Clustering

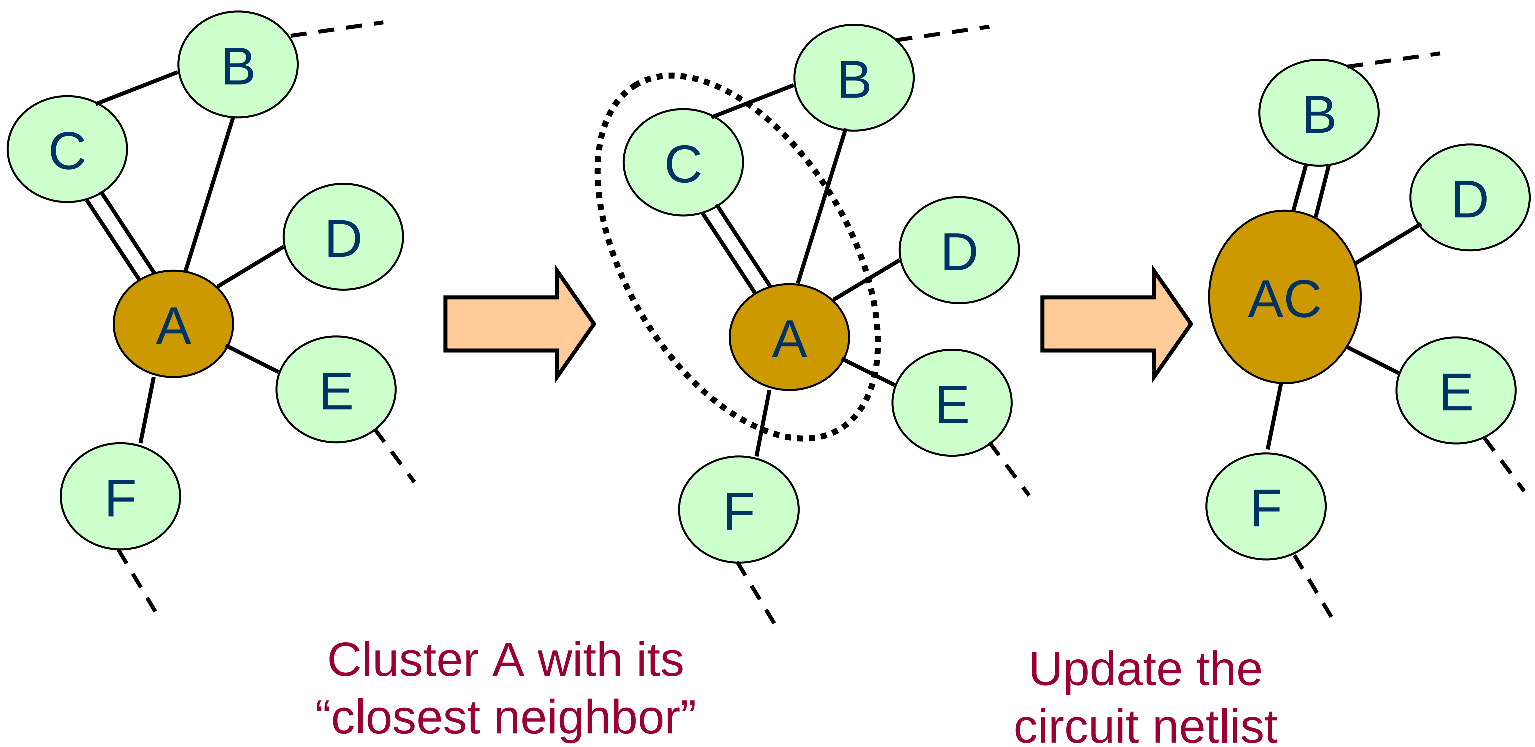
ECE6133

Physical Design Automation of VLSI Systems

Prof. Sung Kyu Lim
School of Electrical and Computer Engineering
Georgia Institute of Technology

Circuit Clustering

- Grouping cells to form bigger cells
  - Why do we do this?



Circuit Clustering

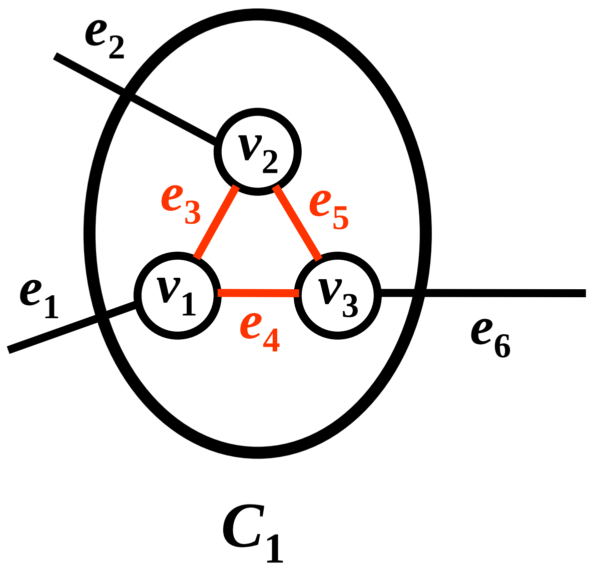
- Motivation
  - Reduce the size of flat netlists
  - Identify natural circuit hierarchy
- Objectives
  - Maximize the connectivity of each cluster
  - Minimize the size, delay, and density of clustered circuits

Clustering *vs* Partitioning

- Differences and similarities
  - Divide cells into groups under area constraint *A*
  - Clustering if *A* is small; partitioning otherwise
  - Clustering = pre-process of partitioning
- Clustering Metrics
  - Absorption, Density, Rent Parameter, Ratio Cut, Closeness, Connectivity, etc….
- Partitioning Metrics
  - Cutsize and delay

Density Metric

- Desire high "density" in each cluster
  - Applied to a single cluster



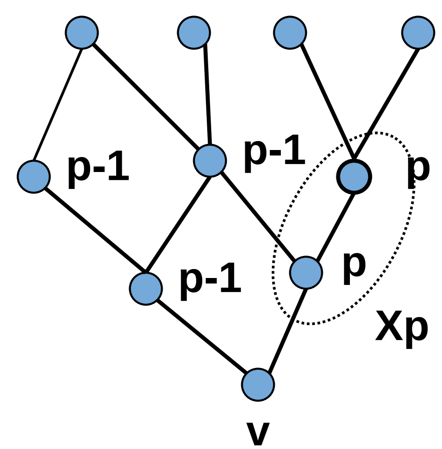
$$DEN(C_1) = \sum_{e \in C_1} w(e) / \sum_{v \in C_1} s(v) = \frac{w(e_3) + w(e_4) + w(e_5)}{s(v_1) + s(v_2) + s(v_3)}$$

Previous Works

- Cutsize-oriented
  - (K, I)-connectivity algorithms [Garber-Promel-Steger 1990]
  - Random-walk based algorithm [Cong et al 1991; Hagen-Kahng 1992]
  - Multicommodity-Flow based algorithm [Yeh-Cheng-Lin 1992]
  - Clique based algorithm [Bui 1989; Cong-Smith 1993]
  - Multi-level clustering [Karypis-Kumar, DAC97; Cong-Lim, ASPDAC'00]
- Delay-oriented
  - For combinational circuits: [Lawler-Levitt-Turner 1969; Murgai-Brayton-Sanjiovanni 1991; Rajaraman-Wong 1995; Cong-Ding 1992]
  - For sequential circuits: [Pan et al, TCAD'99; Cong et al, DAC'99]
  - Signal flow based clustering [Cong-Ding, DAC'93; Cong et al ICCAD'97]

Lawler's Labeling Algorithm

- Assumption:
  - Cluster size ≤ K; intra-cluster delay = 0; inter-cluster delay = 1
- Objective: Find a clustering of minimum delay
- Phase 1: Label all nodes in topological order
  - For each PI node *v*, *L*(*v*)= 0;
  - For each non-PI node *v*
    - *p* = maximum label of predecessors of *v*
    - *Xp* = set of predecessors of *v* with label *p*
    - if |*Xp*| < *K* then *L*(*v*) = *p*; else *L*(*v*) = *p*+1



- Phase 2: Form clusters
  - Start from PO to generate necessary clusters
  - Nodes with the same label form a cluster

Rajaraman-Wong Algorithm

- First optimal algorithm that solves delay-oriented clustering problem under general delay model
- Given
  - DAG, cluster size limit
- Find
  - Optimal clustering that minimizes maximum PI-PO path delay
- Delay model
  - Node delay = d, intra-cluster delay = 0; inter-cluster delay = D
  - Better than "unit delay model" used in Lawler
- Node duplication is allowed

Rajaraman-Wong Algorithm

- Initialization phase
  - Compute $n \times n$ matrix $\Delta(x,v)$: all-pair max-delay value from output of x to output of v, using node delay only
  - Set label(PI) = delay(PI), label(non-PI) = 0
- Labeling Phase
  - Compute label based on topological order of the nodes
  - Label denotes max delay from any PI to the node
  - Clustering info is also computed during labeling
- Clustering Phase
  - Actual grouping and duplication occur
  - Done based on reserve topological order

Labeling for Node v

1 We compute the sub-graph rooted at v, denoted G_v, that includes all the predecessors of v.

2 We compute $l_v(x)$ for each node $x \in G_v \backslash \{v\}$, where $l_v(x) = l(x) + \Delta(x, v)$. $l(x)$ denotes the current label for x, and $\Delta(x, v)$ is an entry of the Δ matrix mentioned above.

3 We sort all nodes in $G_v \backslash \{v\}$ in decreasing order of their l_v-values and put them into a set S.

4 We remove a node from S one-by-one in the sorted order and add it to the cluster for v, denoted $cluster(v)$, until the size constraint is violate.

5 We compute two values l_1 and l_2. If $cluster(v)$ contains any PI nodes, the maximum l_v value among these PI nodes becomes l_1. If S is not empty after filling up $cluster(v)$, the maximum $l_v + D$ among the nodes remaining in S becomes l_2, where D is the inter-cluster delay.

6 The new label for v is the maximum between l_1 and l_2.

What is going on?

v). By the definition of ℓ_v, $\ell_v(u)$ is a lower bound on the delay along any path from a primary input to v that passes through u. The greater the value of $\ell_v(u)$, the more the need to include u in $cluster(v)$. Hence, we try to cluster v with as many high ℓ_v-valued nodes as the capacity constraint permits. After building $cluster(v)$, v is labeled by considering all possible paths from an input to the output of v. All of the paths can be divided into two categories:

1) Paths that lie entirely in $cluster(v)$. Such paths start from a primary input that is in $cluster(v)$, and never exit the cluster. The maximum delay along any such path is

$$\ell_1(v) = \max\{\ell_v(u) \mid u \in cluster(v) \cap \mathcal{PI}\}. \quad (1)$$

2) Paths that cross the "boundary" of $cluster(v)$. Among these paths, the maximum delay is

$$\ell_2(v) = \max\{\ell_v(u) + D \mid u \in G_v \backslash cluster(v)\}. \quad (2)$$

Clustering Phase

1 We first put all PO nodes in a set L. We then remove a node from L and form its cluster.

2 Given a node v, we form a cluster by grouping all nodes in $cluster(v)$, which was computed during the labeling phase.

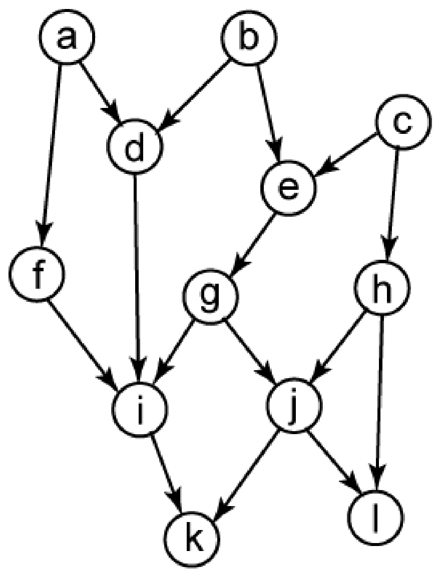
3 We then compute $input(v)$, the set of input nodes of $cluster(v)$.

4 We remove a node x from $input(v)$ one-by-one and add it to L if we have not formed the cluster for x yet.

5 We repeat the entire process until L becomes empty.

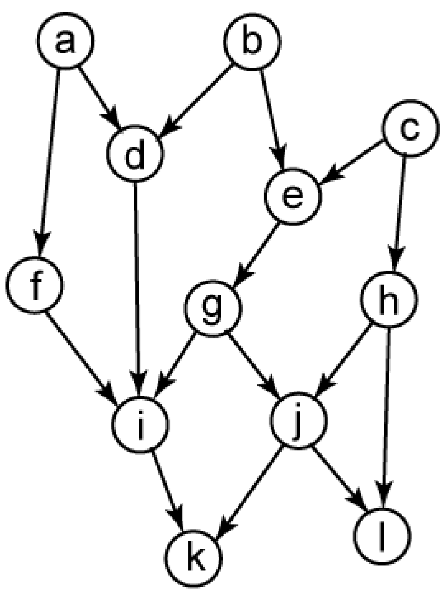
Rajaraman-Wong Algorithm

- Perform RW clustering on the following di-graph.
  - Inter-cluster delay = 3, node delay = 1
  - Size limit = 4
  - Topological order *T* = [*d*,*e*,*f*,*g*,*h*,*i*,*j*,*k*,*l*] (not unique)



Max Delay Matrix

- All-pair delay matrix $\Delta(x,y)$
  - Max delay from output of the PIs to output of destination



	a	b	c	d	e	f	g	h	i	j	k	l
a	0	0	0	1	0	1	0	0	2	0	3	0
b	0	0	0	1	1	0	2	0	3	3	4	4
c	0	0	0	0	1	0	2	1	3	3	4	4
d	0	0	0	0	0	0	0	0	1	0	2	0
e	0	0	0	0	0	0	1	0	2	2	3	3
f	0	0	0	0	0	0	0	0	1	0	2	0
g	0	0	0	0	0	0	0	0	1	1	2	2
h	0	0	0	0	0	0	0	0	0	1	2	2
i	0	0	0	0	0	0	0	0	0	0	1	0
j	0	0	0	0	0	0	0	0	0	0	1	1
k	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0

Label and Clustering Computation

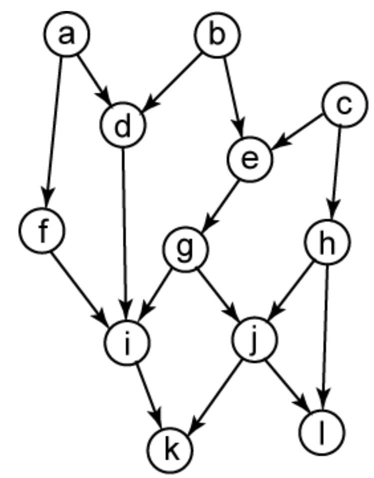
- Compute *l*(*d*) and *cluster*(*d*)

First, $G_d = \{a, b, d\}$. By definition $l(a) = l(b) = 1$. Thus,

$$l_d(a) = l(a) + \Delta(a, d) = 1 + 1 = 2$$
$$l_d(b) = l(b) + \Delta(b, d) = 1 + 1 = 2$$

Then we have $S = \{a, b\}$ (recall that S contains $G_d \backslash \{d\}$ with their l_d values sorted in a decreasing order). Since both a and b can be clustered together with d while not violating the size constraint of 4, we form

$$cluster(d) = \{a, b, d\}$$



Since both a and b are PI nodes, we see that

$$l_1 = \max\{l_d(a), l_d(b)\} = 2$$

Since S is empty after clustering, l_2 remains zero. Thus,

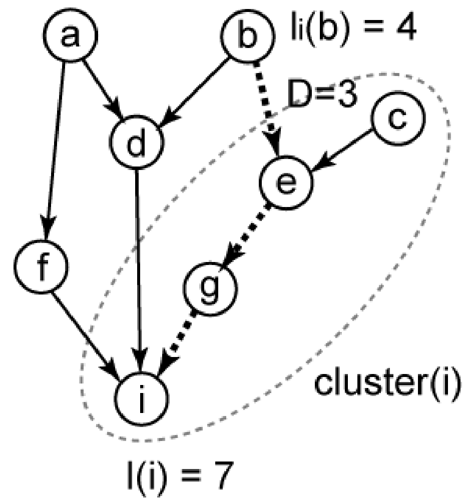
$$l(d) = \max\{l_1, l_2\} = 2$$

Label Computation

- Compute *l*(*i*) and *cluster*(*i*)

node i: $G_i = \{a, b, c, d, e, f, g, i\}$ (see Figure 1.3). Thus,

$$\begin{aligned} l_i(a) &= l(a) + \Delta(a,i) = 1 + 2 = 3 \\ l_i(b) &= l(b) + \Delta(b,i) = 1 + 3 = 4 \\ l_i(c) &= l(c) + \Delta(c,i) = 1 + 3 = 4 \\ l_i(d) &= l(d) + \Delta(d,i) = 2 + 1 = 3 \\ l_i(e) &= l(e) + \Delta(e,i) = 2 + 2 = 4 \\ l_i(f) &= l(f) + \Delta(f,i) = 2 + 1 = 3 \\ l_i(g) &= l(g) + \Delta(g,i) = 3 + 1 = 4 \end{aligned}$$

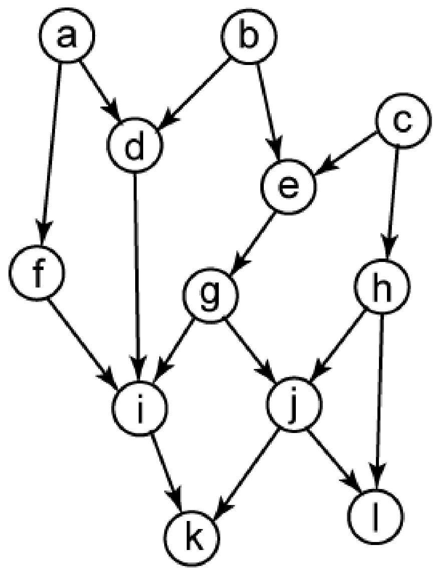


$S = \{g, e, c, b, a, d, f\}$, and we form $cluster(i) = \{i, g, e, c\}$.[1] Note that c is PI, so $l_1 = l_i(c) = 4$. Since $S = \{b, a, d, f\} \neq \emptyset$ after clustering, we have $l_2 = l_i(m(S)) + D = l_i(b) + D = 4 + 3 = 7$ (recall that $m(S)$ is the node in S with the maximum value of l_i value). Thus, $l(i) = \max\{l_1, l_2\} = 7$.

Labeling Summary

- Labeling phase generates the following information.
  - Max label = max delay= 8

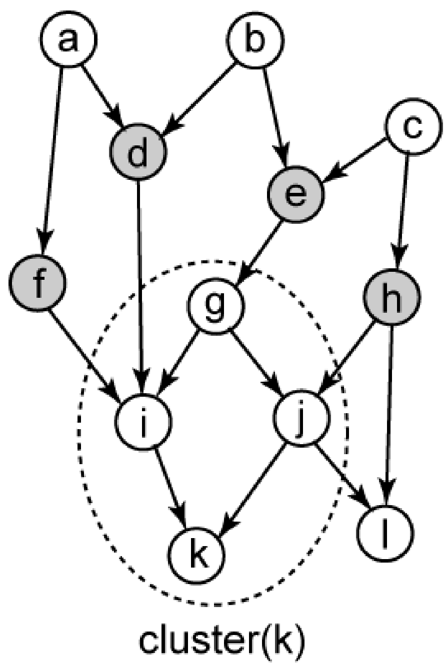
node	label	clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	2	$\{a, b, d\}$
e	2	$\{b, c, e\}$
f	2	$\{a, f\}$
g	3	$\{b, c, e, g\}$
h	2	$\{c, h\}$
i	7	$\{c, e, g, i\}$
j	7	$\{b, e, g, j\}$
k	8	$\{g, i, j, k\}$
l	8	$\{e, g, j, l\}$



Clustering Phase

- Initially L = POs = $\{k,l\}$.

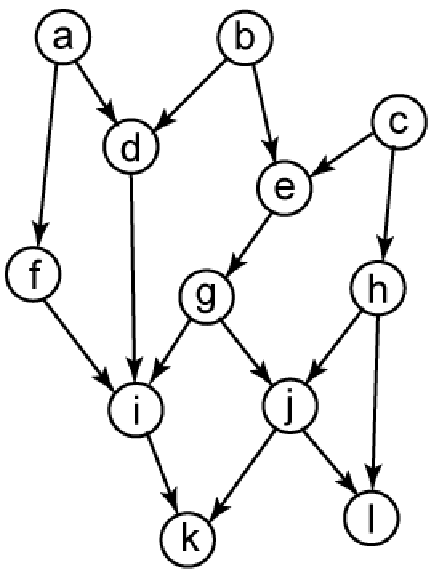
remove k from L, and add $cl(k)$ to $S = \{cl(k)\}$. According to Table 1.1, we see that $cl(k) = \{g, i, j, k\}$. Then, $I[cl(k)] = \{f, d, e, h\}$ as illustrated in Figure 1.4. Since S does not contain clusters rooted at f, d, e, and h, we have $L = \{l\} \cup \{f, d, e, h\} = \{l, f, d, e, h\}$.



Clustering Summary

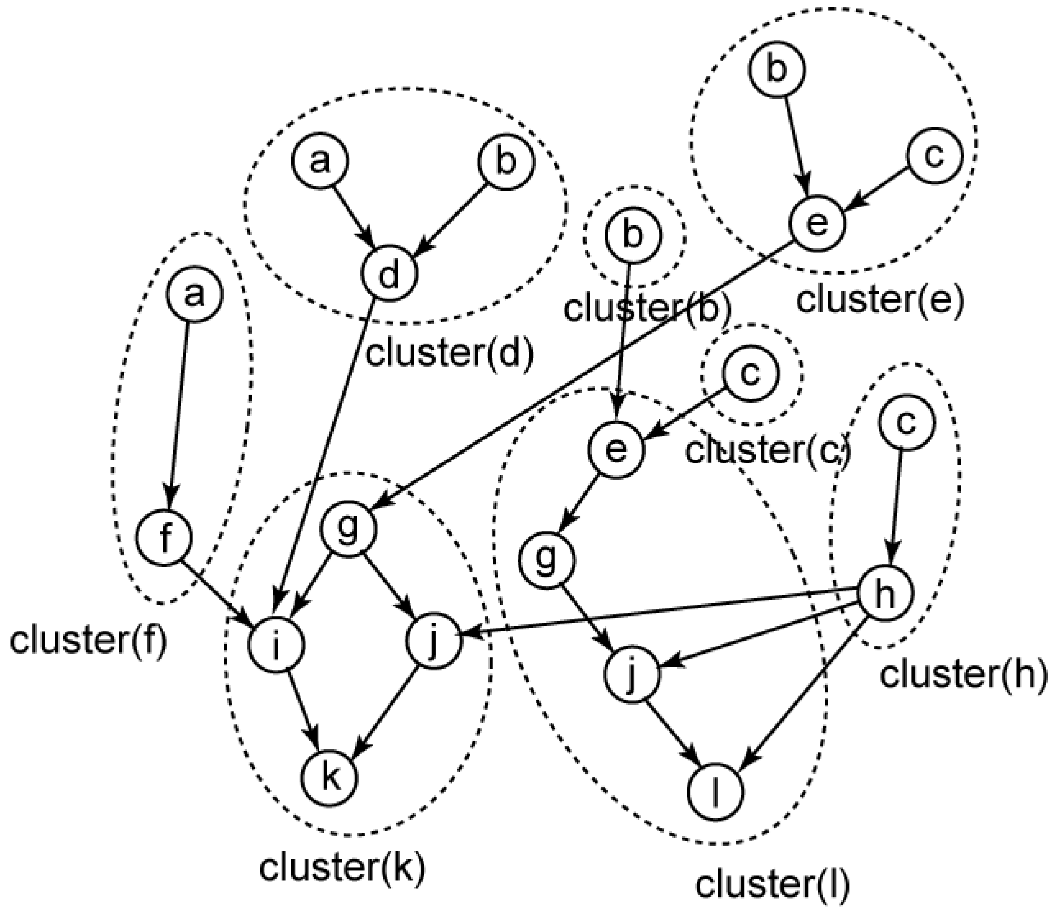
- Clustering phase generates 8 clusters.
  - 8 nodes are duplicated

root	elements
k	$\{g, i, j, k\}$
l	$\{e, g, j, l\}$
f	$\{a, f\}$
d	$\{a, b, d\}$
e	$\{b, c, e\}$
h	$\{c, h\}$
b	$\{b\}$
c	$\{c\}$



Final Clustering Result

- Path *c*-*e*-*g*-*i*-*k* has delay 8 (= max label)

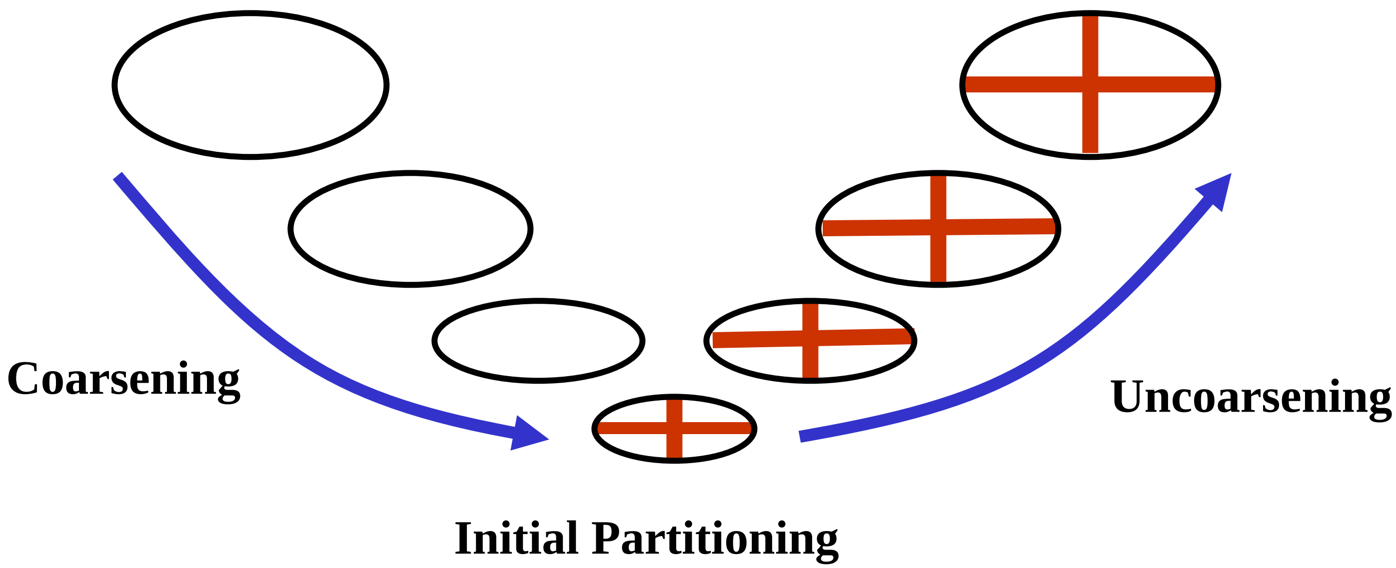


Probing Further

- Rajaraman-Wong Algorithm
  - [Yang and Wong, 1994]: finds set of nodes to be replicated so that cutsize is minimized
  - [Vaishnav and Pedram, 1995]: minimizes power under delay-optimal clustering properties
  - [Yang and Wong, 1997]: performed delay-optimal clustering under area and/or pin constraint
  - [Pan et at, 1998]: performed delay-optimal clustering with retiming for sequential circuits
  - [Cong and Romesis, 2001]: developed heuristic for two-level delay-oriented clustering problem

Multi-level Paradigm

- Combination of Bottom-up and Top-down Methods
  - From coarse-grain into finer-grain optimization
  - Successfully used in partial differential equations, image processing, combinatorial optimization, etc, and circuit partitioning.



General Framework

- **Step 1: Coarsening**
  - Generate hierarchical representation of the netlist
- **Step 2: Initial Solution Generation**
  - Obtain initial solution for the top-level clusters
  - Reduced problem size: converge fast
- **Step 3: Uncoarsening and Refinement**
  - Project solution to the next lower-level (uncoarsening)
  - Perturb solution to improve quality (refinement)
- **Step 4: V-cycle**
  - Additional improvement possible from new clustering
  - Iterate Step 1 (with variation) + Step 3 until no further gain

V-cycle Refinement

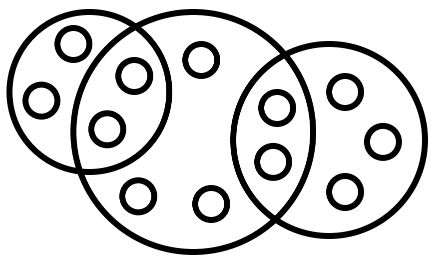
- **Motivation**
  - **Post-refinement scheme for multi-level methods**
  - **Different clustering can give additional improvement**
- **Restricted Coarsening**
  - **Require initial partitioning**
  - **Do not merge clusters in different partition**
  - **Maintain cutline: cutsize degradation is not possible**
- **Two Strategies: V-cycle vs. v-cycle**
  - **V-cycle: start from the bottom-level**
  - **v-cycle: start from some middle-level**
  - **Tradeoff between quality vs. runtime**

Application in Partitioning

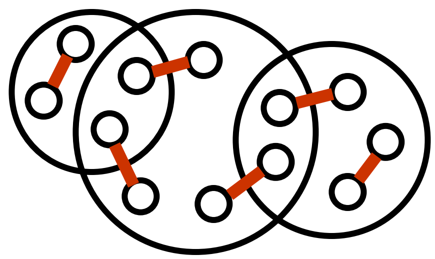
- **Multi-level Partitioning**
  - **Coarsening engine (bottom-up)**
    - **Unrestricted and restricted coarsening**
    - **Any bottom-up clustering algorithm can be used**
    - **Cutsize oriented (MHEC, ESC) vs. delay oriented (PRIME)**
  - **Initial partitioning engine**
    - **Move-based methods are commonly used**
  - **Refinement engine (top-down)**
    - **Move-based methods are commonly used**
    - **Cutsize oriented (FM, LR) vs. delay oriented (xLR)**
- **State-of-the-art Algorithms**
  - **hMetis [DAC97] and hMetis-Kway [DAC99]**

hMetis Algorithm

- **Best Bipartitioning Algorithm [DAC97]**
  - **Contribution: 3 new coarsening schemes for hypergraphs**



Original Graph



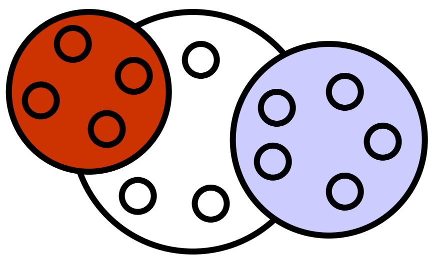
Edge Coarsening

Edge Coarsening = heavy-edge maximal matching

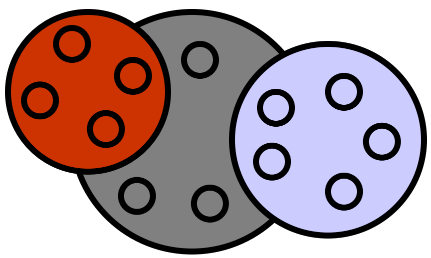
1. Visit vertices randomly
2. Compute edge-weights (=1/(|*n*|-1)) for all unmatched neighbors
3. Match with an unmatched neighbor via max edge-weight

hMetis Algorithm (cont)

- Best Bipartitioning Algorithm [DAC97]
  - Contribution: 3 new coarsening schemes for hypergraphs



Hyperedge Coarsening



Modified Hyperedge Coarsening

Hyperedge Coarsening = independent hyperedge merging

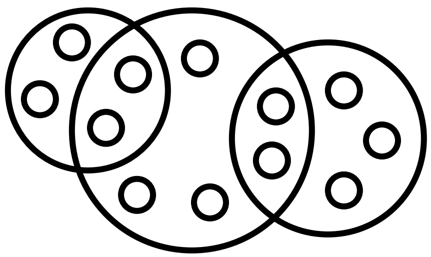
1. Sort hyperedges in non-decreasing order of their size
2. Pick an hyperedge with no merged vertices and merge

Modified Hyperedge Coarsening = Hyeredge Coarsening + post process

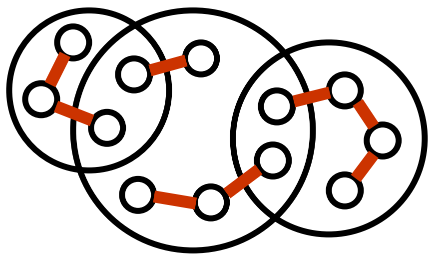
1. Perform Hyperedge Coarsening
2. Pick a non-merged hyperedge and merge its non-merged vertices

hMetis-Kway Algorithm

- Multiway Partitioning Algorithm [DAC99]
  - New coarsening: First Choice (variant of Edge Coarsening)
    - Can match with either unmatched or matched neighbors



Original Graph

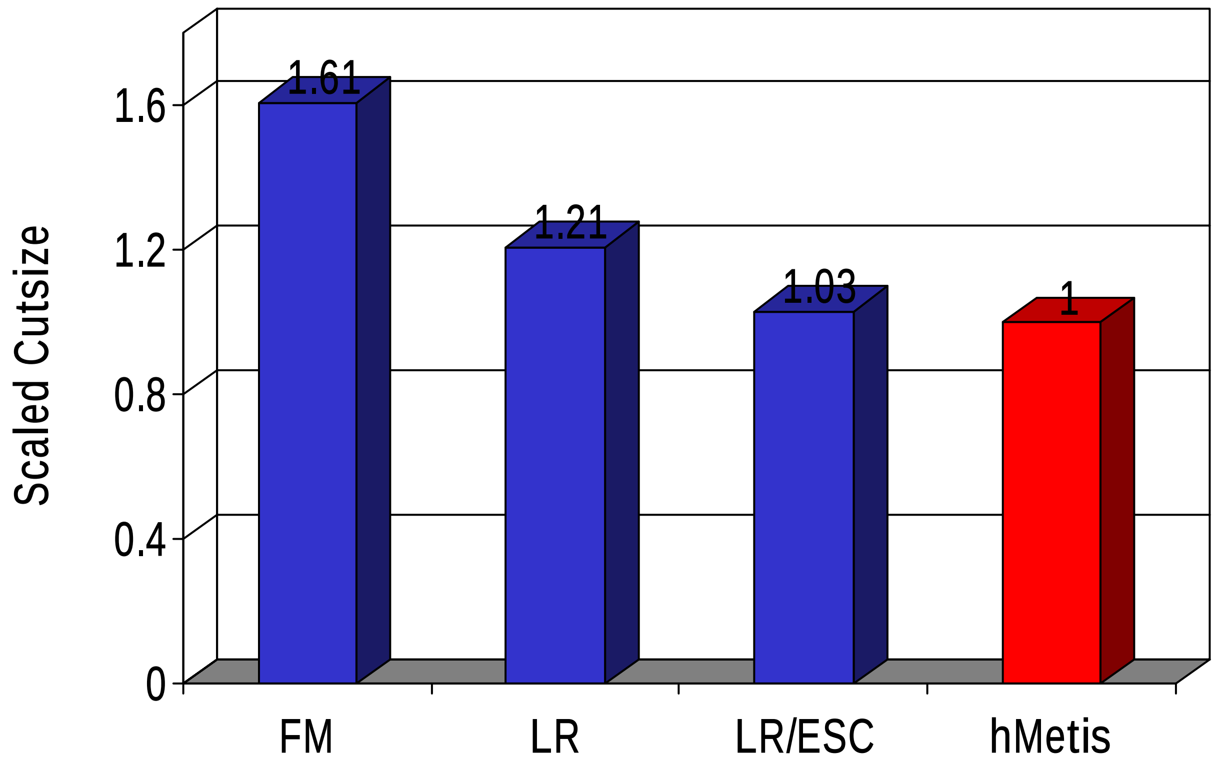


First Choice

  - Greedy refinement
    - On-the-fly gain computation
    - No bucket: not necessarily the max-gain cell moves
    - Save time and space requirements

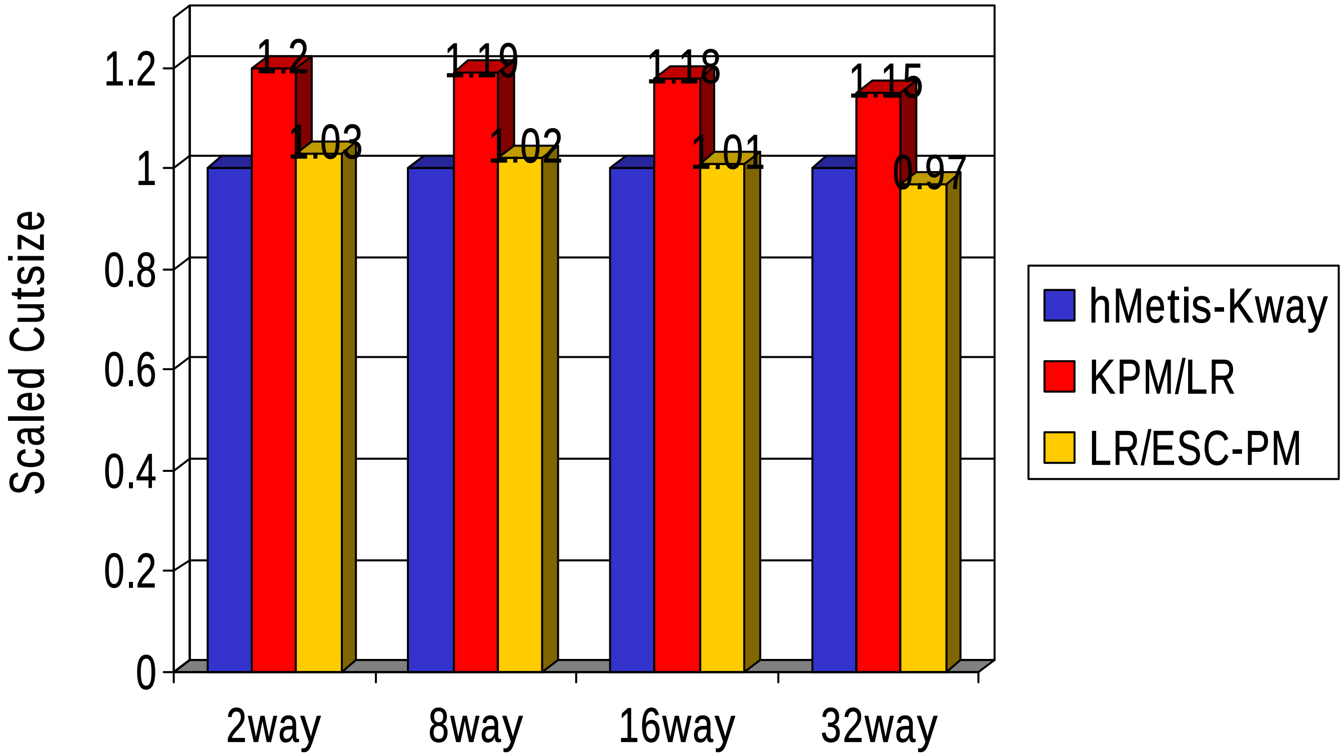
hMetis Results

- **Bipartitioning on ISPD98 Benchmark Suite**



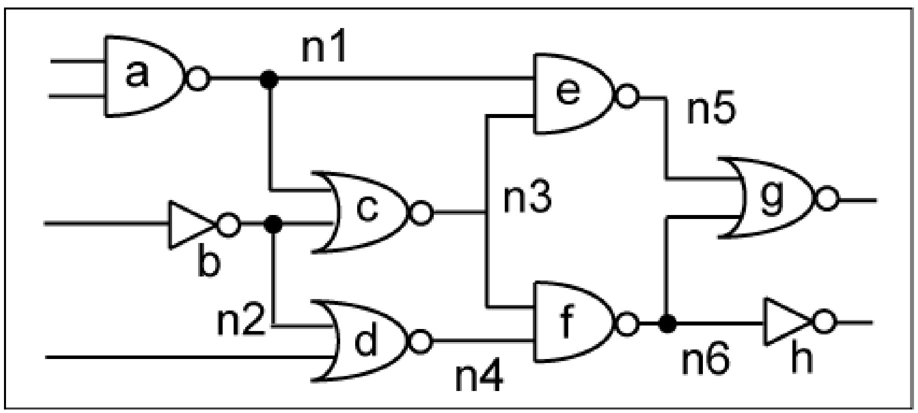
hMetis-Kway Results

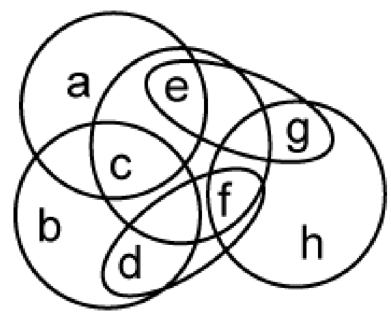
- **Multiway Partitioning on ISPD98 Benchmark Suite**



Multi-level Coarsening Algorithm

- Perform Edge Coarsening (EC)
  - Visit nodes and break ties in alphabetical order
  - Explicit clique-based graph model is not necessary



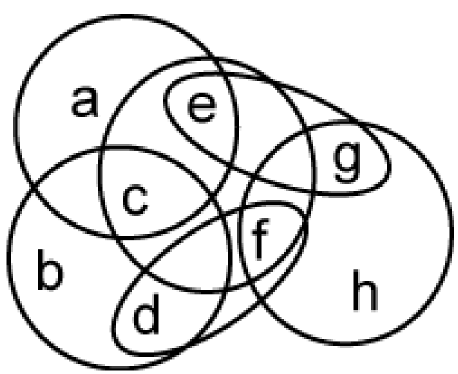


Edge Coarsening

(a) visit a: Note that a is contained in n_1 only. So, $neighbor(a) = \{c, e\}$. The weight of $(a, c) = 1/(|n_1| - 1) = 0.5$. The weight of $(a, e) = 1/(|n_1| - 1) = 0.5$. Thus, we break the tie based on alphabetical order. So, a merges with c. We form $C_1 = \{a, c\}$ and mark a and c.

(b) visit b: Note that b is contained in n_2 only. So, $neighbor(b) = \{c, d\}$. Since c is already marked, b merges with d. We form $C_2 = \{b, d\}$ and mark b and d.

(c) since c and d are marked, we skip them.



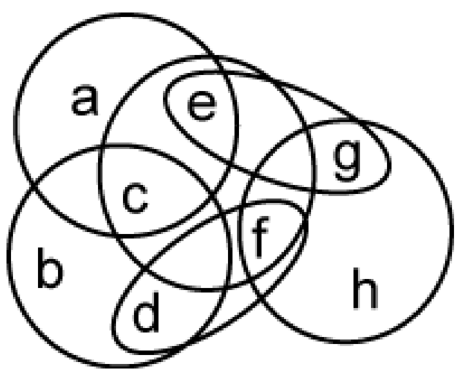
cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$

Edge Coarsening (cont)

(d) visit e: the unmarked neighbors of e are g and f. We see that $w(e, g) = 1$ and $w(e, f) = 0.5$. So, e merges with g. We form $C_3 = \{e, g\}$ and mark e and g.

(e) visit f: Node f is contained in n_3, n_4, and n_6. So, $neighbor(f) = \{c, d, e, g, h\}$. But, the only unmarked neighbor is h. So, f merges with h. We form $C_4 = \{f, h\}$ and mark f and h.

(f) since g and h are marked, we skip them.



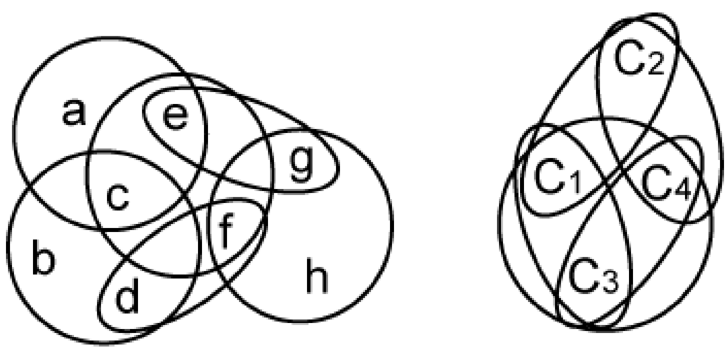
cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$

Obtaining Clustered-level Netlist

- # of nodes/hyperedges reduced: 4 nodes, 5 hyperedges

net	gate-level	cluster-level	final
n_1	$\{a, c, e\}$	$\{C_1, C_1, C_3\}$	$\{C_1, C_3\}$
n_2	$\{b, c, d\}$	$\{C_2, C_1, C_2\}$	$\{C_1, C_2\}$
n_3	$\{c, e, f\}$	$\{C_1, C_3, C_4\}$	$\{C_1, C_3, C_4\}$
n_4	$\{d, f\}$	$\{C_2, C_4\}$	$\{C_2, C_4\}$
n_5	$\{e, g\}$	$\{C_3, C_3\}$	$\emptyset$
n_6	$\{f, g, h\}$	$\{C_4, C_3, C_4\}$	$\{C_3, C_4\}$

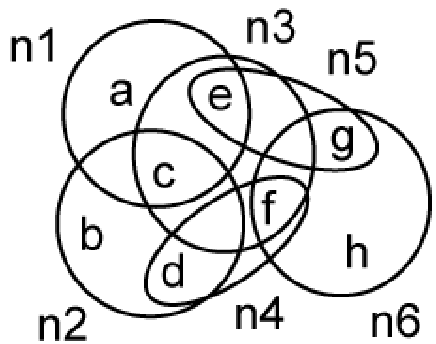
cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$



(a) (b)

Hyperedge Coarsening

- Initial setup
  - Sort hyper-edges in increasing size: n_4, n_5, n_1, n_2, n_3, n_6
  - Unmark all nodes

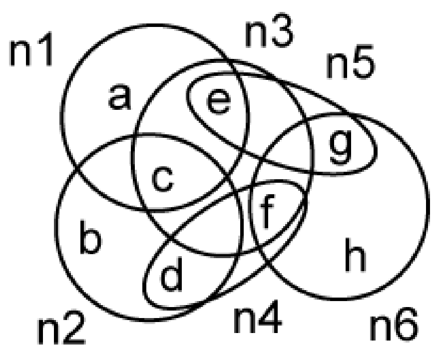


Hyperedge Coarsening

(a) visit $n_4 = \{d, f\}$: since d and f are not marked yet, we form $C_1 = \{d, f\}$ and mark d and f.

(b) visit $n_5 = \{e, g\}$: since e and g are not marked yet, we form $C_2 = \{e, g\}$ and mark e and g.

(c) visit $n_1 = \{a, c, e\}$: since e is already marked, we skip n_1.



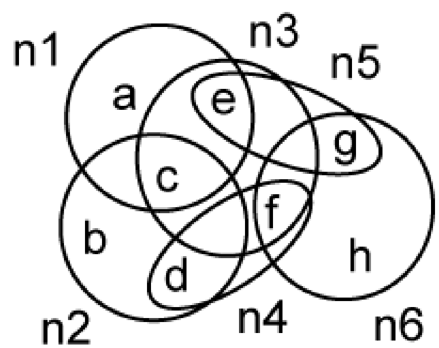
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$

Hyperedge Coarsening

(d) visit $n_2 = \{b, c, d\}$: since d is already marked, we skip n_2.

(e) visit $n_3 = \{c, e, f\}$: since e and f are already marked, we skip n_3.

(f) visit $n_6 = \{f, g, h\}$: since f and g are already marked, we skip n_6.



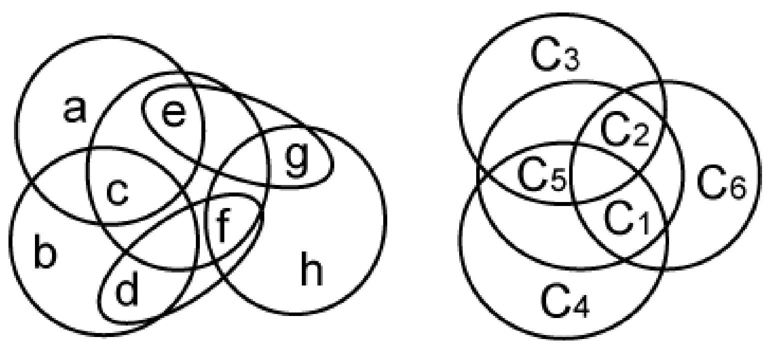
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$

Obtaining Clustered-level Netlist

- # of nodes/hyperedges reduced: 6 nodes, 4 hyperedges

net	gate-level	cluster-level	final
n_1	$\{a, c, e\}$	$\{C_3, C_5, C_2\}$	$\{C_3, C_5, C_2\}$
n_2	$\{b, c, d\}$	$\{C_4, C_5, C_1\}$	$\{C_4, C_5, C_1\}$
n_3	$\{c, e, f\}$	$\{C_5, C_2, C_1\}$	$\{C_5, C_2, C_1\}$
n_4	$\{d, f\}$	$\{C_1, C_1\}$	$\emptyset$
n_5	$\{e, g\}$	$\{C_2, C_2\}$	$\emptyset$
n_6	$\{f, g, h\}$	$\{C_1, C_2, C_6\}$	$\{C_1, C_2, C_6\}$

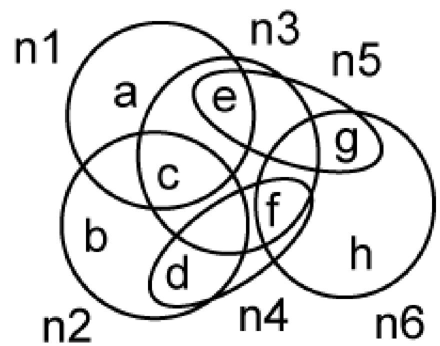
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$



(a) (b)

Modified Hyperedge Coarsening

- Revisit skipped nets during hyperedge coarsening
  - We skipped n_1, n_2, n_3, n_6
  - Coarsen un-coarsened nodes in each net



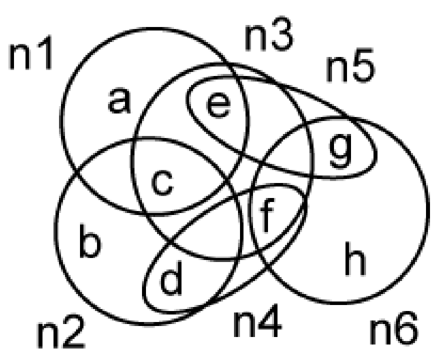
Modified Hyperedge Coarsening

(a) visit $n_1 = \{a, c, e\}$: since e is already marked during HEC, we group the remaining unmarked nodes a and c. We form $C_3 = \{a, c\}$ and mark a and c.

(b) visit $n_2 = \{b, c, d\}$: since d is marked during HEC and c during MHEC as above, we form $C_4 = \{b\}$ and mark b.

(c) visit $n_3 = \{c, e, f\}$: all nodes are already marked, so we skip n_3.

(d) visit $n_6 = \{f, g, h\}$: since f and g are already marked, we form $C_5 = \{h\}$ and mark h.



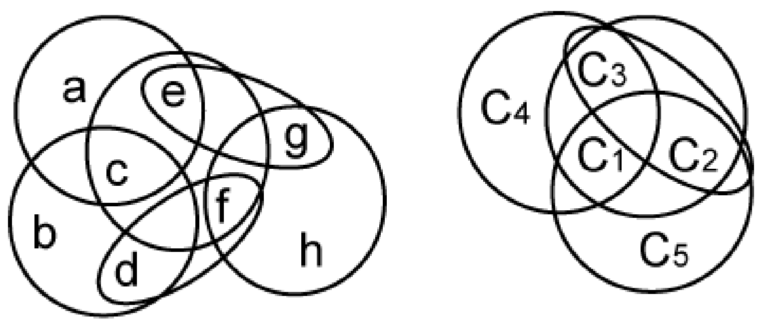
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a, c\}$
C_4	$\{b\}$
C_5	$\{h\}$

Obtaining Clustered-level Netlist

- # of nodes/hyperedges reduced: 5 nodes, 4 hyperedges

net	gate-level	cluster-level	final
n_1	$\{a, c, e\}$	$\{C_3, C_3, C_2\}$	$\{C_3, C_2\}$
n_2	$\{b, c, d\}$	$\{C_4, C_3, C_1\}$	$\{C_4, C_3, C_1\}$
n_3	$\{c, e, f\}$	$\{C_3, C_2, C_1\}$	$\{C_3, C_2, C_1\}$
n_4	$\{d, f\}$	$\{C_1, C_1\}$	$\emptyset$
n_5	$\{e, g\}$	$\{C_2, C_2\}$	$\emptyset$
n_6	$\{f, g, h\}$	$\{C_1, C_2, C_5\}$	$\{C_1, C_2, C_5\}$

cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a, c\}$
C_4	$\{b\}$
C_5	$\{h\}$



(a) (b)