CAD Algorithms for Physical Design - Clustering

Christos P Sotiriou

Clustering Algorithms Reviewed

- Rajaraman-Wong Algorithm
- Flowmap
- Best Choice Clustering
- Hmetis

Clustering Algorithms Overview

- RW and Flowmap
  - Delay-Optimal Clustering
- RW
  - "general-delay model"
    - Each node has a unique delay, inter-cluster edge has delay D, intra-cluster edge has zero delay
  - Cluster size is bounded
- Flowmap
  - "unit-delay model"
    - Inter-cluster delay has a unit (1) delay, nodes and intra-cluster edges do not incur any delay
  - Cluster connections are bounded ("pin-constraint")

Rajaraman-Wong (RW) Algorithm

- Cluster Delays
  - Inter-cluster edge has **constant delay D**
  - Intra-cluster edge has **delay of zero**
- Two phases: labelling and clustering
  - **Labelling** phase: compute node label in topological order
    - label denotes longest path delay from an PI to each node, including both node delay and inter-cluster delay
  - **Clustering** phase: actual grouping and duplication occurs while visiting the nodes in reverse topological order
- Maximum Delay from PIs to POs is minimised
- An n x n matrix Delta is computed containing all-pair maximum delay (longest level-based path) values
- Labels of PIs are initialized to 1, of other nodes to 0
- Non-PIs are then visited in topological order to compute their labels

Rajaraman-Wong (RW) Algorithm - Labelling

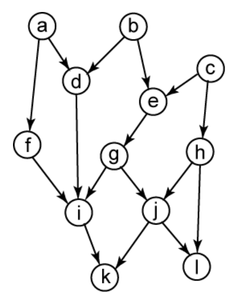
- Given a node v, to compute l(v), its label:
  - We compute the sub-graph rooted at *v*, denoted *Gv*, that includes all the predecessors of *v*.
  - We compute *lv*(*x*) for each node $x \in Gv \backslash \{v\}$, where $lv(x) = l(x) + \Delta(x, v)$
    - *l*(*x*) denotes the current label for *x*, and $\Delta(x, v)$ is Delta matrix
  - We sort all nodes in $Gv \backslash \{v\}$ in decreasing order of their *lv*-values and put them into a set *S*
  - 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 a size constraint is violated
  - We compute two values *l1* and *l2*
    - If *cluster*(*v*) contains any PI nodes, the maximum *lv* value among these PI nodes becomes *l1*
    - If *S* is not empty after filling up *cluster*(*v*), the maximum *lv* + *D* among the nodes remaining in *S* becomes *l2*, where *D* is the inter-cluster delay
  - The new label for *v* is the maximum between *l1* and *l2*

Rajaraman-Wong (RW) Algorithm - Clustering

- Clustering Phase:
  - During the clustering phase, we first put all PO nodes in a set *L*
  - We then remove a node from *L* and form its cluster. Given a node *v*, we form a cluster by grouping all nodes in *cluster*(*v*), which was computed during the labeling phase
  - We then compute *input*(*v*), the set of input nodes of *cluster*(*v*).
  - Next, 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
  - We repeat the entire process until *L* becomes empty

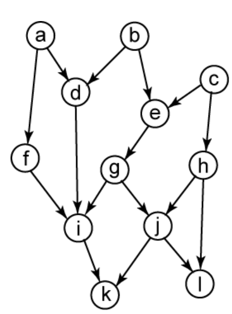
Rajaraman-Wong Algorithm Example

- 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)



Rajaraman-Wong Algorithm Example

- Max-Delay Matrix
  - All-pair delay matrix Δ(*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

Rajaraman-Wong Algorithm Example

- 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

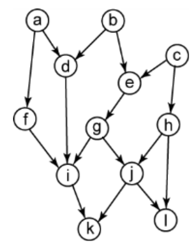
$$cluster(d) = \{a, b, d\}$$

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,

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

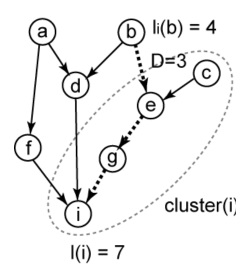


Rajaraman-Wong Algorithm Example

- 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,

$$l_i(a) = l(a) + \Delta(a, i) = 1 + 2 = 3$$
$$l_i(b) = l(b) + \Delta(b, i) = 1 + 3 = 4$$
$$l_i(c) = l(c) + \Delta(c, i) = 1 + 3 = 4$$
$$l_i(d) = l(d) + \Delta(d, i) = 2 + 1 = 3$$
$$l_i(e) = l(e) + \Delta(e, i) = 2 + 2 = 4$$
$$l_i(f) = l(f) + \Delta(f, i) = 2 + 1 = 3$$
$$l_i(g) = l(g) + \Delta(g, i) = 3 + 1 = 4$$

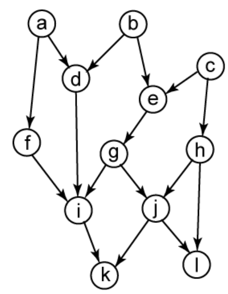


$S = \{g, e, c, b, a, d, f\}$, and we form $cluster(i) = \{i, g, e, c\}$.[1] Note that c is PI, so $l_1 = l_i(c) = 4$. Since $S = \{b, a, d, f\} \neq \emptyset$ after clustering, we have $l_2 = l_i(m(S)) + D = l_i(b) + D = 4 + 3 = 7$ (recall that $m(S)$ is the node in S with the maximum value of l_i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

Rajaraman-Wong Algorithm Example

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

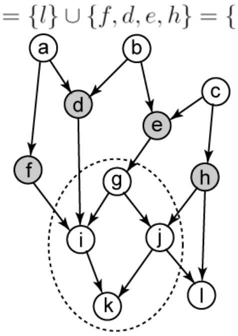
node	label	clustering
a	1	$\{a\}$
b	1	$\{b\}$
c	1	$\{c\}$
d	2	$\{a, b, d\}$
e	2	$\{b, c, e\}$
f	2	$\{a, f\}$
g	3	$\{b, c, e, g\}$
h	2	$\{c, h\}$
i	7	$\{c, e, g, i\}$
j	7	$\{b, e, g, j\}$
k	8	$\{g, i, j, k\}$
l	8	$\{e, g, j, l\}$



Rajaraman-Wong Algorithm Example

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

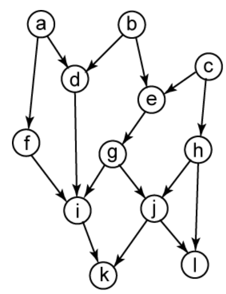
remove k from L, and add $cl(k)$ to $S = \{cl(k)\}$. According to Table 1.1, we see that $cl(k) = \{g, i, j, k\}$. Then, $I[cl(k)] = \{f, d, e, h\}$ as illustrated in Figure 1.4. Since S does not contain clusters rooted at f, d, e, and h, we have $L = \{l\} \cup \{f, d, e, h\} = \{l, f, d, e, h\}$.



Rajaraman-Wong Algorithm Example

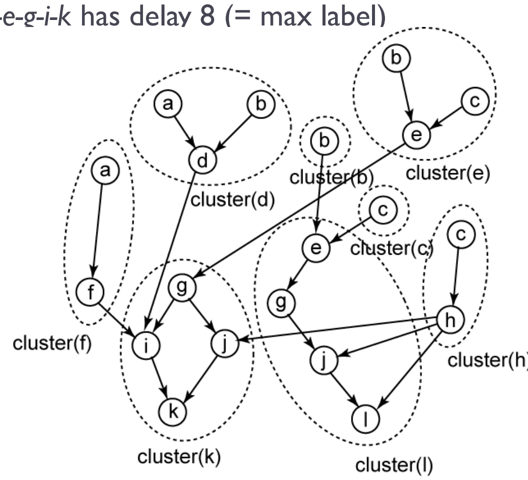
- Clustering Summary
  - Clustering phase generates 8 clusters.
    - 8 nodes are duplicated

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	$\{c\}$



Rajaraman-Wong Algorithm Example

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



Flowmap Algorithm

- Cluster Delays
  - Inter-cluster edge has **unit (1) delay**
  - Intra-cluster edge has **delay of zero**
- Cluster External Connections are Constrained
  - Applicable to FPGA Technology-Mapping
- Two phases: labelling and mapping
  - **Labelling** phase: compute node label, and clustering ~Xv
    - ~Xv denotes set of nodes to be clustered together with v
    - label denotes longest path delay from an PI to each node, where only inter-cluster edges incur unit delay
  - **Mapping** phase: actual grouping and duplication occurs while visiting the nodes in reverse topological order
- Maximum Delay from PIs to POs is minimised
- Labels of PIs are initialized to 0
- Non-PIs are then visited in topological order to compute their labels and ~X sets

Flowmap Algorithm - Labelling

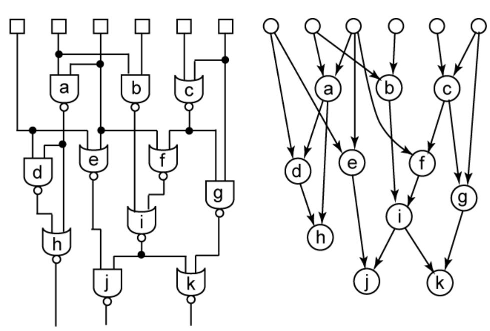
- Given a node *t*, we do the following to compute its new label, *l*(*t*):
  - We compute the sub-graph rooted at *t*, denoted *Nt*, that includes all of the predecessors of *t*. We then add a source node *s* to *Nt* and connect it to all PIs in *Nt*
  - We compute *p*, the maximum label among all fan-in nodes of *t*.
  - We obtain *N't* , where all nodes with their labels equal to *p* are collapsed into *t*
  - We obtain a flow-network *N"t*, where each node *x* in *N't* except *s* and *t* is split into two nodes (*x*, *x'*), and connected via a "bridging edge" e(*x*, *x'*). We assign the capacity of 1 to all bridging edges and infinity to all non-bridging edges of *N"t*
  - We compute a cut *C*(*X"*, ~*X"*) that separates *s* and *t* in *N"t* with the cutsize not larger than K, the pin constraint. This is performed by identifying augmenting paths from *s* to *t*. If multiple cuts are found, select the minimum-height cut, *i.e.* maximum label of *X* nodes
  - We include all nodes of ~*X"* into ~*Xt*. If a node *x* is split and e(*x*, *x'*) is cut in *C*, *x'* is removed from ~*X"*; *l*(*t*) = *p*
  - If C is not found, ~*X"* contains *t* only and *l*(*t*) = *p* + *1*

Flowmap Algorithm - Mapping

- During Mapping, all PO nodes are placed in set L
- We then remove a node from *L* and form its cluster as follows:
  - given a node *v*, we form a cluster, named *v'*, by grouping all non-PI nodes in ~*Xv*, computed during labelling phase
  - We then compute *input(v')*, the set of input nodes of *v'*, and include them in L
    - A node x is an input node of v' if e(x, y) exists in the original DAG, and y is in v'.
- Process is repeated until L is empty

Flowmap Algorithm Example

- Perform clustering on the following 2-bounded network
  - Intra-cluster and node delay = 0, inter-cluster = 1
  - Pin constraint = 3



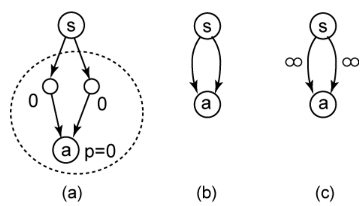
(a) (b)

Flowmap Algorithm Example

- Label Computation

First, all PIs are assigned zero for their label. We then visit the remaining nodes in topological order $T = [a, b, c, d, e, f, g, h, i, j, k]$.

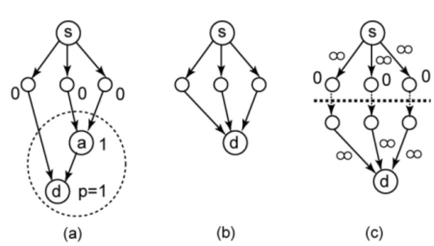
(a) node a: We first build N_a as shown in Figure 1.7(a). We see that $p = 0$. This helps us build N'_a and N''_a as shown in Figure 1.7(b) and Figure 1.7(c). Note that it is not possible to find a cut in N''_a with a cutsize smaller or equal to $K = 3$. Thus, $\overline{X}_a = \{a\}$ and $l(a) = p + 1 = 1$.



Flowmap Algorithm Example

- Label Computation

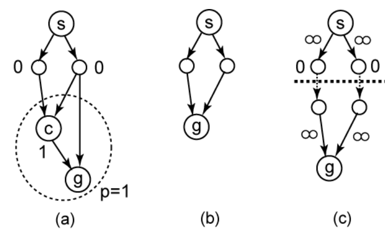
(c) node d: Figure 1.8 shows N_d, N'_d, and N''_d under $p = 1$. There is a possible cut in N''_d as shown on Figure 1.8(c), where the maximum flow value and the cutsize is 3. The height of this cut is zero because the label for all nodes in the source-side partition is zero. Node a and d are partitioned to the sink-side. Thus, $\overline{X}_d = \{a, d\}$, and $l(d) = p = 1$.



Flowmap Algorithm Example

- Label Computation

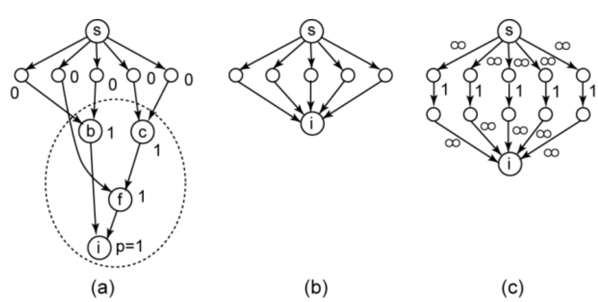
(f) node g: Figure 1.9 shows N_g, N'_g, and N''_g. There is only one cut possible in N''_g as shown on Figure 1.9(c). Thus, $\overline{X}_g = \{c, g\}$, and $l(g) = p = 1$.



Flowmap Algorithm Example

- Label Computation

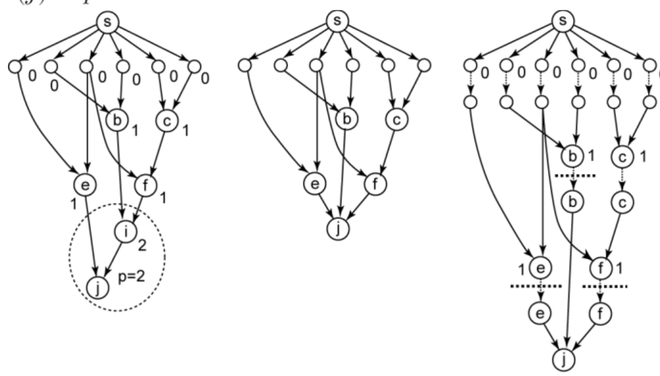
(h) node i: Figure 1.11 shows N_i, N'_i, and N''_i. We see that $p = 1$. In this case, N''_i does not contain a K-feasible cut. Thus, $\overline{X}_i = \{i\}$, and $l(i) = p + 1 = 2$.



Flowmap Algorithm Example

- Label Computation

(i) node j: Figure 1.12 shows N_j, N'_j, and N''_j. $p = 2$ in this case. There is only one K-feasible cut in N''_j, and its height is 1. Thus, $\overline{X}_j = \{i, j\}$, and $l(j) = p = 2$.

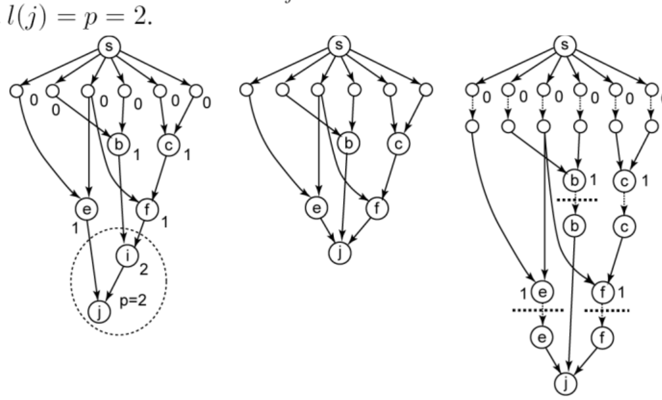


(a) (b) (c)

Flowmap Algorithm Example

- Label Computation

(i) node j: Figure 1.12 shows N_j, N'_j, and N''_j. $p = 2$ in this case. There is only one K-feasible cut in N''_j, and its height is 1. Thus, $\overline{X}_j = \{i, j\}$, and $l(j) = p = 2$.

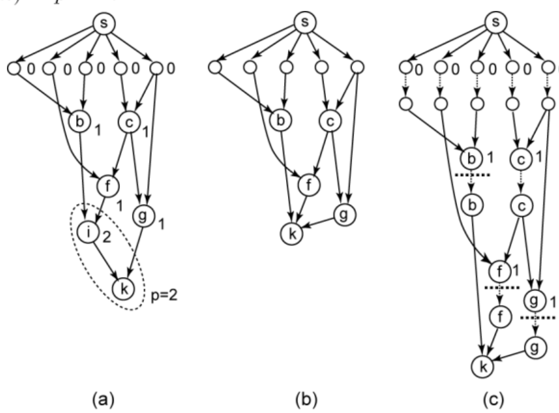


(a) (b) (c)

Flowmap Algorithm Example

- Label Computation

(j) node k: Figure 1.13 shows N_k, N'_k, and N''_k. $p = 2$ in this case. There is only one K-feasible cut in N''_k, and its height is 1. Thus, $\overline{X}_k = \{i, k\}$, and $l(k) = p = 2$.



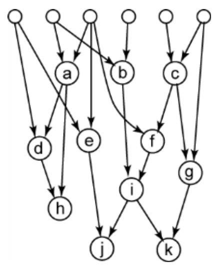
Flowmap Algorithm Example

- Summary
  - Max label = max delay in the clustered network = 2

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

Flowmap Algorithm Example

- Clustering Phase
  - Traverse the nodes from PO to PI
    - We begin with L = POs = {*h,j,k*}
    - Clustering is based on:



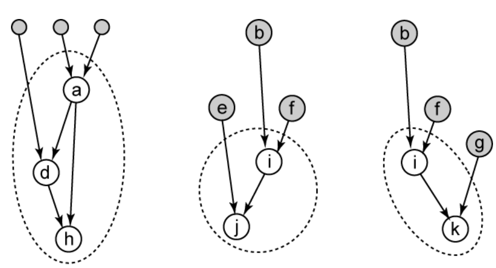
+

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

Flowmap Algorithm Example

- Clustering Phase

(a) remove h from L. Then, h', the K-LUT implementation of h, contains $\{a, d, h\}$ according to Table 1.3. We note that $input(h')$ contains three PI nodes as shown in Figure 1.14(a). Since we do not add PI nodes into L, we have $L = \{j, k\}$.



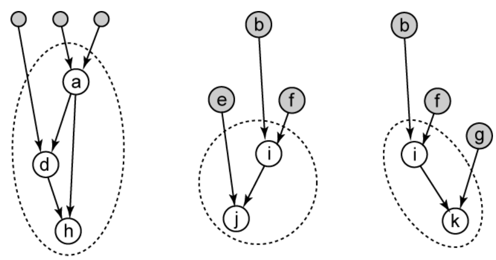
(a) (b) (c)

Flowmap Algorithm Example

- Clustering Phase

(b) remove j from L: $j' = \{i, j\}$ according to Table 1.3. We see that $input(j') = \{e, b, f\}$ as shown in Figure 1.14(b). Thus, $L = \{k\} \cup \{e, b, f\} = \{k, e, b, f\}$.

(c) remove k from L: $k' = \{i, k\}$, and $input(k') = \{b, f, g\}$ as shown in Figure 1.14(c). Thus, $L = \{e, b, f\} \cup \{b, f, g\} = \{e, b, f, g\}$.



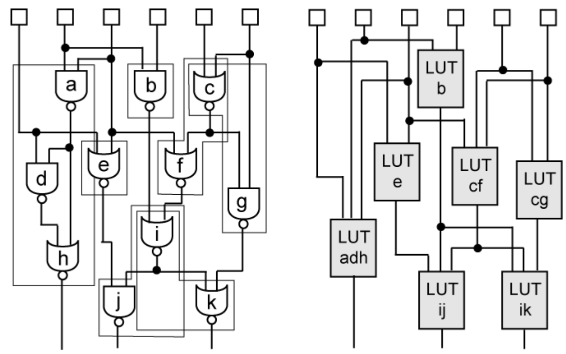
Flowmap Algorithm Example

- Summary
  - 6 clusters (= LUT-3) are generated
  - Node *c* and *i* are duplicated

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

Flowmap Algorithm Example

- Clustered Network
  - Max delay = 2



Hmetis Algorithm

- Combination of Clustering and Partitioning
- Clustering is multi-level, *i.e.* takes place in multiple passes
  - First Iteration: level 1 clusters
  - Second Iteration: level 2 clusters
  - … until K-levels of clustering hierarchy exist
- Partitioning Phase
  - Bipartitioning using existing algorithm e.g. FM
  - K-level clusters are decomposed into K-1 level clusters
  - Decomposition and refinement process
- Hmetis Clustering
  - (1) Edge Coarsening (EC), (2) Hyperedge Coarsening (HEC), (3) Modified Hyperedge Coarsening (MHEC)

Hmetis Algorithm - EC

- **Edge Coarsening (EC)**
  - Hypergraph nodes are visited in a random order
  - For an unmarked node, v, collect neighbours of v
    - Set of unmarked nodes contained in v's hyperedges
  - For each neighbour, n, compute weight of edge (v, n), by assigning a weight 1/(|h|-1) to relevant hyperedge h
  - Select neighbour with maximum edge weight m
    - Merge v and m together
    - Mark them so that these nodes are not re-clustered
  - Repeat until all nodes are marked

Hmetis Algorithm - HEC

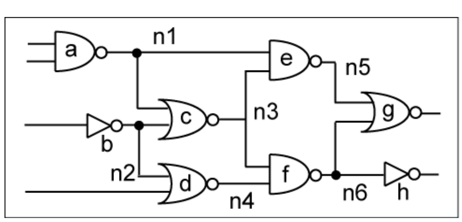
- **Hyperedge Coarsening:**
  - Unmark all nodes
  - Sort hyperedges in increasing size order
    - If weighted, sort in decreasing order of weight, and break ties for smaller size
  - Visit hyperedges in sorted order
    - If hyperedge does NOT contain an already marked node
      - Group all nodes in the hyperedge to form a cluster
    - Else skip to next
  - After visiting all hyperedges, each node that is NOT part of any cluster becomes a singleton cluster

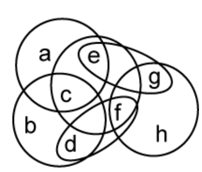
Hmetis Algorithm - MHEC

- **Modified Hyperedge Coarsening:**
  - Apply HEC first
  - After clustered hyperedges have been selected, visit them again in sorted order
  - For each hyperedge that is NOT yet clustered – because it contains marked nodes – all unmarked nodes are clustered together

Hmetis Algorithm Example

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



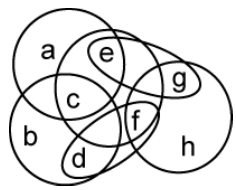


Hmetis Algorithm Example – Edge Coarsening

(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.



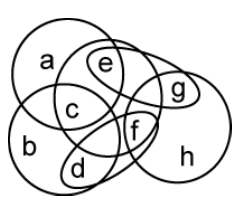
cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$

Hmetis Algorithm Example – Edge Coarsening - 2

(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.



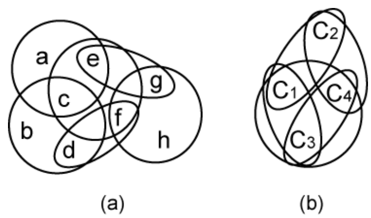
cluster	nodes
C_1	$\{a, c\}$
C_2	$\{b, d\}$
C_3	$\{e, g\}$
C_4	$\{f, h\}$

Hmetis Algorithm Example – Obtaining Clustered-level Netlist

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

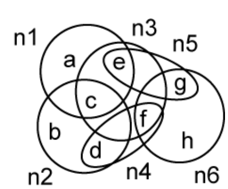
net	gate-level	cluster-level	final
n_1	$\{a, c, e\}$	$\{C_1, C_1, C_3\}$	$\{C_1, C_3\}$
n_2	$\{b, c, d\}$	$\{C_2, C_1, C_2\}$	$\{C_1, C_2\}$
n_3	$\{c, e, f\}$	$\{C_1, C_3, C_4\}$	$\{C_1, C_3, C_4\}$
n_4	$\{d, f\}$	$\{C_2, C_4\}$	$\{C_2, C_4\}$
n_5	$\{e, g\}$	$\{C_3, C_3\}$	$\emptyset$
n_6	$\{f, g, h\}$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

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



Hmetis Algorithm Example – 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

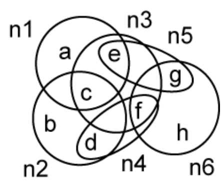


Hmetis Algorithm Example – Hyperedge Coarsening

(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.



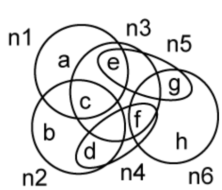
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$

Hmetis Algorithm Example – Hyperedge Coarsening - 2

(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.



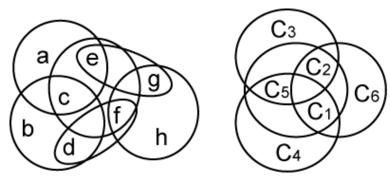
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$

Hmetis Algorithm Example – 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	$\{f, g, h\}$	$\{C_1, C_2, C_6\}$	$\{C_1, C_2, C_6\}$

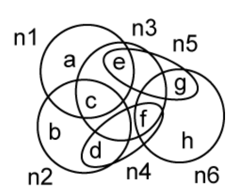
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a\}$
C_4	$\{b\}$
C_5	$\{c\}$
C_6	$\{h\}$



(a) (b)

Hmetis Algorithm Example – 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



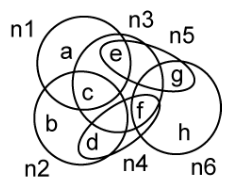
Hmetis Algorithm Example – Modified Hyperedge Coarsening

(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.



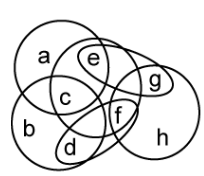
cluster	nodes
C_1	$\{d, f\}$
C_2	$\{e, g\}$
C_3	$\{a, c\}$
C_4	$\{b\}$
C_5	$\{h\}$

Hmetis Algorithm Example – Obtaining Clustered-level Netlist

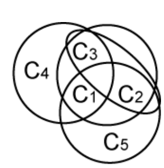
- # of nodes/hyperedges reduced: 5 nodes, 4 hyperedges

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)

Best Choice Clustering

- **Score Function**
  - Hyperedge weight we of e is defined as 1/|e|
    - Weight is inversely proportional to objects incident to hyperedge
  - Given two objects u and v, the clustering score d(u, v) is defined as:

$$d(u,v) = \sum_{e} \frac{w_e}{a(u) + a(v)}$$

    - Where a(u) and a(v) are the corresponding areas of objects u
- **Closest object**
  - For object u, let Nu be the neighboring objects of u
  - Closest object of u, c(u) is the neighbor with largest clustering score to u, *i.e.*:

$$c(u) = v : d(u,v) = \max_{N_u} d(u,z), \forall z \in N_u$$

Best Choice Clustering Algorithm

- Termination Conditions
  - Goal cluster bottom-up until a desired # is reached:
    - Clustering Ratio a

Input: Flat Netlist Output: Clustered Netlist
1. Until *target object number* is reached: 2. Find *closest pair* of objects 3. Cluster them 4. Update netlist

Best Choice Clustering Algorithm

Input: Flat Netlist
Output: Clustered Netlist

Phase I. Priority-queue PQ Initialization:
1. For each object u:
2. Find *closest object* v, and its associated clustering score d
3. Insert tuple (u, v, d) into PQ with d as key

Phase II. Clustering:
1. While *target object number* is not reached and top tuple's score $d > 0$:
   2. Pick top tuple (u, v, d) of PQ
   3. Cluster u and v into new object u'
   4. Update netlist
   5. Find *closest object* v' to u' with its clustering score d'
   6. Insert tuple (u', v', d') into PQ with d' as key
   7. Update clustering scores of all neighbors of u'

- Step 7 is most time-consuming step
  - Clustering scores of the neighbors of the new object *u'*, (equivalently all neighbors of *u* and *v*) are re-calculated

Best Choice Clustering – Lazy Speedup

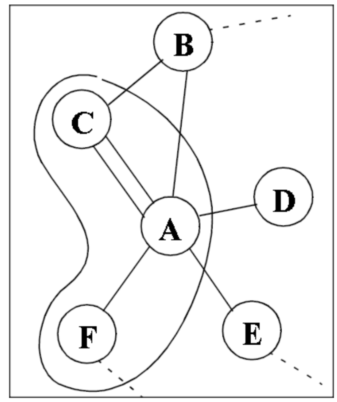
Input: Flat Netlist
Output: Clustered Netlist

Phase II. Clustering:
1. While *target object number* is not reached and top tuple's score $d > 0$:
   2. Pick top tuple (u, v, d) of PQ
   3. If u is marked as invalid, re-calculate *closest object* v' and score d' and insert tuple (u, v', d') to PQ
   4. else
      5. Cluster u and v into new object u'
      6. Update netlist
      7. Find *closest object* v' to u' with its clustering score d'
      8. Insert tuple (u', v', d') into PQ with d' as key
      9. Mark all neighbors of u' as invalid

- *Lazy-Update* technique
  - delays updates of clustering scores as late as possible, thus reducing the actual number of score update operations on the priority queue

Best Choice Clustering Example

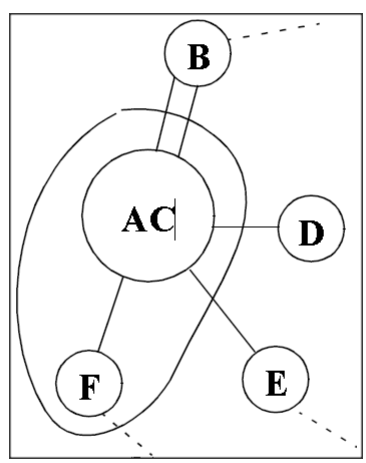
- Assume the input netlist with 5 objects
  - {A, B, C, D, E, F} and 8 hyperedges {A, B}, {A, D}, {A, E}, {A, F}, {A, C}, another {A, C}, {B, C} and {A, C, F}



- Clustering Score of A to its neighbors
  - *d(C, B) = 1/2, d(A,B) =1/2, d(A, C) = 4/3, d(A, D) = 1/2, d(A,E) = 1/2,* and *d(A, F) =5/6.*
  - *d(A, C)* has the highest score, and *C* is declared as the closest object to *A*

Best Choice Clustering Example

- If we assume that *d(A, C)* is the highest score in the priority queue,
  - *A* will be clustered with *C* and the circuit netlist will be updated as shown



- With new object *AC* introduced, corresponding cluster scores will be
  - *d(AC, F)= 1, d(AC, E) = 1/2, d(AC, D) = 1/2,* and *d(AC, B) = 1.*

Cluster Size Bounds

- Without an area control, gigantic clustered objects might be formed by absorbing small objects and/or clusters around it
- **Indirect Area Control**

$$d(u, v) = \sum_e w_e / [a(u) + a(v)]^k$$

  - where $k = \lceil (a(u) + a(v))/\mu \rceil$
  - μ = average cell area x clustering ratio
    - and represents the expected average area of clustered objects
  - Another possibility is to use cluster # of pins
- **Direct Area Control**
  - Hard Bound: if resultant area > (k x μ), reject clustering
  - Soft Bound: if resultant area > (k x μ), accept with probability
  - $2^{(\mu/(a(u)+a(v)))^k} - 1$ where $k \geq 1$

Clustering Scores Comparison

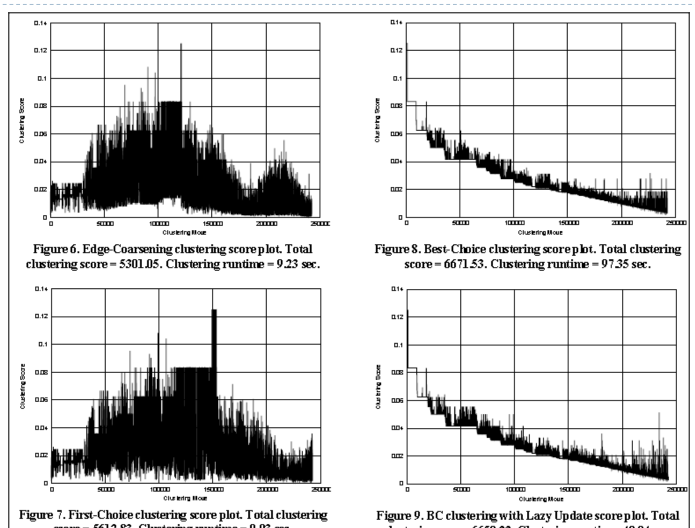


Figure 6. Edge-Coarsening clustering score plot. Total clustering score = 5301.05. Clustering runtime = 9.23 sec.

Figure 8. Best-Choice clustering score plot. Total clustering score = 6671.53. Clustering runtime = 97.35 sec.

Figure 7. First-Choice clustering score plot. Total clustering score = 5612.83. Clustering runtime = 9.03 sec.

Figure 9. BC clustering with Lazy Update score plot. Total clustering score=6658.23. Clustering runtime=49.84 sec.