Slicing Floorplan Design with Boundary-Constrained Modules

En-Cheng Liu[1], Ming-Shiun Lin[1]

Jianbang Lai[2], Ting-Chi Wang[3]

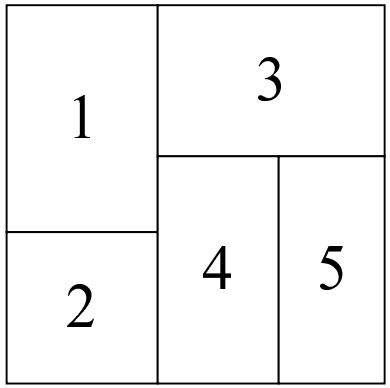
[1]Dept. of Information & Computer Engineering, Chung Yuan Christian Univ., Chungli, Taiwan

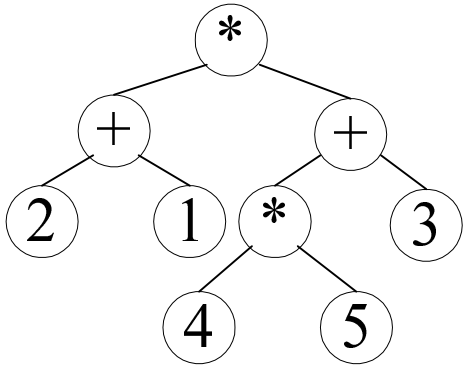
[2]Tatung Corp., Taipei, Taiwan

[3]Dept. of Electrical Engineering, Texas A&M University, College Station, TX

Floorplan Design

- Floorplan Design: determine shapes and locations of modules on a rectangular chip to optimize total area and/or interconnect cost (or other measure).
- Slicing floorplan, slicing tree, and normalized Polish expression:

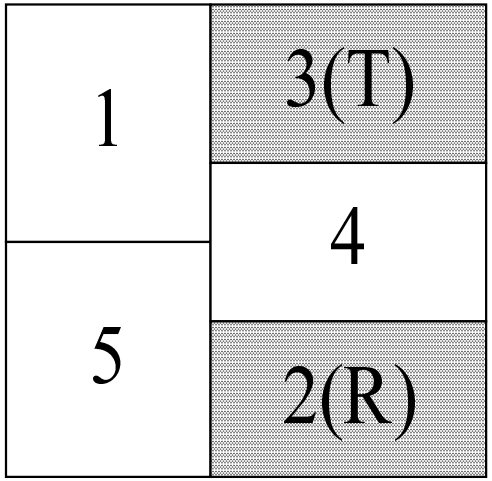


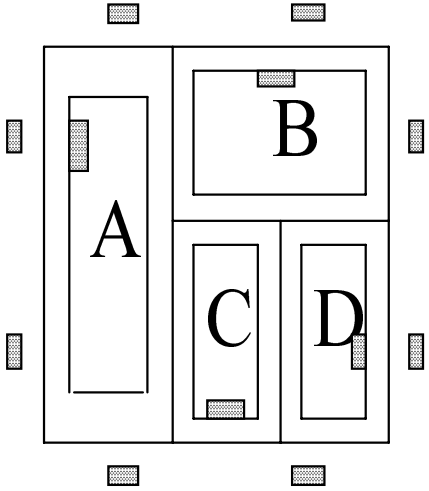


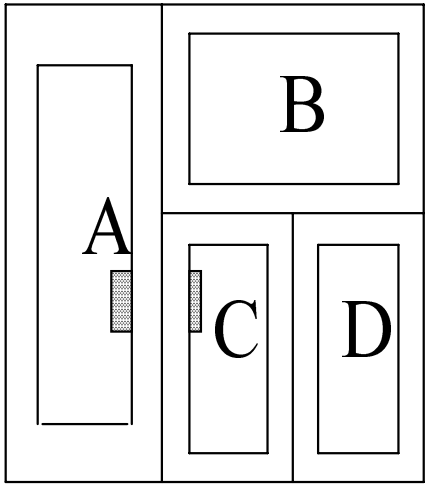
21+45*3+*

Boundary Constraints

- Some modules must be placed along given boundaries.
- Reasons:
  - Easier I/O connections or
  - Easier interconnections between modules of different units.







Problem Formulation

- Input: five *disjoint* rectangular module sets M_F, M_L, M_R, M_T and M_B.
  - Each module in M_F is a *free* module that can be placed anywhere.
  - Each module in M_L (M_R, M_T, M_B, respectively) must be placed along the *left* (*right*, *top*, *bottom*, respectively) boundary.
  - Each module is given the area and aspect ratio range.
- Output: a *feasible* slicing floorplan such that the cost function $A+\lambda W$ is minimized.
  - A: the total area of the floorplan.
  - W: the estimated interconnect wirelength, defined to be $\sum_{i \neq j} c_{ij} \times d_{ij}$.
    * c_{ij}: number of common nets between modules i and j.
    * d_{ij}: Manhattan distance between i and j.
  - λ: user-specified constant.

Previous Work (1)

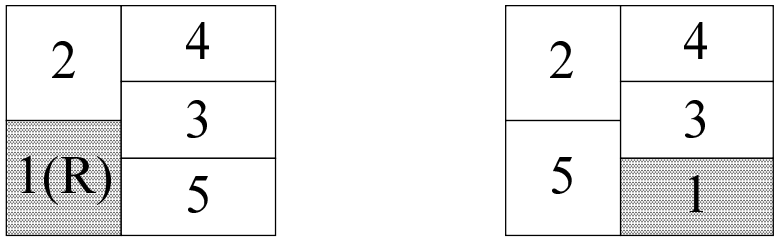
- Wong-Liu algorithm (DAC'86):
  - Slicing floorplan design without any placement constraints.
  - Represent a slicing structure by a normalized Polish expression.
  - A simulated annealing based approach with three types of perturb operations:

    * M1: swap two adjacent modules; 12*34*+ => 13*24*+
    * M2: complement a chain of operators; 12*34*+ => 12*34+*
    * M3: swap two adjacent module and operator; 12*34*+ => 12*3*4+

  - Use an efficient shape curve computation technique to find a floorplan of "best" area for a Polish expression, and then compute the weighted cost of the area and wirelength for evaluating the expression.

Previous Work (2)

- Young-Wong algorithm (ASP-DAC'99):
  - Slicing floorplan design with boundary constraints.
  - An extension of Wong-Liu algorithm.
    * Scan a Polish expression once to find the boundary information of each module in the corresponding floorplan.
    * Fix an infeasible floorplan by swapping modules

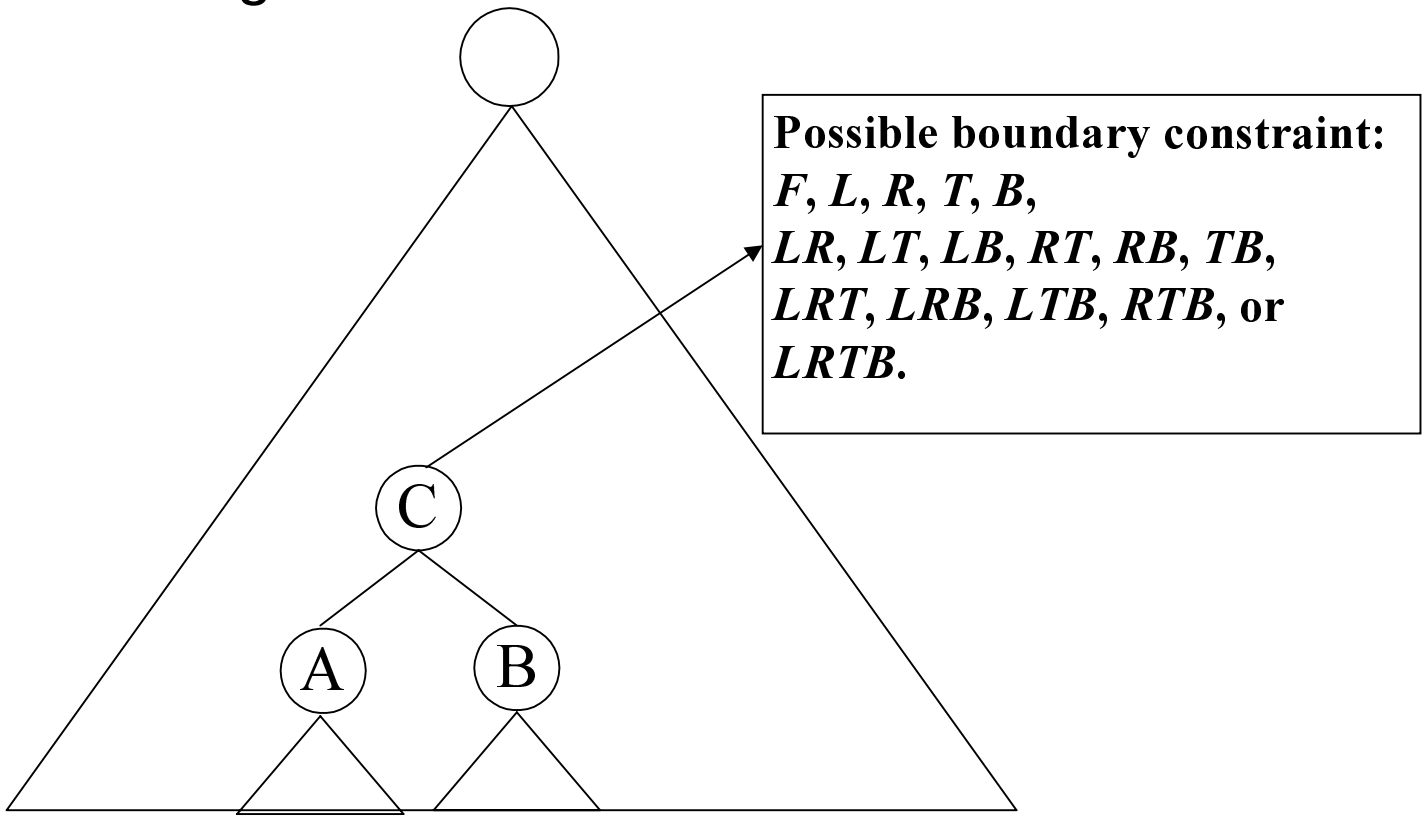


$\underline{1}2+\underline{5}3+4+*$ $\quad\quad$ $52+13+4+*$

  - The fixing method may fail.

Combining

- Node *c* could correspond to a feasible or an infeasible combining.



All Feasible Combinings

Resulting constraint	* (vertical cut)		+ (horizontal cut)	
	A	B	A	B
F	1. F	1. F	1. F	1. F
L	1. L	1. F	1. L	1. F
			2. F	2. L
R	1. F	1. R	1. R	1. F
			2. F	2. R
T	1. T	1. F	1. F	1. T
	2. F	2. T		
B	1. B	1. F	1. B	1. F
	2. F	2. B		

(a)

Resulting constraint	* (vertical cut)		+(horizontal cut)	
	A	B	A	B
LRTB	1. L	1. RTB	1. B	1. LRT
	2. LT	2. RB	2. LB	2. RT
	3. LT	3. RTB	3. LB	3. LRT
	4. LB	4. RT	4. RB	4. LT
	5. LB	5. RTB	5. RB	5. LRT
	6. LTB	6. R	6. LRB	6. T
	7. LTB	7. RT	7. LRB	7. LT
	8. LTB	8. RB	8. LRB	8. RT
	9. LTB	9. RTB	9. LRB	9. LRT

(b)

Resulting constraint	* (vertical cut)		+ (horizontal cut)	
	A	B	A	B
LR	1. L	1. R	1. L	1. R
			2. R	2. L
			3. L	3. LR
			4. R	4. LR
			5. LR	5. L
			6. LR	6. R
			7. LR	7. F
			8. F	8. LR
LT	1. L	1. T	1. L	1. T
	2. LT	2. T	2. L	2. LT
	3. LT	3. F	3. F	3. LT
LB	1. L	1. B	1. B	1. L
	2. LB	2. B	2. LB	2. L
	3. LB	3. F	3. LB	3. F
RT	1. T	1. R	1. R	1. T
	2. T	2. RT	2. R	2. RT
	3. F	3. RT	3. F	3. RT
RB	1. B	1. R	1. B	1. R
	2. B	2. RB	2. RB	2. R
	3. F	3. RB	3. RB	3. F
TB	1. T	1. B	1. B	1. T
	2. B	2. T		
	3. T	3. TB		
	4. B	4. TB		
	5. TB	5. T		
	6. TB	6. B		
	7. TB	7. F		
	8. F	8. TB		

(c)

Resulting constraint	* (vertical cut)		+ (horizontal cut)		Middle constraint
	A	B	A	B	
LRT	1. L	1. RT	1. L	1. RT	T
	2. LT	2. R	2. R	2. LT	
	3. LT	3. RT	3. L	3. LRT	
			4. R	4. LRT	
			5. LR	5. T	
			6. LR	6. RT	
			7. LR	7. LT	
			8. LR	8. LRT	
			9. F	9. LRT	
LRB	1. L	1. RB	1. B	1. LR	B
	2. LB	2. R	2. LB	2. R	
	3. LB	3. RB	3. LB	3. LR	
			4. RB	4. L	
			5. RB	5. LR	
			6. LRB	6. L	
			7. LRB	7. R	
			8. LRB	8. LR	
			9. LRB	9. F	
LTB	1. L	1. TB	1. LB	1. T	L
	2. LT	2. B	2. B	2. LT	
	3. LT	3. TB	3. LB	3. LT	
	4. LB	4. T			
	5. LB	5. TB			
	6. LTB	6. T			
	7. LTB	7. B			
	8. LTB	8. TB			
	9. LTB	9. F			
RTB	1. TB	1. R	1. RB	1. T	R
	2. B	2. RT	2. B	2. RT	
	3. TB	3. RT	3. RB	3. RT	
	4. T	4. RB			
	5. TB	5. RB			
	6. T	6. RTB			
	7. B	7. RTB			
	8. TB	8. RTB			
	9. F	9. RTB			

(d)

Transformation Method (1)

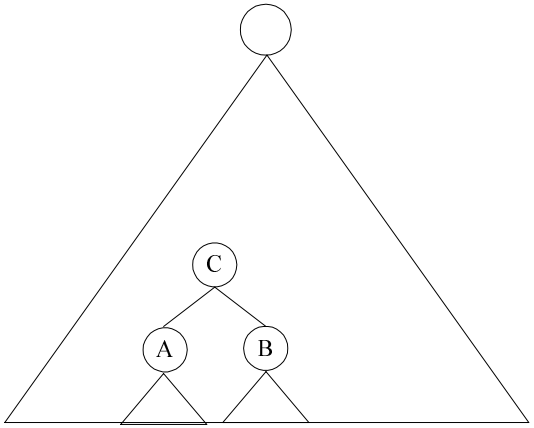
- Objective: transform a normalized Polish expression into a slicing floorplan that satisfies all given boundary constraints.
- Main ideas:
  - First construct the slicing tree from the given Polish expression.
  - Then examine each internal node of the tree in a bottom-up fashion.

    * Determine if the internal node is feasible or not.

    * Whenever necessary, modifying the tree to make the internal node satisfy its associated boundary constraint.

Transformation Method (2)

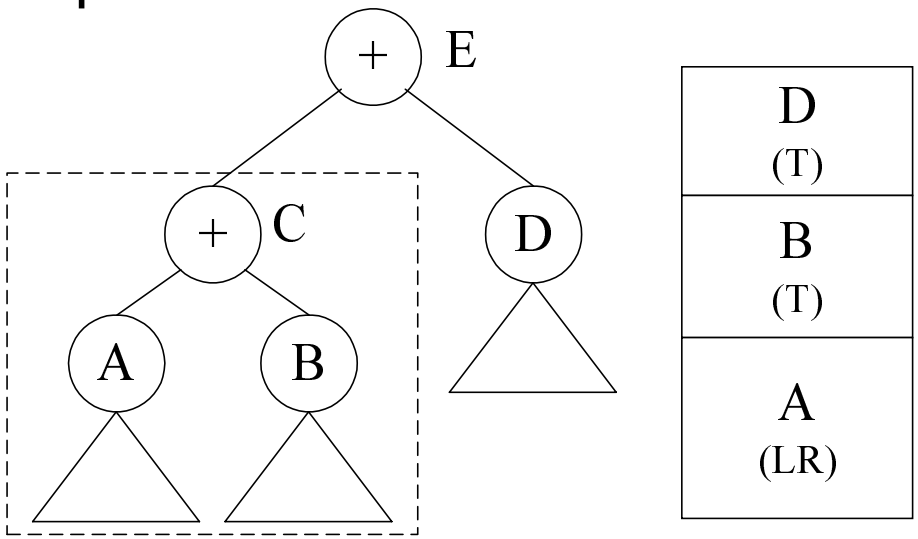
- Assume *C* is the internal node currently being considered, and *A* and *B* are its left and right child nodes.



- When *C* is a feasible combining
  - If *C* is the root of the tree, the transformation is done and the tree is returned as the output.
  - Otherwise, Cases 1-3 are considered.
- When *C* is an infeasible combining
  - Cases 4-6 are considered.

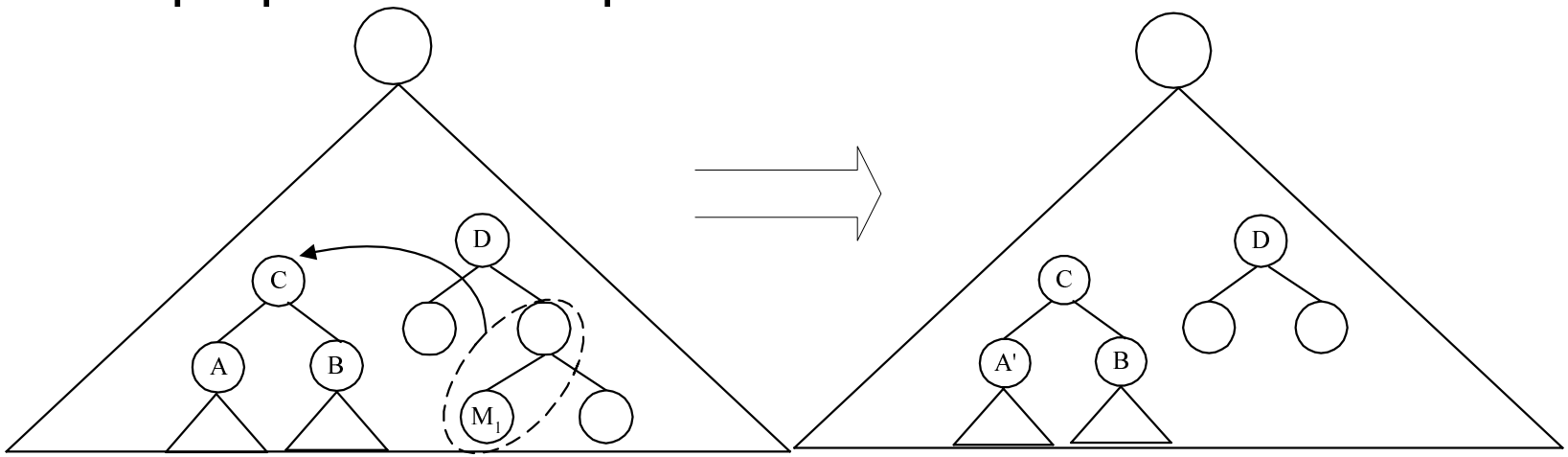
Transformation Method (3)

- Feasible combining:
  - Case 1: *C* has the *LRT*, *LRB*, *LTB*, or *RTB* constraint.
    - Example:



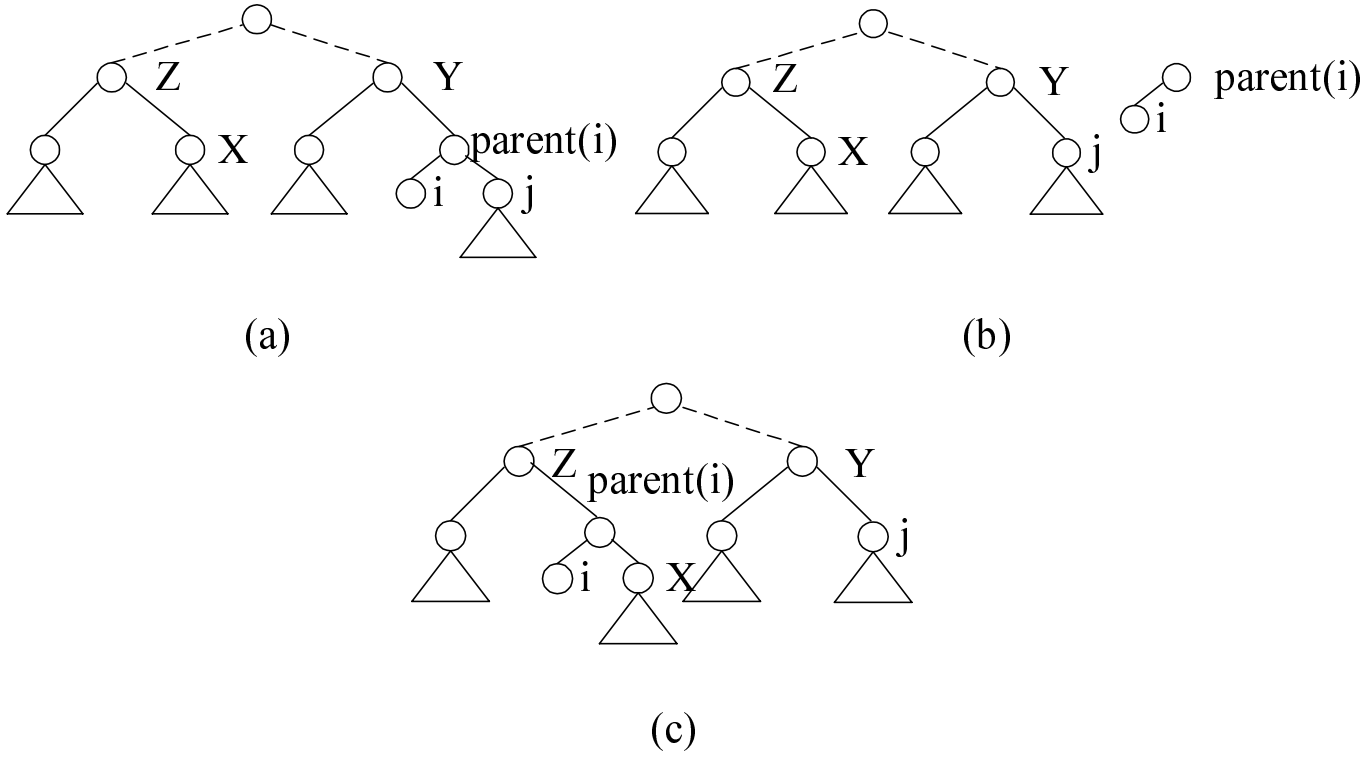
Transformation Method (4)

- Method for Case 1 (linear time):
  - Move each module with “middle” boundary constraint to the subtree rooted at C.
  - Each move can be implemented as a “delete” operation, an “insert” operation, and possibly a proper “basic” operation.



Transformation Method (5)

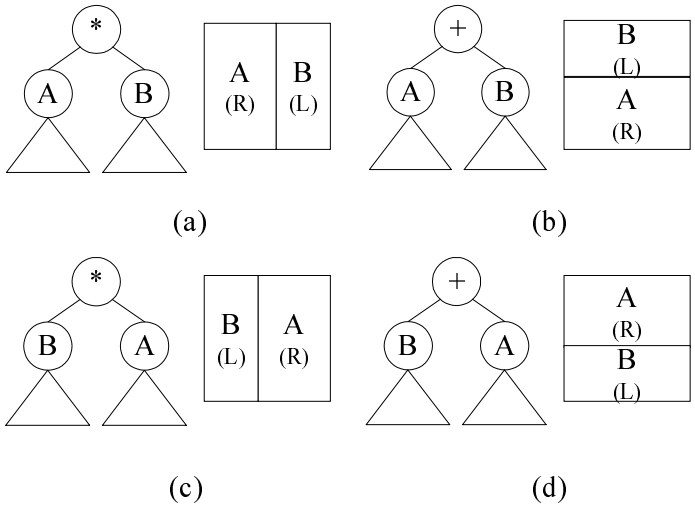
- Illustration of insert and delete operations:
  (a): a given slicing tree; (b): delete(i);
  (c) insert(parent(i),X).



- Each operation can be implemented in constant time.

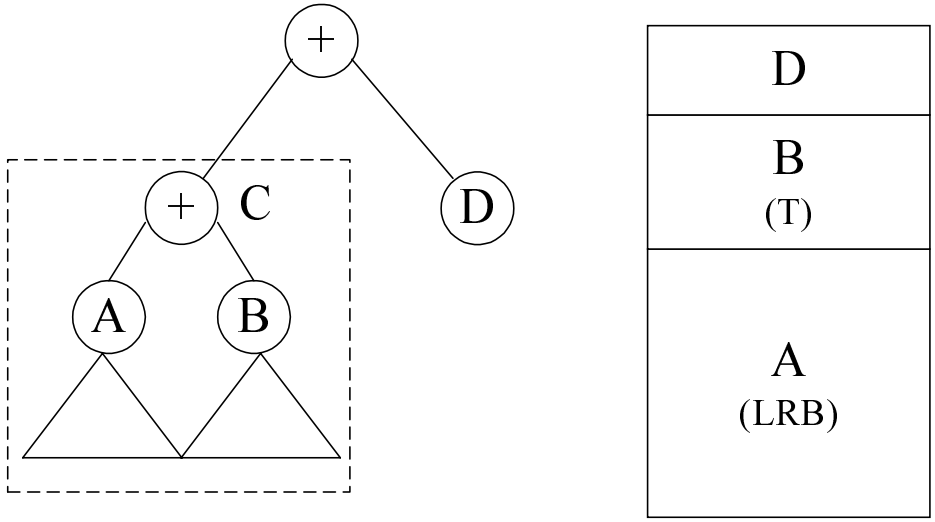
Transformation Method (6)

- Three basic operations O_1, O_2 and O_3 on a node:
  - An O_1 operation changes the cut direction of a node (e.g., (a)→(b)).
  - An O_2 operation swaps the left and the right subtrees of a node (e.g., (a)→(c)).
  - An O_3 operation performs an O_1 operation followed by an O_2 operation (e.g., (a)→(d)).
  - O_1, O_2 and O_3 each can be implemented in constant time.



Transformation Method (7)

- Feasible combining:
  - Case 2: *C* has the *LRTB* constraint.
    - Example:



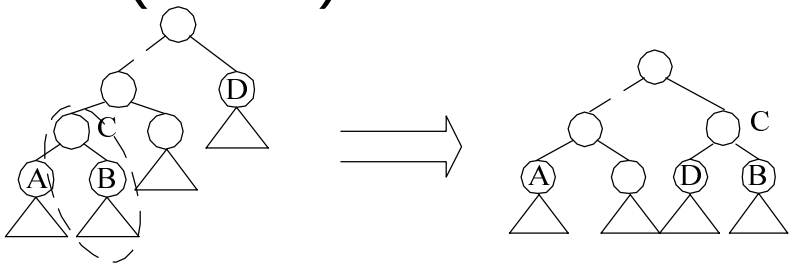
Transformation Method (8)

- Method for Case 2 (linear time):
  - Make one child have three types of constraints, and the other child the remaining type of constraint. (This is done by performing a delete, an insert and possibly a proper basic operations on each boundary-constrained module.)
  - Consider two cases for C:
    - C is in the left subtree of the root:

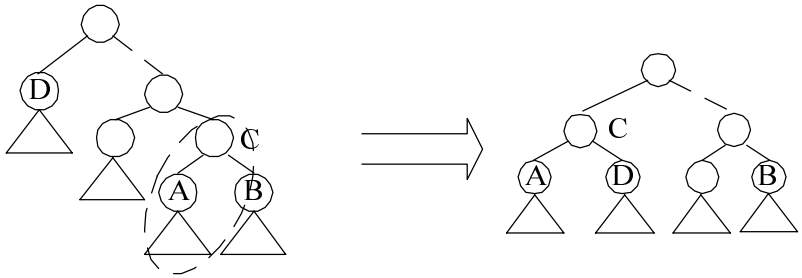
      perform "*subtree_delete*(*B*)" and then "*insert*(*parent*(*B*),*right_child*(*root*))" operations.
    - C is in the right subtree of the root: perform"*subtree_delete*(*A*)" and then "*insert*(*parent*(*A*),*left_child*(*root*))" operations.

Transformation Method (9)

- Method for Case 2 (cont'd):



(a)



(b)

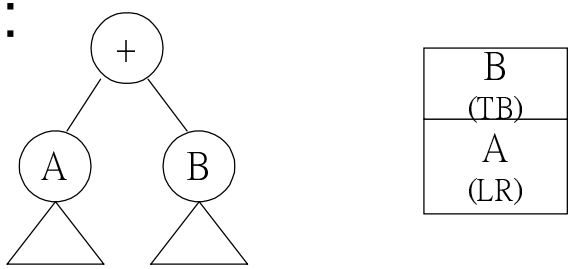
  - Apply a proper basic operation to make C feasible if necessary.
  - Finally if the root is infeasible, apply a proper basic operation to make it feasible. Now the whole transformation is done and the resulting tree is the output.

Transformation Method (10)

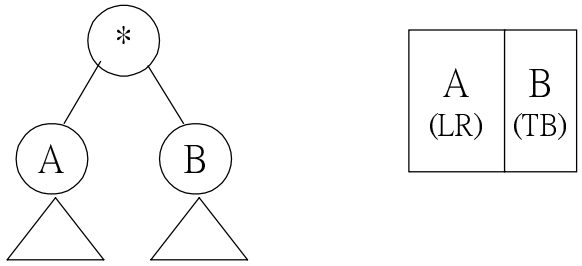
- Feasible combining:
  - Case 3: *C* is obtained from one of the remaining feasible combinings.
    - In this case, we do nothing.

Transformation Method (11)

- Infeasible combining:
  - Case 4: The two child nodes of *C* have the *LR* and *TB* constraints, respectively.
    - Examples:



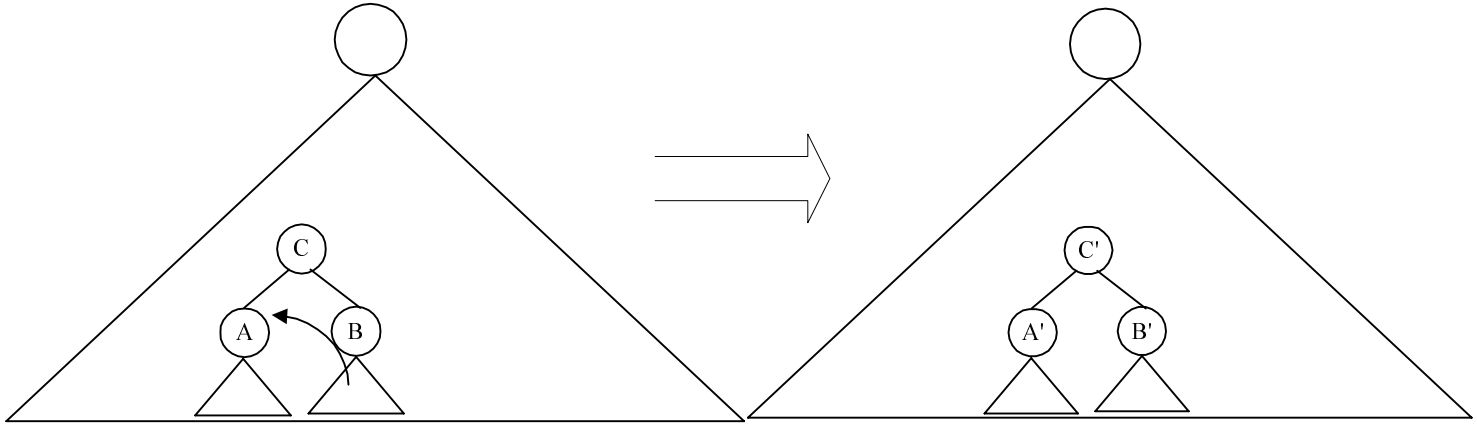
(a)



(b)

Transformation Method (12)

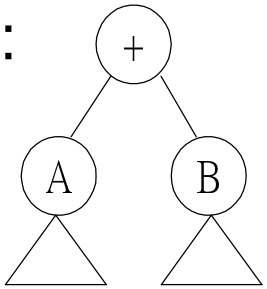
- Method for Case 4 (linear time):
  - Move a boundary-constrained module in the subtree rooted at B to the subtree rooted at A.

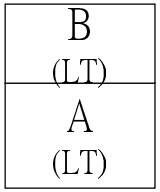


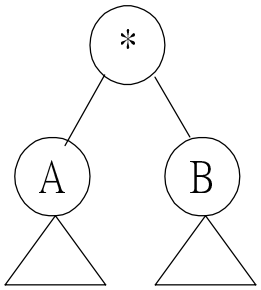
  - Apply Case 1 method to A.
  - Apply a proper basic operation to C to make it feasible if necessary.
  - Apply Case 2 method to C if C is not the root.

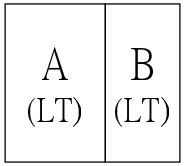
Transformation Method (13)

- Infeasible combining:
  - Case 5: Both *A* and *B* have the same *LT* (*RT*, *LB*, or *RB*) constraint .
    - Examples:



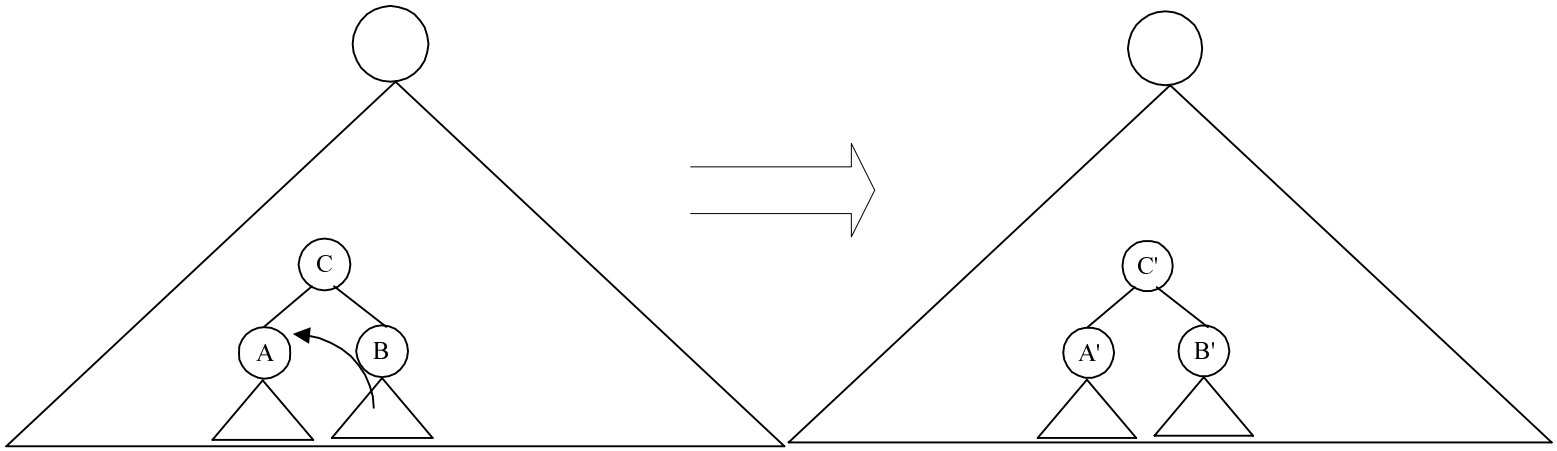






Transformation Method (14)

- Method for Case 5 (linear time):
  - Choose a “target” boundary constraint.
  - Move each module with the target boundary constraint from the subtree rooted at B to the subtree rooted at A.



  - Apply a proper basic operation to C if necessary.

Transformation Method (15)

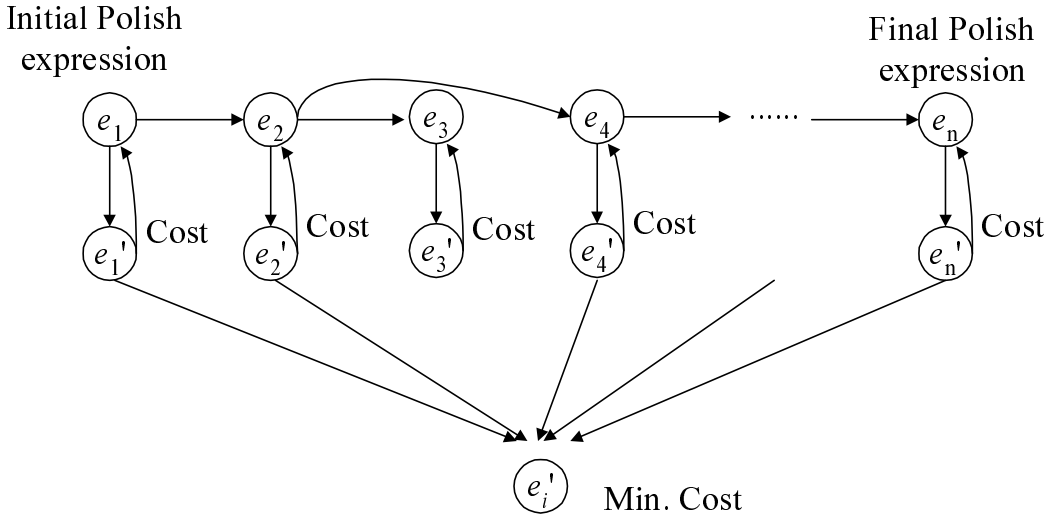
- Infeasible combining:
  - Case 6: *C* is obtained from one of the remaining infeasible combinings.
  - Method for Case 6 (linear time):
    - Apply a proper basic operation to make *C* feasible.
    - If C is not the root but meets the condition given in Case 1 (Case 2), apply Case 1 (Case 2) method to C.

Transformation Method (16)

- Overall time complexity: quadratic time.
  - Checking and fixing each internal node takes linear time.
  - The number of internal nodes is linear.
- Correctness:
  - Each internal node is ensured to be feasible after transformation.

Our Floorplanning Algorithm

- Extend the Wong-Liu algorithm by incorporating the transformation method into it.
- Ensure to always generate solutions satisfying all given boundary constraints.
- Main ideas:
  - Transform each e_i into e_i', and then evaluate e_i'.
    - e_i' is the same as e_i if e_i is already feasible.
  - If e_i' gets accepted, use e_i (instead of e_i') to generate next solution.
    - Each feasible slicing floorplan is reachable in the solution space.



Experimental Results (1)

- Three sets of 16 and 20 boundary-constrained modules were randomly chosen from ami33 and ami49, respectively.
- Same parameter settings as Young-Wong algorithm.
- Machine: Pentium-III 600 processor with 128MB RAM.
- Each algorithm was run five times on each test circuit.
- Results of optimizing area alone.

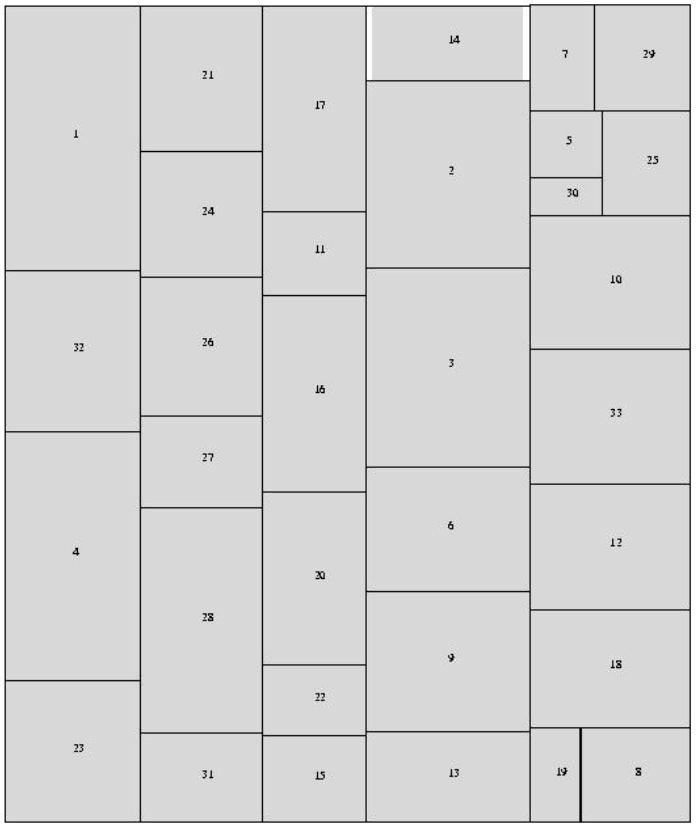
	Young-Wong Algorithm							Our Algorithm						Improv. over Young-Wong Alg.			
	Area (mm^2)		Wirelength (mm)		# Failure	Time (sec)		Area (mm^2)		Wirelength (mm)		Time (sec)		Area		Wirelength	
$\lambda = 0$	Min	Average	Min	Average		Min	Average	Min	Average	Min	Average	Min	Average	Min	Average	Min	Average
ami33-1	1.17	1.22	89.09	95.40	0	3.09	5.706	1.16	1.17	84.19	90.58	14.25	15.51	0.85%	4.10%	5.50%	5.05%
ami33-2	1.17	1.28	94.82	111.89	1	7.02	7.523	1.16	1.16	95.08	104.17	14.35	16.07	0.85%	9.38%	-0.27%	6.90%
ami33-3	1.17	1.19	103.25	113.70	0	5.13	7.146	1.16	1.17	102.19	106.03	12.83	15.36	0.85%	1.68%	1.03%	6.75%
ami49-1	36.47	37.52	1818.60	1906.39	2	10.28	32.92	36.41	37.04	1763.30	1826.19	80.04	84.05	0.16%	1.28%	3.04%	4.21%
ami49-2	37.11	38.17	1864.80	1874.27	2	13.82	30.2	36.25	37.25	1497.54	1581.15	52.01	63.25	2.32%	2.41%	19.69%	15.64%
ami49-3	36.22	38.09	1702.43	1802.80	1	14.03	27.74	36.01	36.96	1357.10	1522.17	50.98	56.65	0.58%	2.97%	20.28%	15.57%

Experimental Results (2)

- Results of optimizing both area and interconnect wirelength.

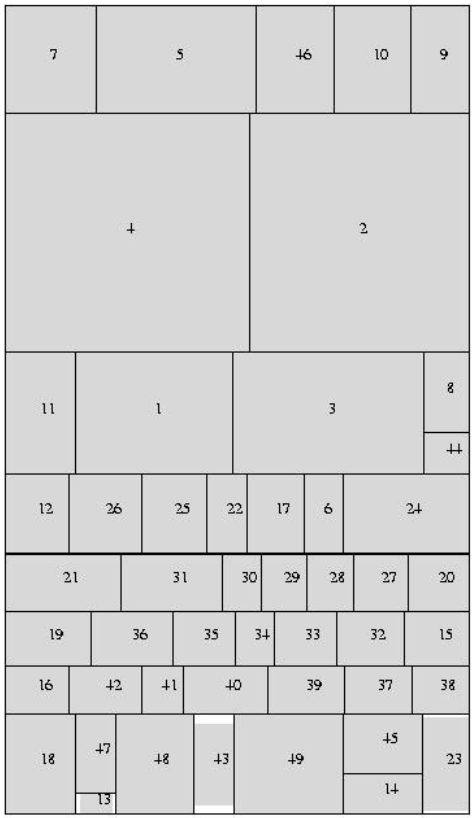
λ = 0.0158	Young-Wong Algorithm							Our Algorithm						Improv. over Young-Wong Alg.			
	Area (mm^2)		Wirelength (mm)		# Failure	Time (sec)		Area (mm^2)		Wirelength (mm)		Time (sec)		Area		Wirelength	
	Min	Average	Min	Average		Min	Average	Min	Average	Min	Average	Min	Average	Min	Average	Min	Average
ami33-1	1.17	1.21	75.13	78.56	0	4.04	6.97	1.17	1.18	67.99	73.58	12.64	15.39	0.00%	2.48%	9.50%	6.34%
ami33-2	1.21	1.23	76.20	83.12	0	3.68	6.87	1.17	1.17	76.08	82.16	15.04	16.44	3.31%	4.88%	0.16%	1.15%
ami33-3	1.19	1.24	90.67	99.02	0	3.48	6.29	1.17	1.18	88.16	90.57	13.56	15.19	1.68%	4.84%	2.77%	8.53%
ami49-1	37.00	38.87	1238.29	1436.56	0	5.83	22.64	36.44	37.13	1260.99	1323.91	50.19	56.67	1.51%	4.48%	-1.83%	7.84%
ami49-2	37.10	38.10	1316.60	1416.82	1	10.82	23.84	35.66	37.02	1306.20	1341.09	52.42	55.95	3.88%	2.83%	0.79%	5.34%
ami49-3	37.27	38.46	1287.51	1492.07	1	9.58	24.49	36.01	36.95	1204.94	1329.87	42.68	56.51	3.38%	3.93%	6.41%	10.87%

Experimental Results (3)



The best result of ami33-2 generated by our floorplanning algorithm when λ=0, M_T={14, 17, 21, 29}, M_L={1, 4, 23, 32}, M_B={8, 13, 15, 31}, and M_R = {10, 12, 18, 33}.

Experimental Results (4)



The best result of ami49-2 generated by our floorplanning algorithm when λ=0.0158, M_T={5, 7, 9, 10, 46}, M_L={11, 12, 16, 19, 21}, M_B={13, 14, 18, 23, 49}, and M_R = {8, 15, 20, 24, 38}.

Conclusion

- We have developed a quadratic-time method to transform a slicing floorplan into one that satisfying all given boundary constraints.
- We have extended Wong-Liu algorithm by incorporating our transformation method into it to solve slicing floorplan design with boundary-constrained modules.
- Our floorplanning algorithm ensures to always generate solutions satisfying all given boundary constraints.
- The experimental results indicate that our floorplanning algorithm can generate solutions with smaller area and interconnect wirelength than an existing algorithm.