A Practical Detailed Placement Algorithm under Multi-Cell Spacing Constraints

Yu-Hsiang Cheng, Ding-Wei Huang, Wai-Kei Mak, and Ting-Chi Wang

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan, 300

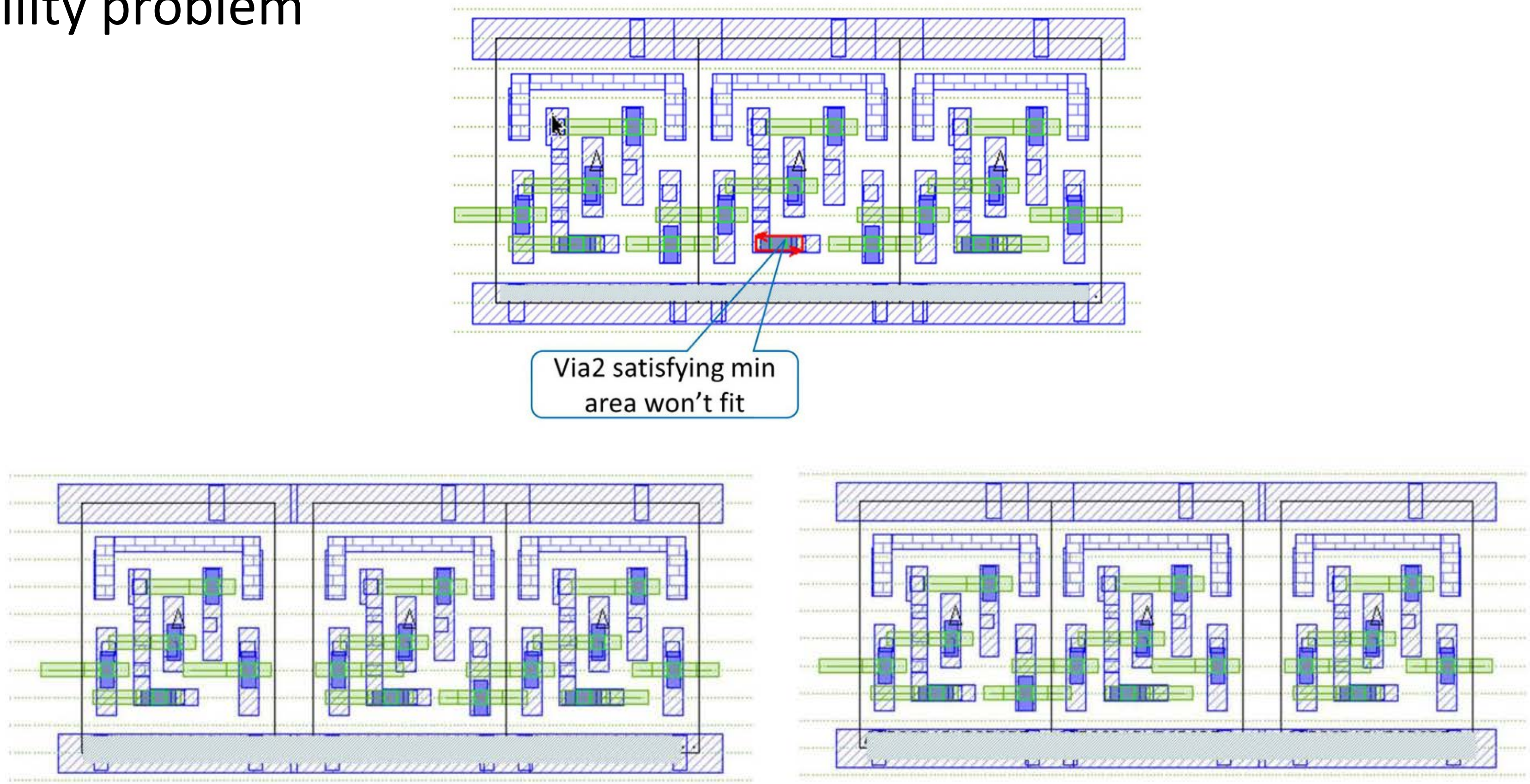
{slightencheng,nba23098,awkmak,tcwang.nthu}@gmail.com

Agenda

- Motivation
- Preliminaries
- Our Approach
  - Overall Flow
  - Constraint & Layout Analysis
  - Fast Violation Recognition (FVR)
  - Intra-Row Move (IRM)
    - SRDP
    - Cost Object
    - Extensions for Mixed-Cell-Height Designs
    - Acceleration
  - Global Move (GM)
- Experiment Setup & Results
- Conclusion

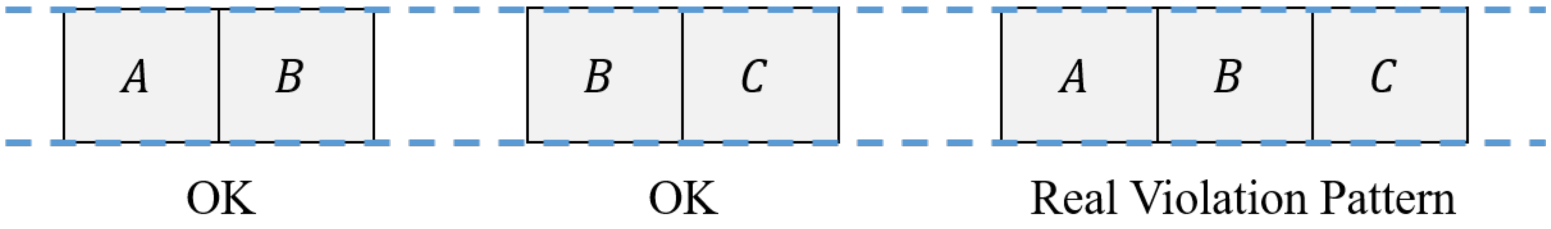
Motivation (1/2)

- Multi-cell spacing constraints arise due to manufacturing issues of aggressive technology scaling
- In sub-10nm nodes, we may impose multi-cell spacing constraints for pin accessibility problem



Motivation (2/2)

- Abstractly, a multi-cell spacing constraint addresses a forbidden pattern containing multiple cells.
- A naïve 2-cell method divides a multi-cell spacing constraint into several 2-cell constraints. However, it will lead to overkills.

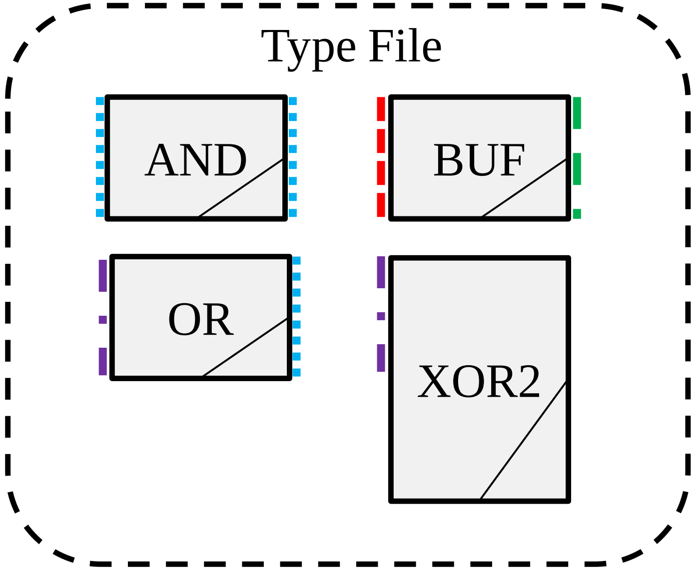


Our Contributions

- We propose a fast violation recognition (FVR) approach to rapidly find all constraint violations on a given layout.
- We propose a practical approach to perform detailed placement considering multi-cell spacing constraints.
    1. Constraint & Layout Analysis
    2. Intra-Row Move (Dynamic Programming-based)
    3. Global Move (Integer Linear Programming-based)
- By **cell virtualization** and **movable region computation** techniques, we can extend our intra-row move to handle **mixed-cell-height designs** without constructing a different dynamic programming model.

Definitions (1/3)

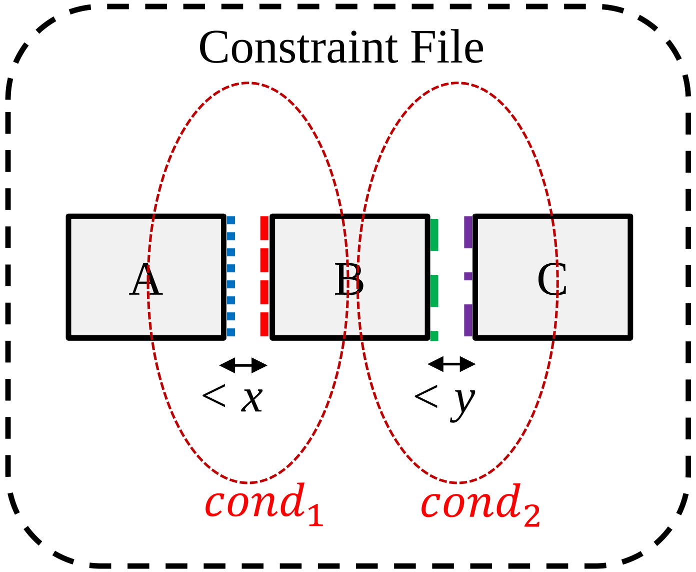
- **Segment**
  - The left and right edges of a cell are divided into one or more segments of one-row height long, depending on the cell height
- **Segment Type**
  - Each segment is associated with 0 or 1 segment type



Different colors represent different segment types

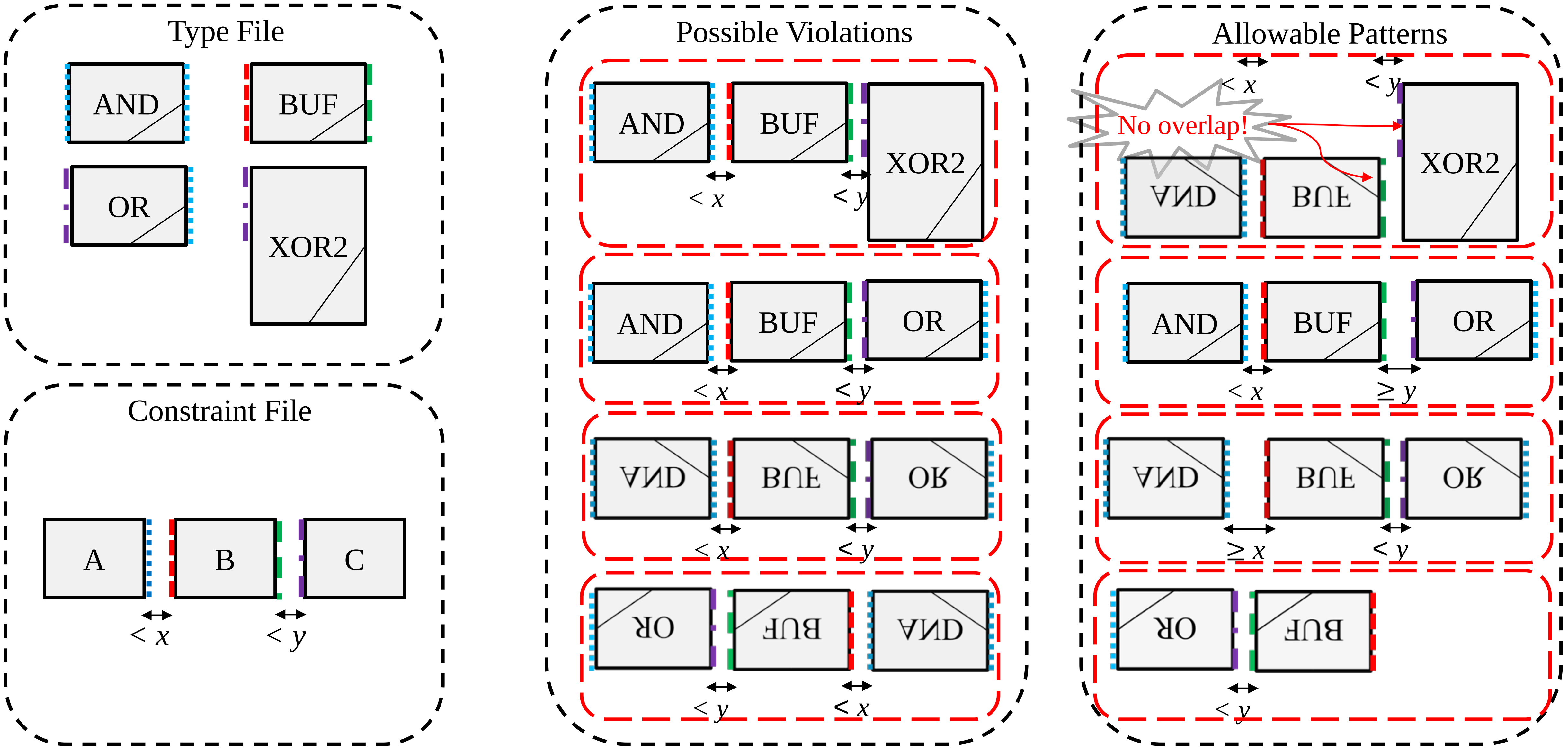
Definitions (2/3)

- **Condition**
  - A condition states that two specific segment types from two horizontally adjacent cells in a layout is less than a specified distance apart
- **Constraint & Constraint Violation**
  - A multi-cell spacing constraint is an ordered conjunction of multiple conditions
  - A **constraint violation** occurs ⟺ ∃ a group of cells on the given layout that makes all conditions in the constraint hold



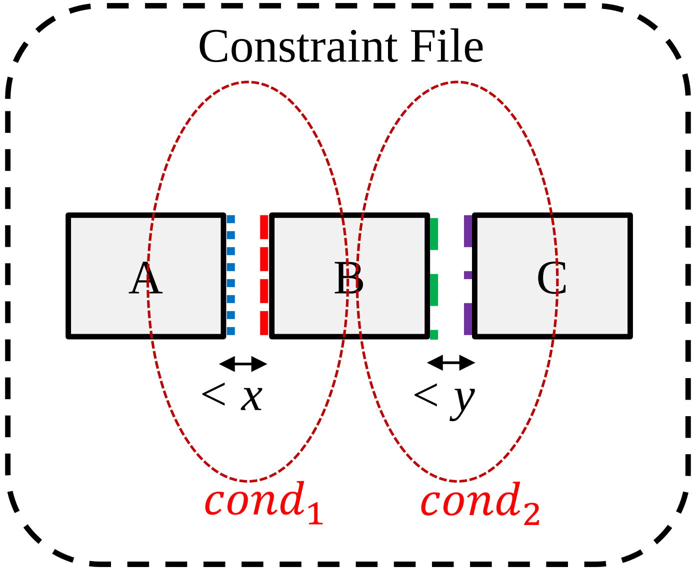
Constraint segments may be in given order from left to right, or vice versa

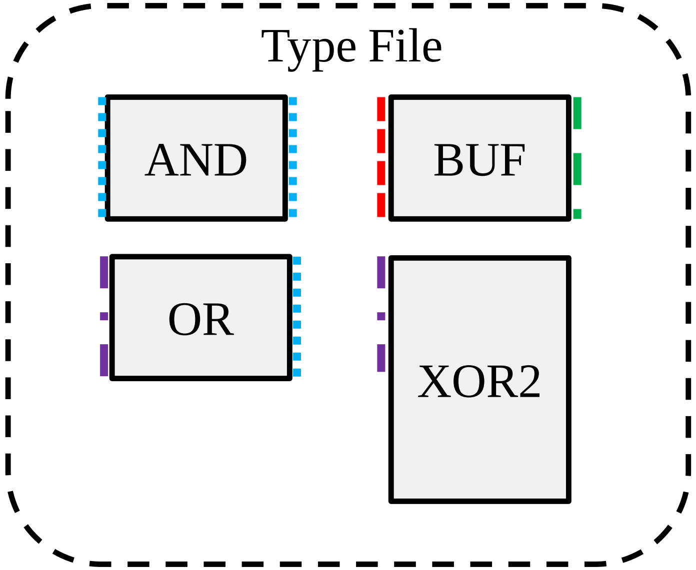
Multi-Cell Spacing Constraint Example



Definitions (3/3)

- **Cell Pair**
  - We call two horizontally adjacent cells a cell pair
- **Hit Pair**
  - A hit pair of a condition is a cell pair whose corresponding segment types match those specified in the condition.
    - e.g., hit pairs of Condition 1: (AND, BUF) and (OR, BUF), …
    - e.g., hit pairs of Condition 2: (BUF, OR) and (BUF, XOR2), …

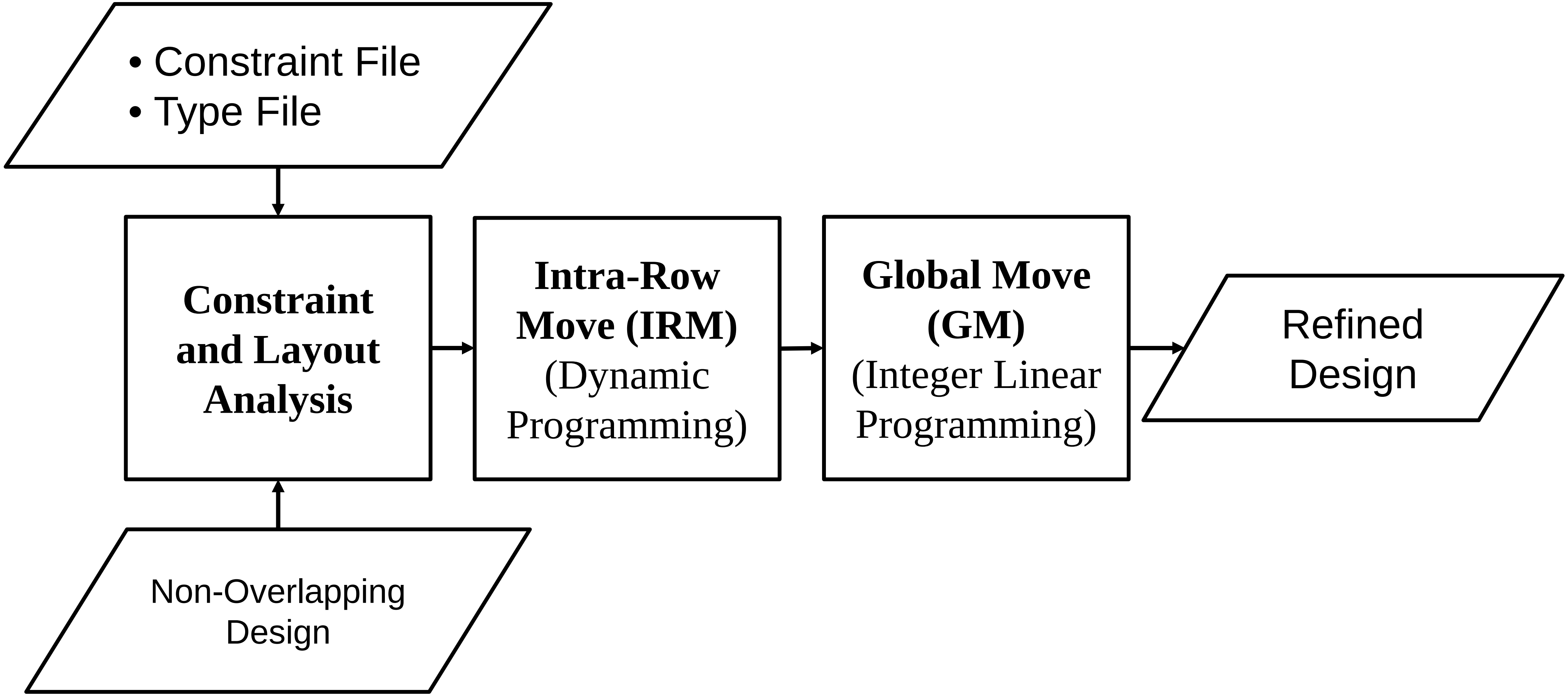




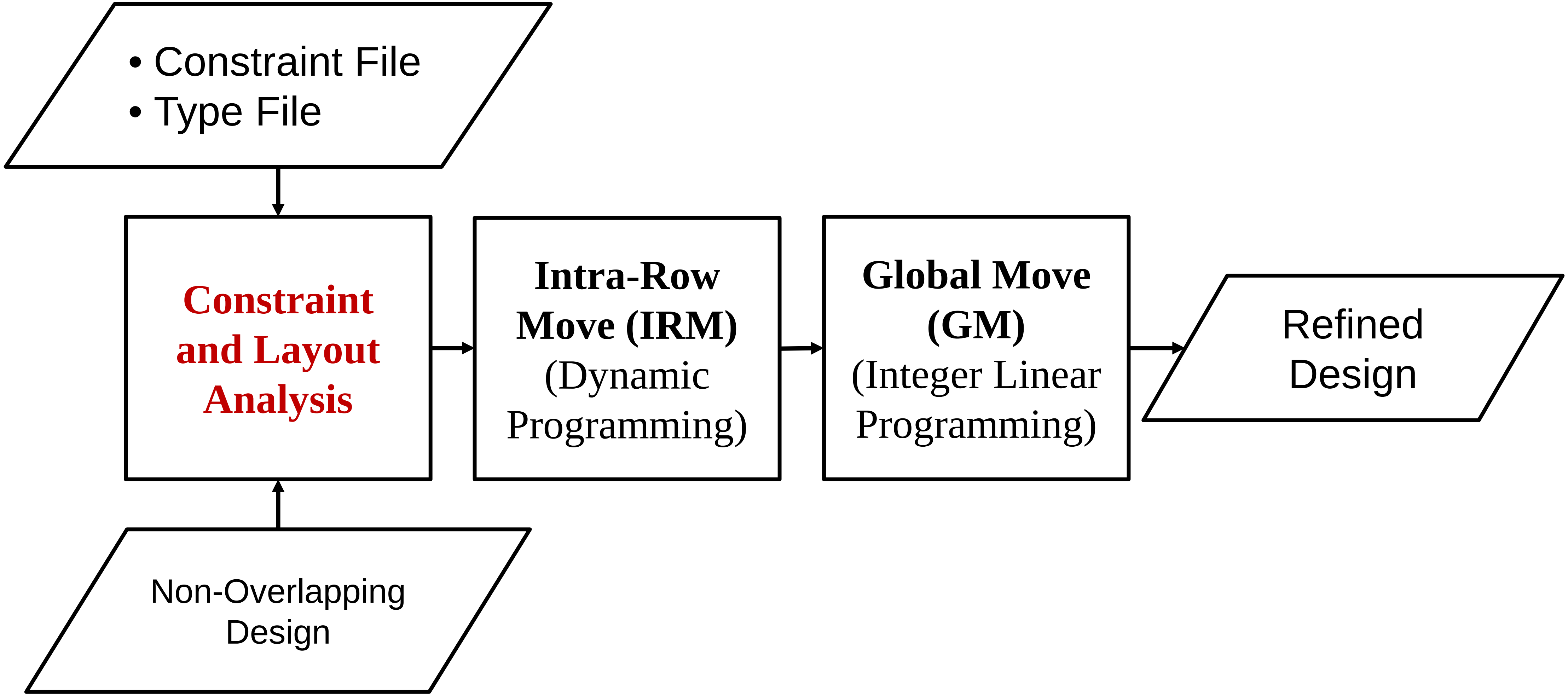
Problem Statement

- Given
  - a *non-overlapping initial placement* of a mixed-cell-height design
  - a set of segment types of cells
  - a set of multi-cell spacing constraints
- Refine the placement such that
  1. The number of **constraint violations** (i.e., forbidden patterns on layout) is minimized.
  2. Some other objectives such as **total cell displacement** and/or **wirelength increase** are minimized.

Overall Flow



Overall Flow



Constraint and Layout Analysis (Critical Condition)

- For each condition $cond$, we define its breaking cost b_{cost} as follows.

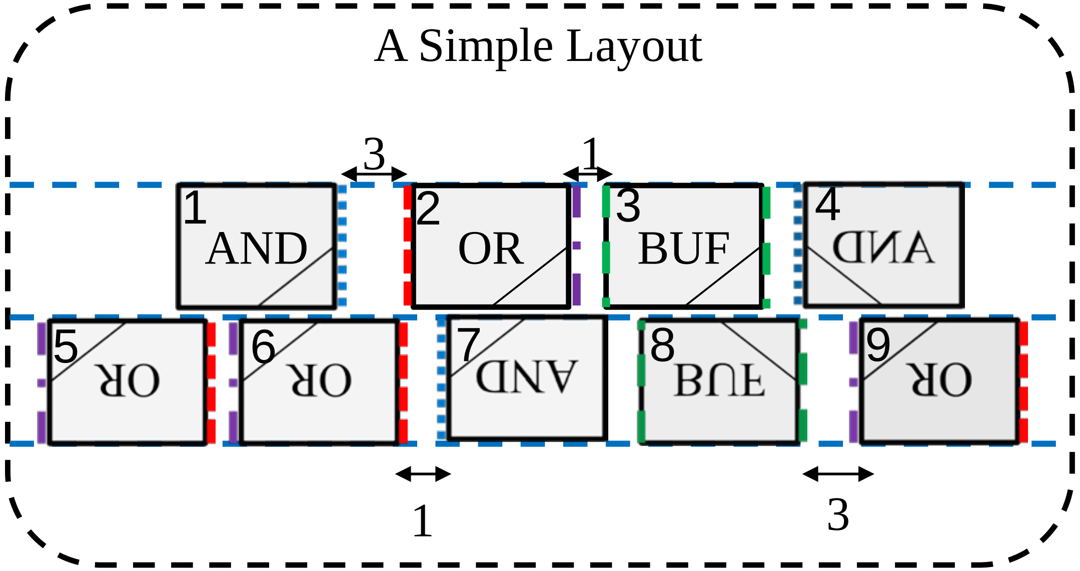
$$b_{cost}(cond) = \sum_{\forall cp \in cond.HP} \max(0, rs(cond) - cs(cp))$$

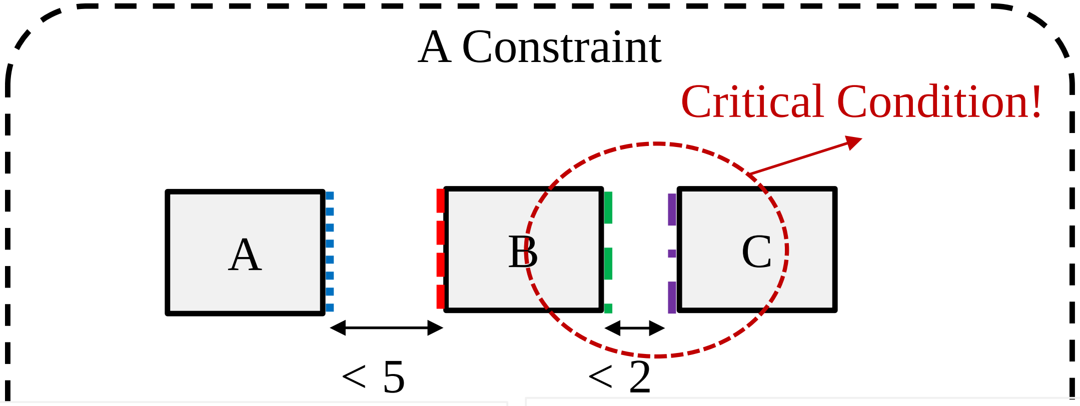
  - cp: a cell pair in the layout
  - $cond.HP$: set of hit pairs of $cond$
  - $rs(cond)$: required spacing of $cond$
  - $cs(cp)$: current spacing of cp

- For constraint $constr$, its critical condition is the one with minimum b_{cost} among all its conditions

$$cond_{crit} = \underset{cond \in constr}{\arg\min} \{b_{cost}(cond)\}$$

i.e., easiest to break in current layout





$b_{cost} = \max(5-3, 0) + \max(5-1, 0) = 2 + 4 = 6$

$b_{cost} = \max(2-1, 0) + \max(2-3, 0) = 1 + 0 = 1$

Fast Violation Recognition (FVR) (1/2)

- We want to
  - rapidly find all multi-cell spacing constraint violations on the given layout
  - tag all active critical cell pairs that make any critical condition hold
- Time complexity: $O(nt)$
  - n: total #cells
  - t: total #conditions

Fast Violation Recognition (FVR) (2/2)

- For a constraint with *k* conditions, use
  - a *k*-bit integer variable as a flag
  - a **queue** to record order of occurrence
- Example
  - Constraint 1 = cond1·cond2·cond3
  - Constraint 2 = cond4·cond5

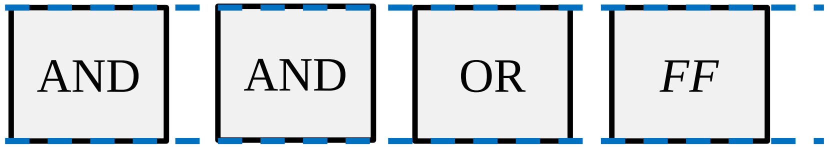
cond1: S < 3, (AND,AND)/(AND,XOR)

cond2: S < 5, (AND,OR)

cond3: S < 3, (OR,FF)

cond4: S < 4, (AND,AND)

cond5: S < 3, (FF,AND)



Initialization
FLAG1: 0 0 0
FLAG2: 0 0

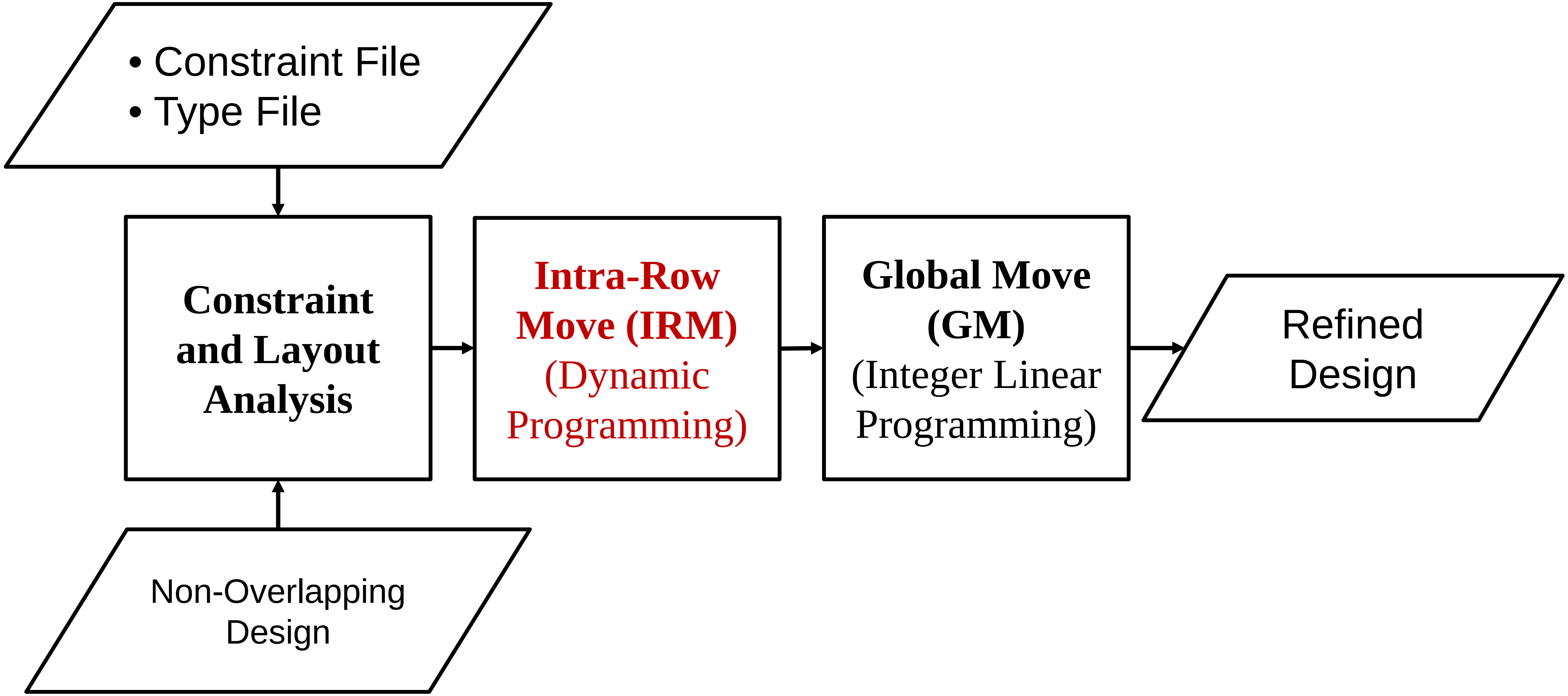
(AND,AND, spacing = 2)
FLAG1: 1 0 0 (enqueue 1)
FLAG2: 1 0 (enqueue 1)

(AND,OR, spacing = 4)
FLAG1: 1 1 0 (enqueue 2)
FLAG2: 0 0 (empty queue)

(OR,FF, spacing = 1)
FLAG1: 1 1 1 (all 1 && correct order => #Vios++)
=> 0 1 1 (dequeue)
FLAG2: 0 0

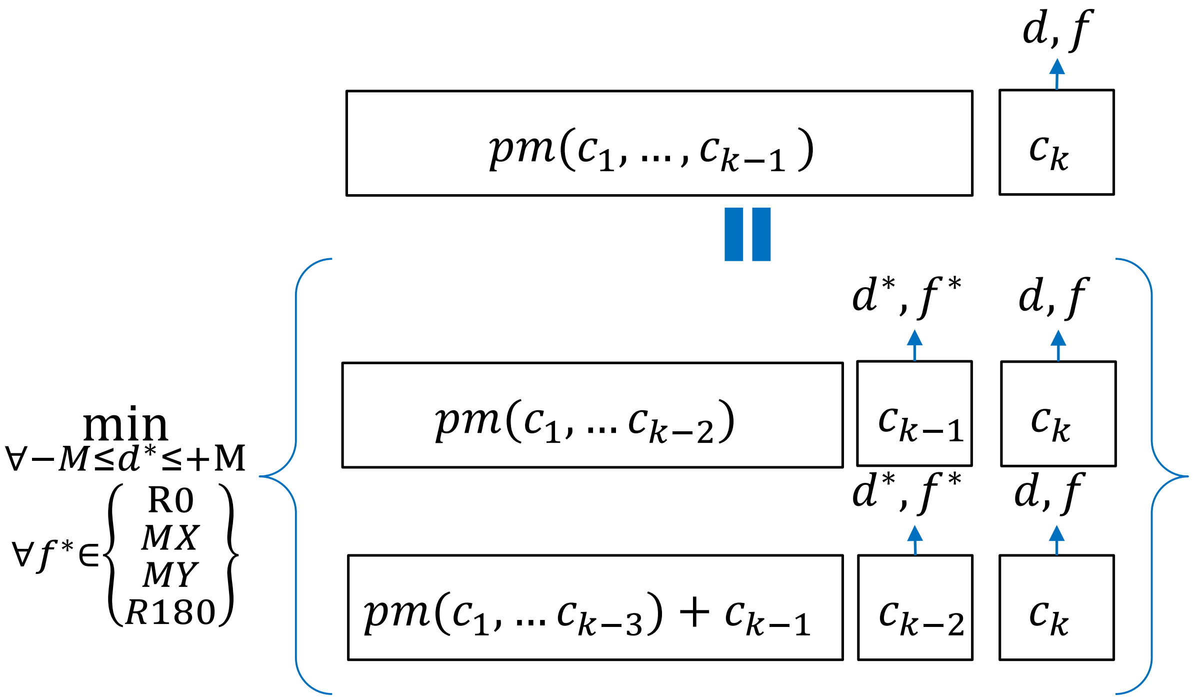
(OR,FF, spacing = 6)
FLAG1: 0 0 0 (empty queue)
FLAG2: 0 0

Overall Flow



Intra-Row Move: SRDP (1/2)

- Perform **SRDP** on a row each time
  - Allow **flipping**, **shifting**, and **adjacent swapping**
    - Time Complexity: $O(nM^2)$
      - n: #cells in this row
      - M: max shifting amount
  - 3 cases
    - **Case 1**



MX and R180 are forbidden for odd row-height cells

Base cases:

$$\{P_1(c_0, d, f) = \infty, P_1(c_k, d, f) = \Delta P_1(-, c_k, -, d, -, f):$$
$$\forall k \in \{1, 2\}, f \in \{R0, MX, MY, R180\}, -M \le d \le M\}$$

Recursive formulas:

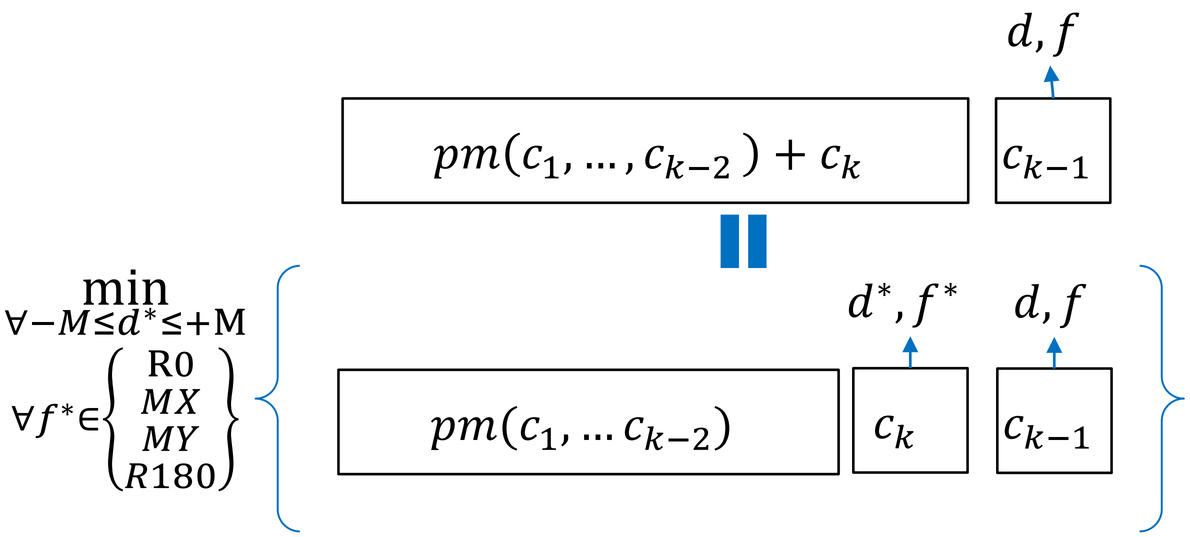
$$P_k(c_k, d, f) = \min_{\substack{\forall f^* \in \{R0, MX, MY, R180\} \\ \forall -M \le d^* \le M}} \{P_{k-1}(c_{k-1}, d^*, f^*) + \Delta P_k(c_{k-1}, c_k, d^*, d, f^*, f), P_{k-1}(c_{k-2}, d^*, f^*) + \Delta P_k(c_{k-2}, c_k, d^*, d, f^*, f\} \quad (3.12a)$$

$$P_k(c_{k-1}, d, f) = \min_{\substack{\forall f^* \in \{R0, MX, MY, R180\} \\ \forall -M \le d^* \le M}} \{P_{k-1}(c_k, d^*, f^*) + \Delta P_k(c_k, c_{k-1}, d^*, d, f^*, f)\} \quad (3.12b)$$

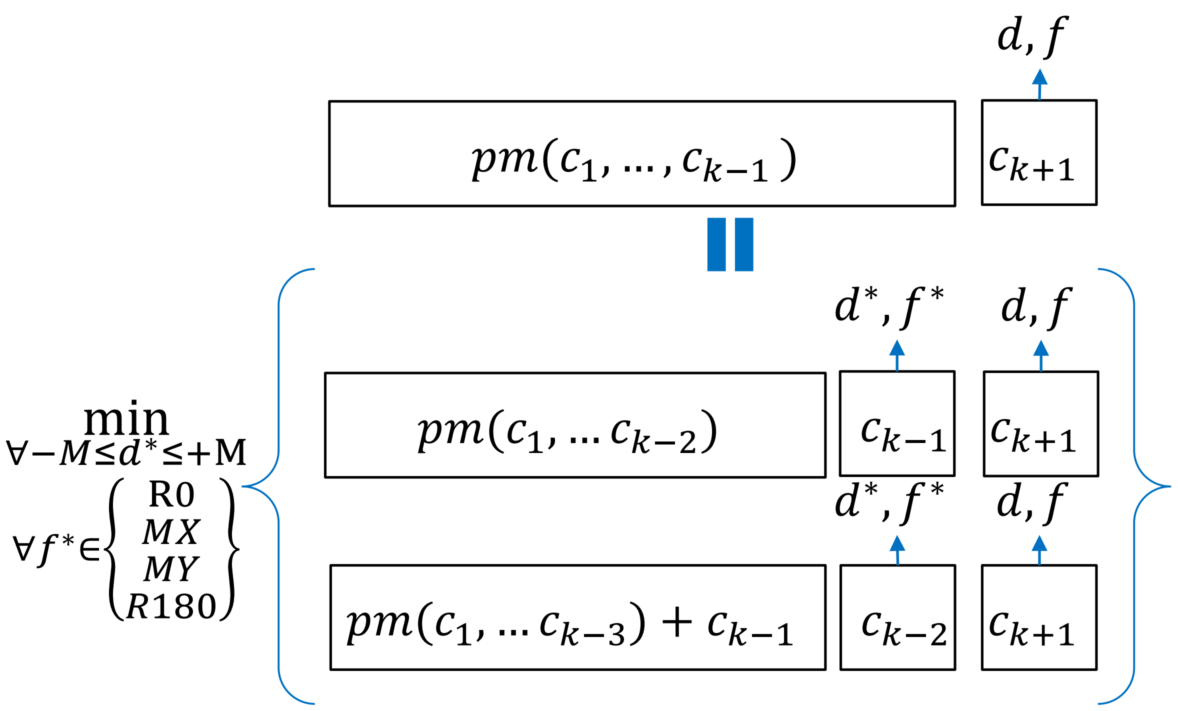
$$P_k(c_{k+1}, d, f) = \min_{\substack{\forall f^* \in \{R0, MX, MY, R180\} \\ \forall -M \le d^* \le M}} \{P_{k-1}(c_{k-1}, d^*, f^*) + \Delta P_k(c_{k-1}, c_{k+1}, d^*, d, f^*, f), P_{k-1}(c_{k-2}, d^*, f^*) + \Delta P_k(c_{k-2}, c_{k+1}, d^*, d, f^*, f)\} \quad (3.12c)$$

Intra-Row Move: SRDP (2/2)

Case 2



Case 3



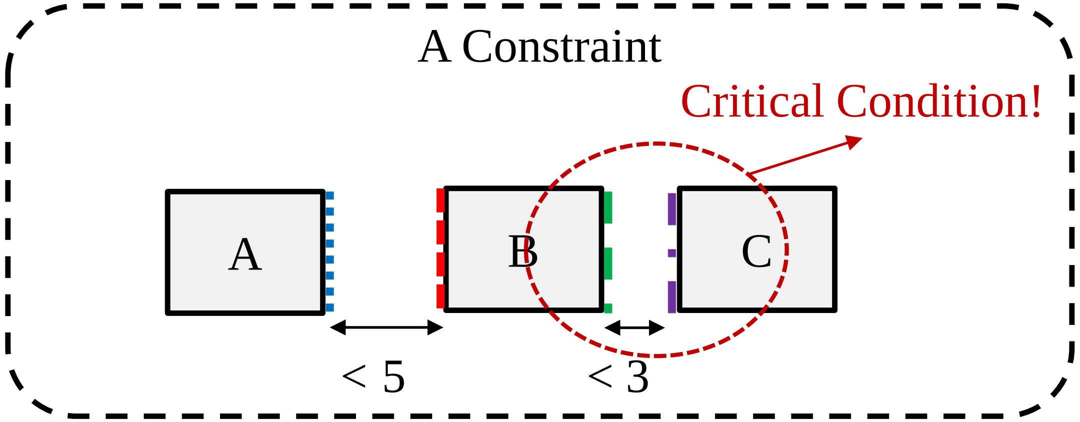
Cost Computation (1/2)

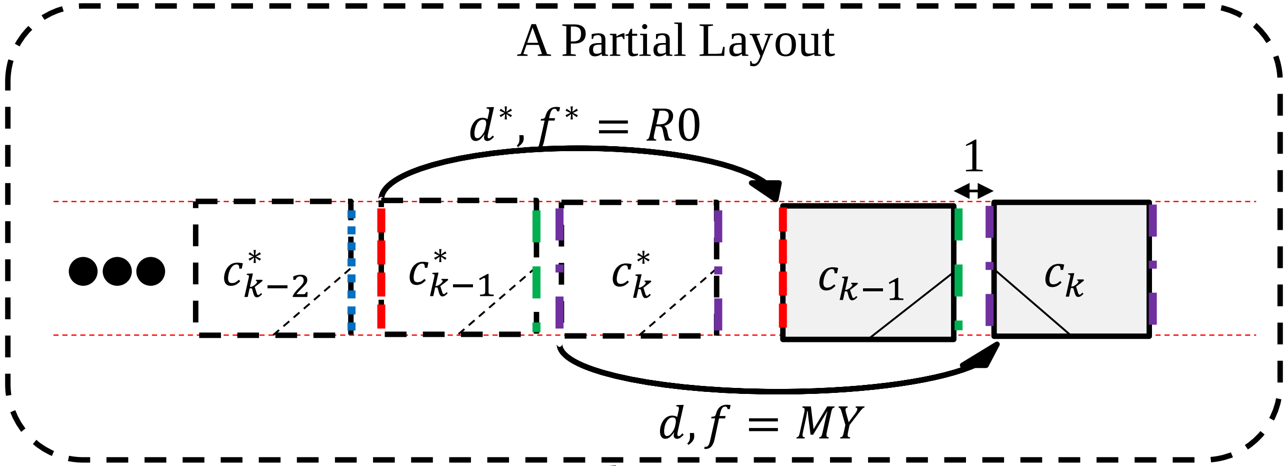
- 4 Items
  1. Accumulated spacing deficit of active critical cell pairs
  2. Accumulated net span cost [APA99]
     1. This term for **W**ire**L**ength **F**irst (**WLF**) mode
     2. For displacement first mode (**ORIG**), this term is always equal to 0
  3. Accumulated cell displacement
  4. Accumulated number of cell flips

- The spacing deficit of a cell pair cp is defined as follows

$$Deficit(cp) = \max_{\forall cond \wedge cp \in cond.HP} (0, reqSpace(cond) - curSpace(cp))$$

Cost Computation (2/2)





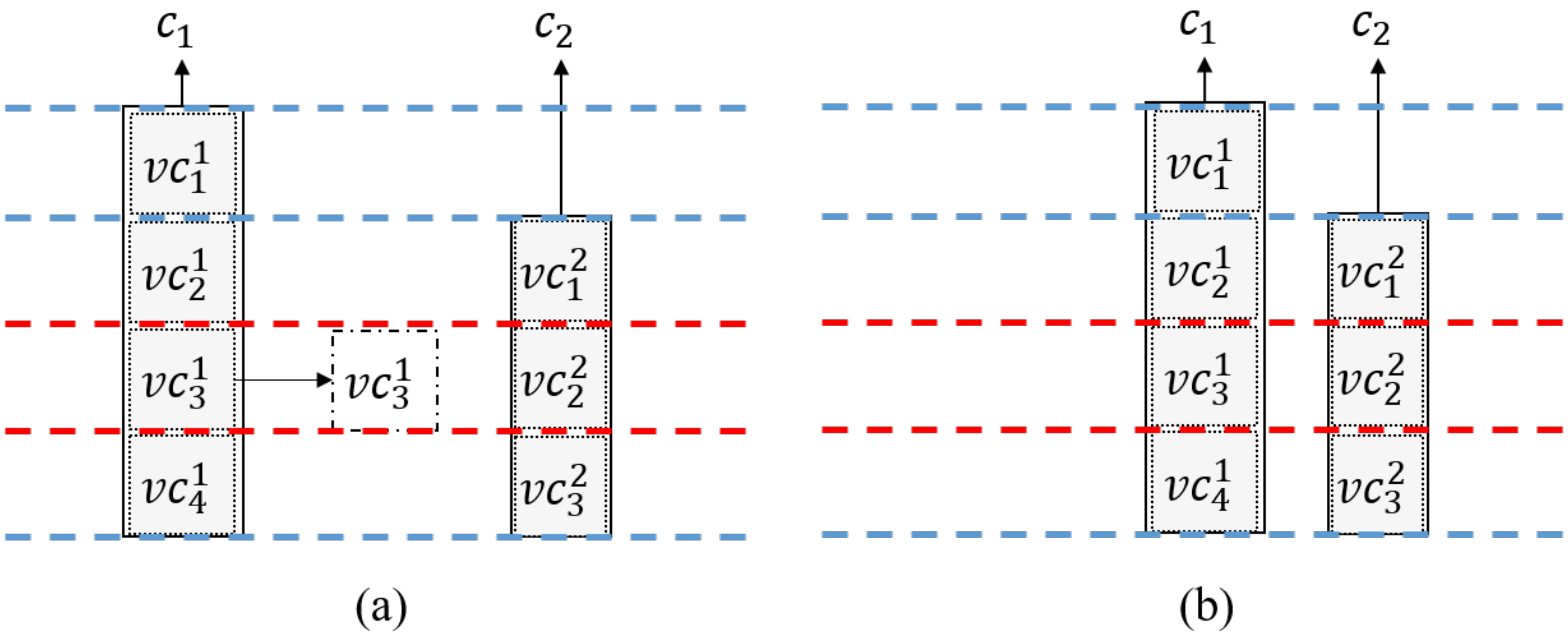
$$\Delta P_k(c_{k-1}, c_k, d^*, d, f^*, f) = \begin{cases} \text{Item (1): } \max(0, 3-1) = 2 \\ \text{Item (2): } \textbf{ORIG}\text{: 0, } \textbf{WLF}\text{: } \Delta SpanCost(c_k) \\ \text{Item (3): } d \\ \text{Item (4): } 1 \end{cases}$$

Extensions for Mixed-Cell-Height Designs

- 2 techniques
  - **Cell Virtualization**
  - **Movable Region Computation**

Technique 1: Cell Virtualization

- Slice each multi-row height cell (parent) into several single-row height cells (children)
  - Parent: a real cell
  - Children: virtual cells
- Whenever a child moves, its parent and all siblings move (vertical alignment)



Technique 2: Movable Region Computation (1/2)

- The remaining problem is how to prevent a multi-row height real cell from overlapping cells in other rows after it is relocated.
- Each virtual cell has a **movable region (MR)**.
- Compute Left/Right Movable Distance MD_l, MD_r by (see example on next page)

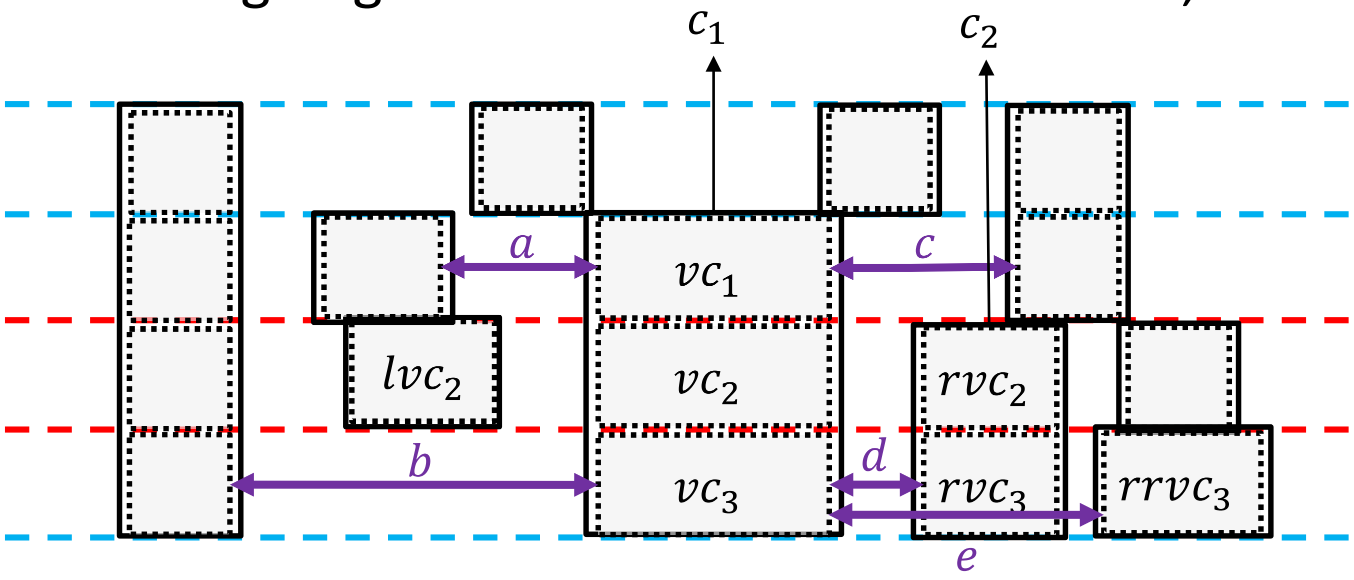
$$MD_l(vc_i) = \min_{\forall vc \in c, vc \neq vc_i} \{cs(vs, ltvc)\}$$
$$MD_r(vc_i) = \min_{\forall vc \in c, vc \neq vc_i} \{cs(vs, rtvc)\}$$

where

$$ltvc = \begin{cases} lvc, & if\ lvc.p \neq lvc_i.p \\ llvc, & otherwise \end{cases}$$
$$rtvc = \begin{cases} rvc, & if\ rvc.p \neq rvc_i.p \\ rrvc, & otherwise \end{cases}$$

Technique 2: Movable Region Computation (2/2)

- Movable Region (MR) of $vc_i = [vc_i.x_l - MD_l(vc_i), vc_i.x_r + MD_r(vc_i)]$
  - e.g., $MR(vc_2) = [vc_2.x_l - a, vc_2.x_r + c]$
- If a cell is going to be relocated outside its MR, cost = ∞



$$\begin{cases} MD_l(vc_2) = \min(a, b) = a \\ MD_r(vc_2) = \min(c, e) = c \end{cases}$$

- Current spacing (cs) should be updated after SRDP processed a row i
  - Need to re-compute cs of all cells in **relevant rows** only

  (i.e., rows containing all processed virtual cells in row i and their siblings)

Intra-Row Move

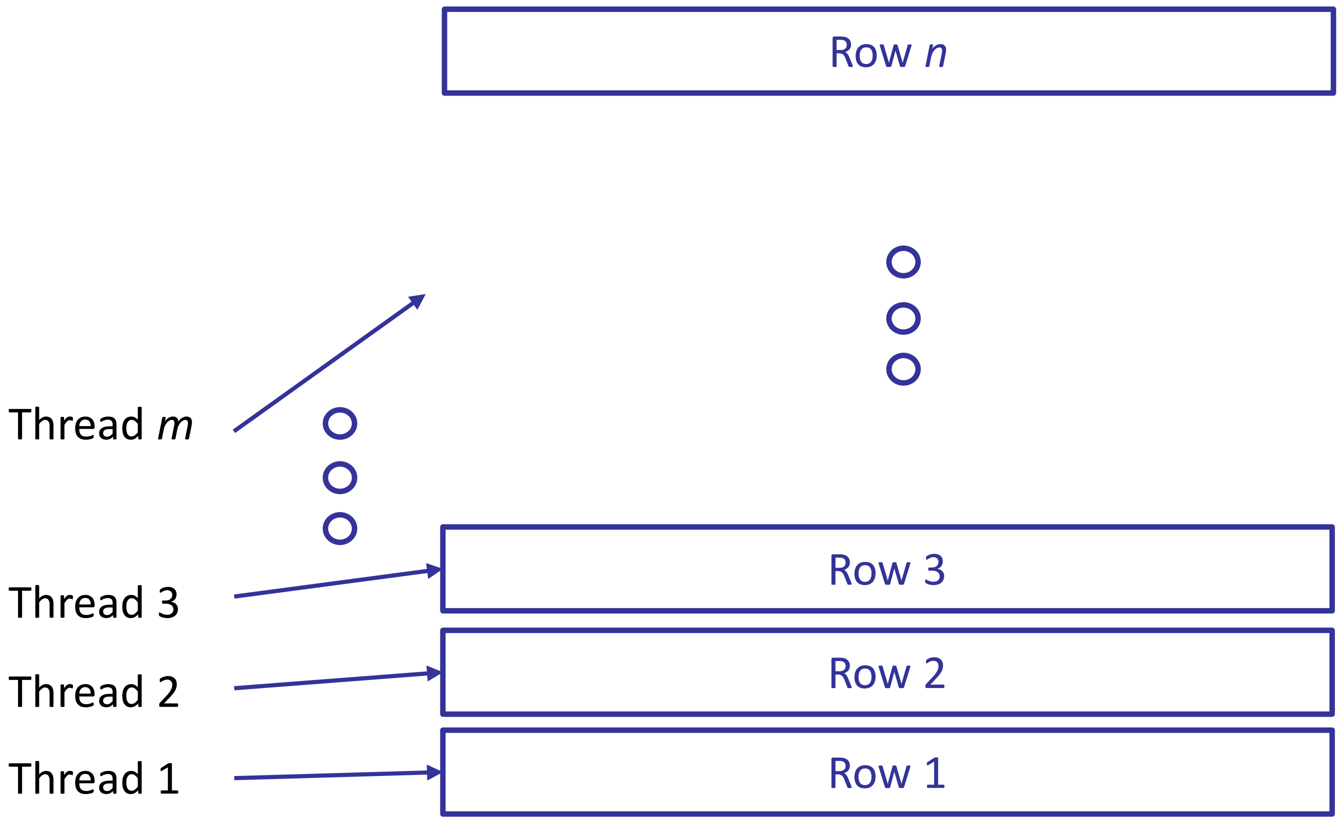
ALGORITHM 3: Intra-Row Move

Data: A maximum number of rounds denoted by $round_{max}$, a set $setRows$ containing the IDs of rows that will be processed by $SRDP()$, a maximum cell shifting amount M, and all related configurations.

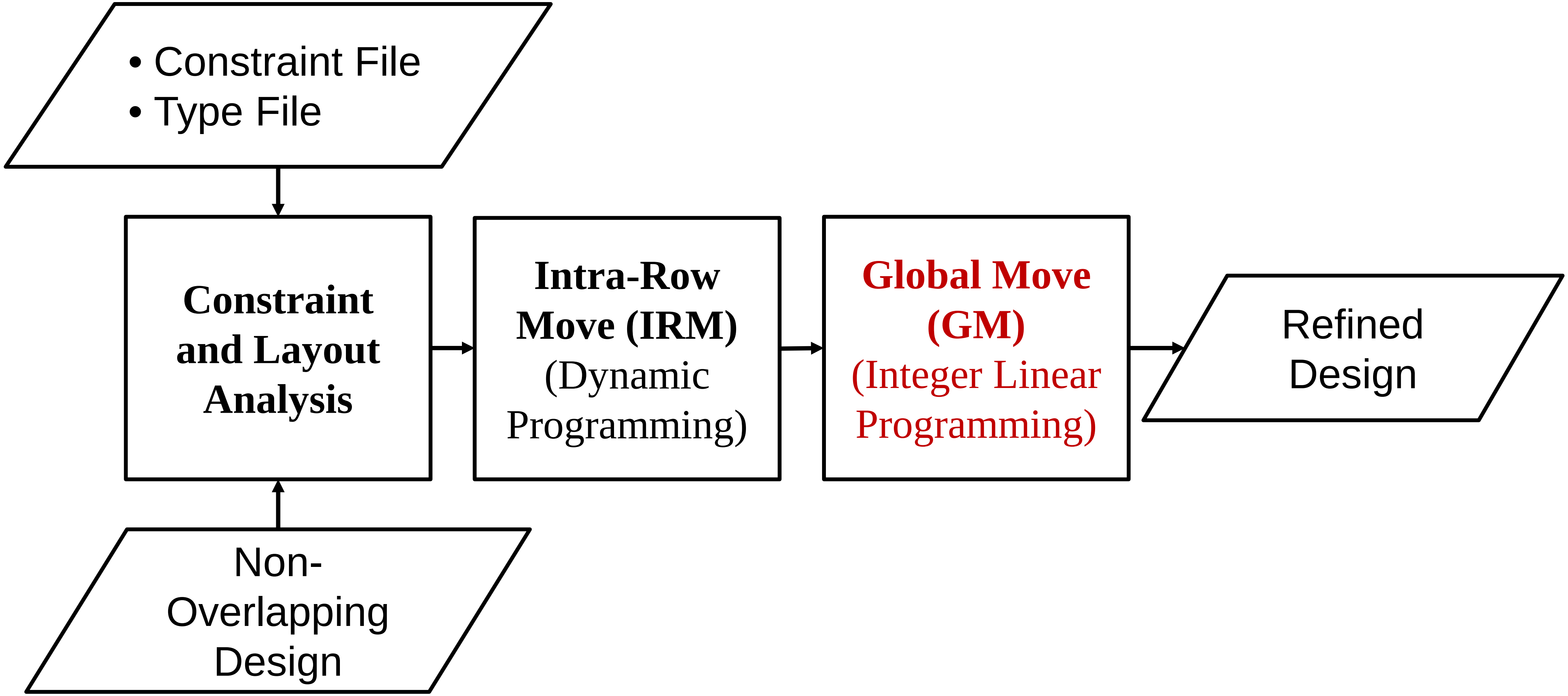
```
Call FVR() and calculate HPWL
for round = 1 to round_max do
    foreach row ∈ setRows do
        Compute movable regions for all virtual cells in row.
        Call SRDP(row, M)
        Update MD_l and MD_r of all cells in relevant rows.
    Call FVR() and calculate HPWL
    if solution converges then return                // little Improvement
```

Acceleration

- Parallelization (Single-Cell-Height Designs, ORIG mode)
  - Let each thread take a row.



Overall Flow



Global Move (GM)

- To resolve remaining constraint violations after IRM
- A global move contains 3 parts:
  1. *Candidate Cells Finding*
     - All violated cells
  2. *Candidate Empty Spaces Finding and Best Location Finding*
     - For each candidate cell c, find and record the best location of all available empty spaces without making any condition hold into $c.CESL$
  3. *Best Candidate Empty Spaces Choosing and Cell Moving*
     - ILP
       - $\min \alpha \times \#violations + \beta \times \#totalCellDisplacement$
- A multi-round GM would terminate when the number of constraint violations stops to decrease.

Experimental Results

- Experiment Setup
  - Language: C++
  - ILP: Gurobi Optimizer
  - Parallelization: OpenMP
  - Design Explorer: OpenGL
  - System: CentOS 6.9
  - CPU: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz with 32 threads
  - Designs
    - Single-cell-height: OpenCores
    - Mixed-cell-height: ICCAD 2017 Contest Problem C
  - Cell Library: NanGate 15nm Open Cell Library
  - Synthesis: Synopsys Design Compiler Graphical
  - Initial Placement: Cadence Encounter Digital Implementation System

Benchmarks

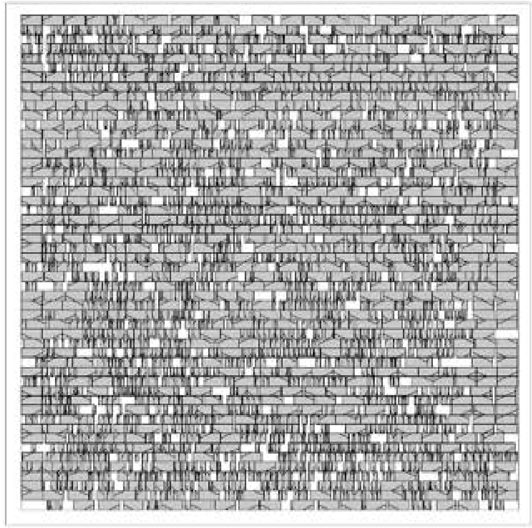
- 4 single-cell-height, 3 mixed-cell-height

Table 2: Benchmarks

Design	M.H.?	Util.	#Cells	Stats.	#Nets	#Rows
sdrc	No	80%	2814	100% 1X	3068	52
aes	No	80%	8345	100% 1X	8408	86
ecg	No	80%	71632	100% 1X	72176	251
tbpc	No	80%	292701	100% 1X	293481	526
pci	Yes	59.7%	29521	90.4% 1X; 6.1% 2X; 2.0% 3X; 1.5% 4X; 4 Macros	29989	200
des	Yes	55.0%	112644	94.8% 1X; 5.2% 2X	112882	300
edit	Yes	67.4%	130661	90.3% 1X; 6.1% 2X; 2.1% 3X; 1.5% 4X	133227	361

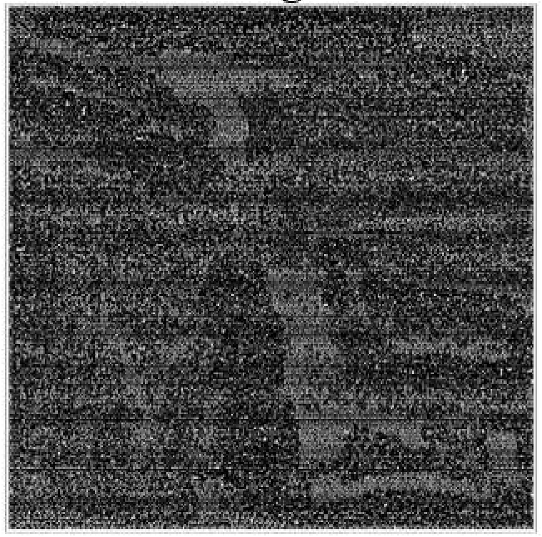
Benchmarks

sdrc



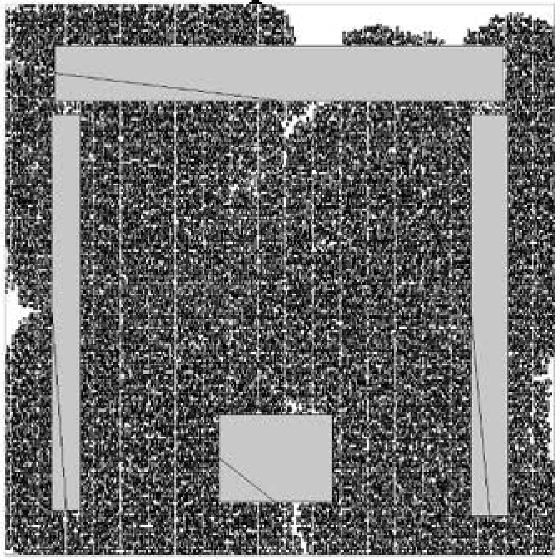
aes

ecg

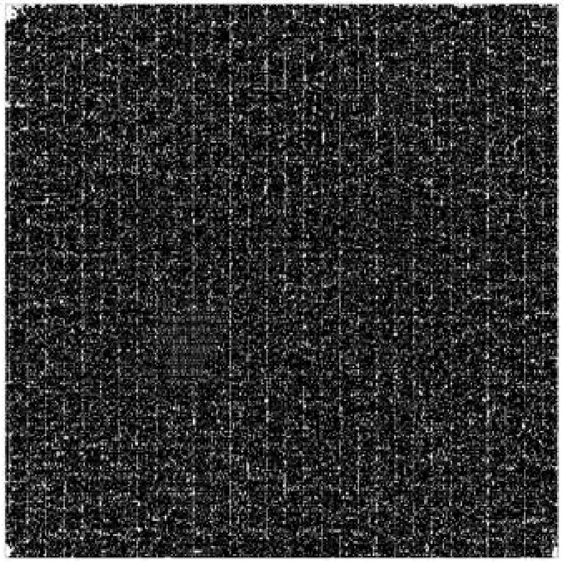


tbpc

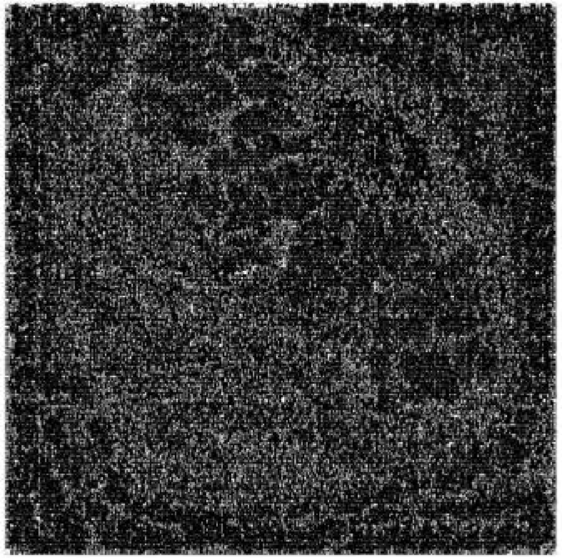
pci



des



edit



Results of Different Modes: Table

- A 2-cell approach
  - Each condition is a critical condition
  - Slightly modify FVR to find 2-cell spacing constraint violations
- Our multi-cell approach can resolve all constraint violations with a better total cell displacement (> 3x less), wirelength (> 2x less), and runtime (up to 33% less) than those in a 2-cell approach

Table 3: Results of Different Modes

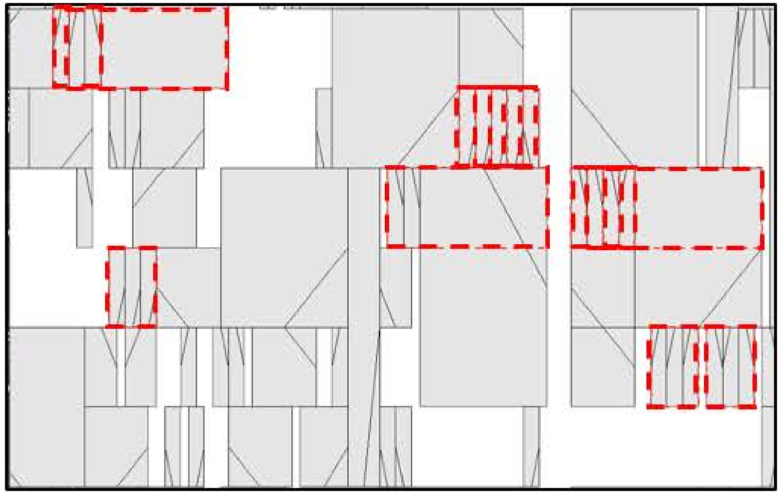
Design	HPWL (μm)	Perc.	#Vios.	#Vios. Remained				Disp. (sites)				ΔHPWL				Runtime (sec.)			
				ORIG2	ORIG	WLF2	WLF	ORIG2	ORIG	WLF2	WLF	ORIG2	ORIG	WLF2	WLF	ORIG2	ORIG	WLF2	WLF
sdrc	9224	20.93%	309 [656]	0 [0]	0	0 [1]	0	1830	540	4439	3301	+0.68%	+0.15%	-0.46%	-1.73%	0.91 (0.82)	0.81 (0.75)	1.48	1.22
aes	46841	16.75%	718 [1367]	0 [2]	0	0 [2]	0	7174	1930	16836	11184	+0.54%	+0.04%	-0.11%	-1.30%	2.26 (1.27)	1.43 (1.03)	2.96	2.63
ecg	302729	13.78%	4529 [13186]	0 [2]	0	1 [10]	0	28855	7709	91047	72389	+0.27%	+0.05%	-0.64%	-1.18%	14.46 (5.14)	8.83 (4.18)	23.71	21.66
tbpc	1629848	9.37%	13190 [32689]	16 [47]	0	27 [136]	0	108559	30654	348805	266971	+0.23%	+0.04%	-0.31%	-0.77%	56.68 (20.21)	34.70 (15.02)	122.03	110.47
pci	365161	19.01%	2438 [6605]	0 [4]	0	0 [2]	0	21534	6012	45379	33535	+0.66%	+0.12%	-0.79%	-1.77%	25.81 [5.19]	14.03 [0.933]	31.94 [10.35]	22.58 [1.25]
des	1520613	10.51%	5123 [23015]	8 [107]	0	5 [138]	0	37665	9907	129624	103302	+0.28%	+0.05%	-0.77%	-1.01%	92.91 [35.16]	49.32 [6.11]	125.05 [55.59]	82.28 [14.34]
edit	3590442	9.25%	5337 [16485]	0 [6]	0	0 [8]	0	24289	9640	137219	122595	+0.03%	+0.01%	-0.57%	-0.59%	94.80 [30.67]	64.70 [10.81]	119.51 [34.12]	96.60 [11.89]
Ratio	-	-	-	-	-	-	-	3.47	1.00	10.71	8.39	-	-	-	-	1.62	1.00	2.39	1.96
Avg.	-	-	-	3 [24]	0	5 [42]	0	-	-	-	-	+0.38%	+0.07%	-0.52%	-1.19%	-	-	-	-

Results of Different Modes: Example

2-cell method

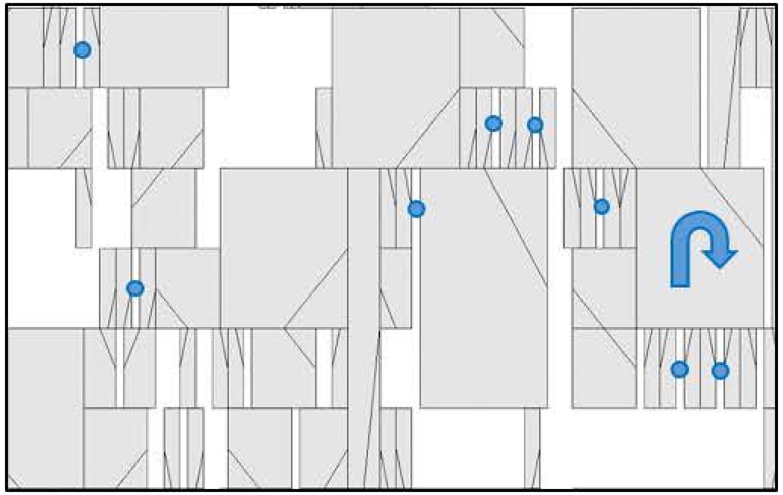
Our (multi-cell) method

Initial Layout



ORIG2

ORIG



Conclusions

- We proposed a practical detailed placement approach considering multi-cell spacing constraints
- We proposed **cell virtualization** and **movable region computation** techniques to extend IRM to handle **mixed-cell-height designs**
- Experiment results showed the efficiency and effectiveness of our approach

Thanks