||f|)$; First **polynomial-time** algorithm by Edmonds & Karp in 1969: $O(|E|^2|V|)$; Goldberg & Tarjan in 1985: $O(|E||V| \lg(|V|^2/|E|))$, etc.

Network Flow Based Partitioning

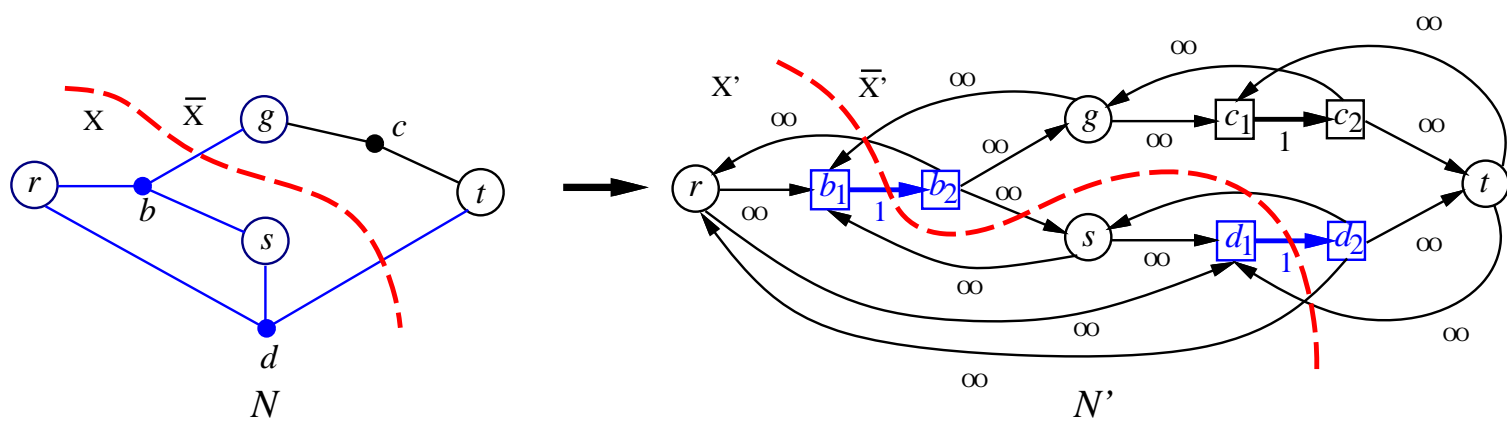
- Why was the technique not wisely used in partitioning?
 - Works on graphs, not hypergraphs.
 - Results in unbalanced partitions; repeated min-cut for balance: $|V|$ max-flows, time-consuming!
- Yang & Wong, ICCAD-94.
 - Exact **net** modeling by flow network.
 - Optimal algorithm for min-net-cut bipartition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

Min-Net-Cut Bipartition

- Net modeling by flow network:

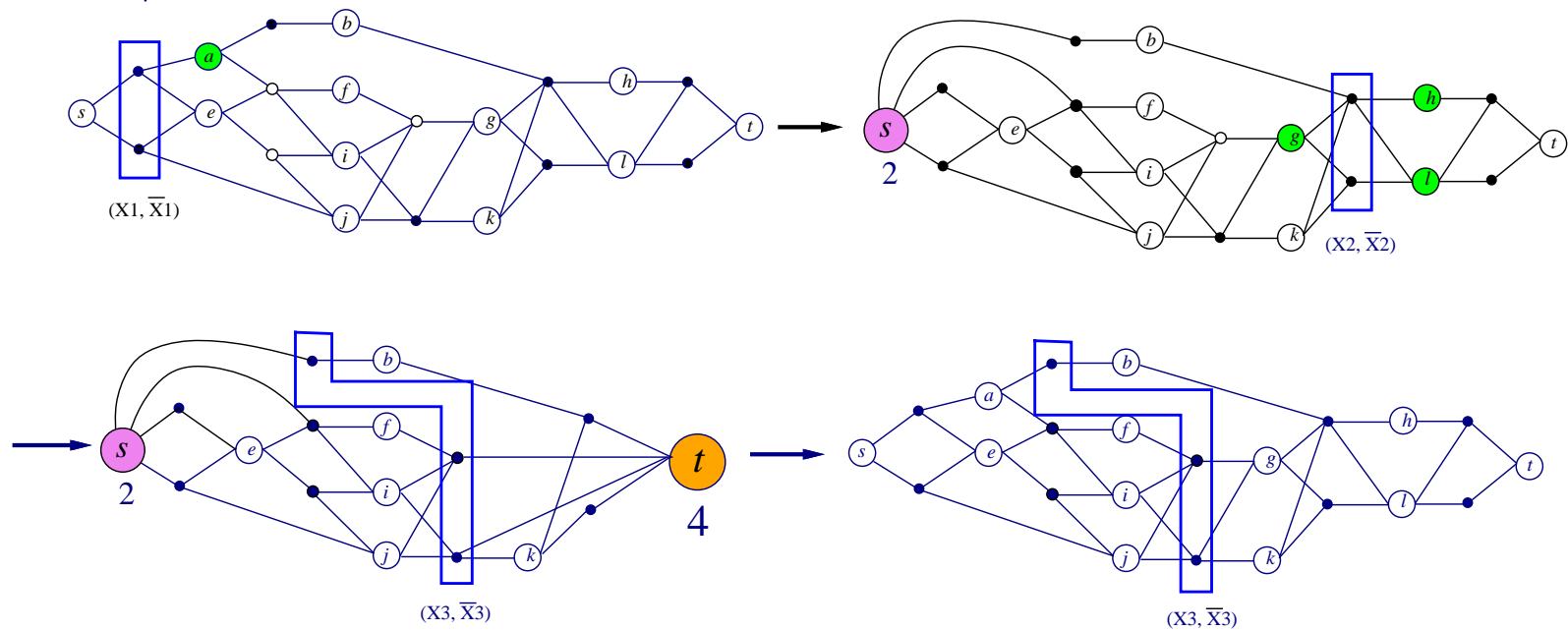


- A min-net-cut (X, \bar{X}) in $N \iff$ A min-capacity-cut (X', \bar{X}') in N' .
- Size of flow network: $|V'| \leq 3|V|$, $|E'| \leq 2|E| + 3|V|$.
- Time complexity: $O(\text{min-net-cut-size}) \times |E| = O(|V||E|)$.



Repeated Min-Cut for Balanced Bipartition (FBB)

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



○ An un-saturated net

● A saturated net

● A node to be collapsed to s or t

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: $O(|V||E|)$, same as **one** max-flow.
 - At most $2|V|$ augmenting path computations.
 - * At each augmenting path computation, either an augmenting path is found, or a new cut is found, and at least 1 node is collapsed to s or t .
 - * At most $|f| \leq |V|$ augmenting paths found, since bridging edges have unit capacity.

– An augmenting path computation: $O(|E|)$ time.

