HPWL-driven & Timing-aware Detailed Placement

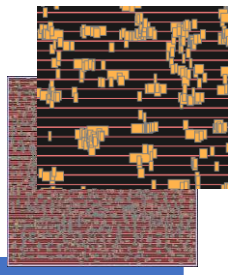
Christos Sotiriou

Christos Georgakidis

Circuits and Systems Lab (CASLab)
University of Thessaly
Volos - Greece

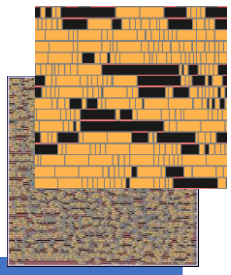
1

Placement Overview



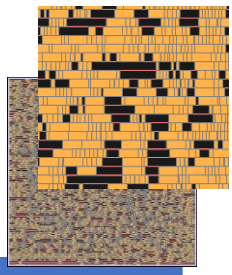
Global Placement

- Initial cell locations with overlaps



Legalisation

- Legalise cells to core rows and power grids resolving any overlaps



Detailed Placement

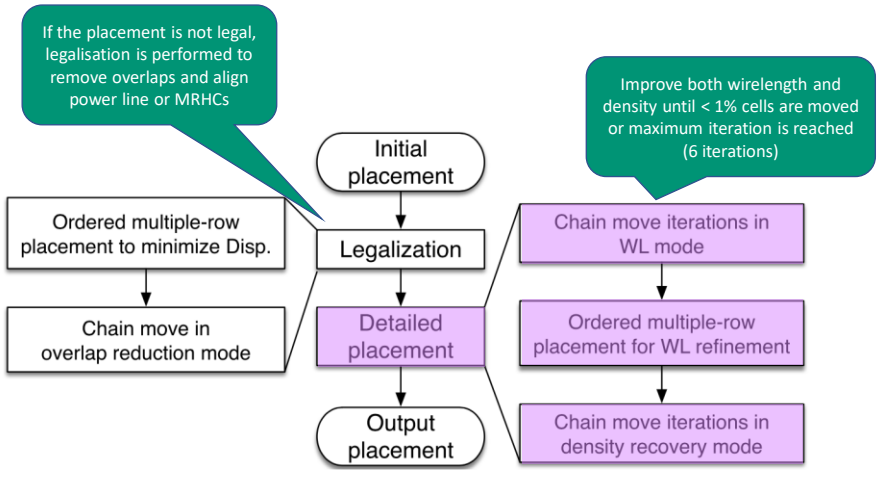
- Iterative Placement optimisations targeting one or more metrics, e.g. HPWL, timing, …

2

MrDP: Multiple-Row Detailed Placement of Heterogeneous-Sized Cells for Advanced Nodes

3

Overall Flow



4

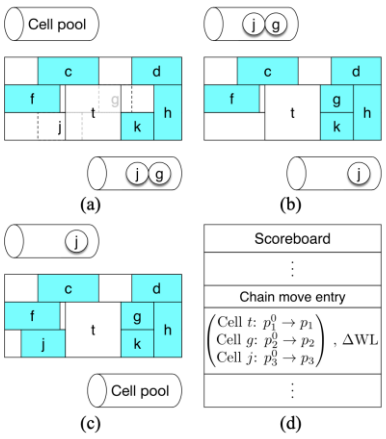
Chain Move Algorithm – Basic Structures (1)

- Chain Move
  - Each chain move contains a set of movements for one or several cells
  - A chain move involving multiple cells is usually triggered by the attempts of inserting a cell into a position resulting in overlaps with existing cells in the region
    - Overlapped cells need to find new positions to resolve overlaps
- Cell Pool
  - Queue structure used for temporary storage of cells within a chain move
  - Cells in the cell pool have higher priority compared to the rest of the cells
    - They are popped out and placed until the cell pool goes empty, which indicated the end of a chain move
    - After the chain move goes empty, the rest cells can be examined for movement

5

Chain Move Algorithm – Basic Structures (2)

- Scoreboard
  - An array of chain move entries with corresponding changes in the cost metric for each chain move
    - Similar to partitioning algorithms (KL, FM)
  - Chain move → Annealing Process
    - Allows temporal degredation of cost as long as it eventually achieves better solutions
    - Help escape local minimum



6

Chain Move Algorithm – Pseudocode

Algorithm 1 Chain Move Algorithm

Require: A set of placed cells C in the layout.
Ensure: Move cells to minimize wirelength cost.

```
1: ReorderCells(C);
2: Re-structure C as a queue;
3: while C is not empty or Pool is not empty do
4:     if Pool is not empty then
5:         c_i ← Pool.pop();
6:     else
7:         c_i ← C.pop();
8:         if c_i has already been moved then
9:             Continue;
10:        end if
11:    end if
12:    r_i ← ComputeSearchRegion(c_i);
13:    A_i ← collect candidate positions in r_i;
14:    cost_b ← ∞;
15:    for each a_j ∈ A_i do
16:        (cost_i, p_i, O_i) ← ComputeMoveCost(c_i, a_j);
17:        if cost_i < cost_b then
18:            cost_b ← cost_i; p_b ← p_i; O_b ← O_i;
19:        end if
20:    end for
21:    Move c_i to p_b;
22:    Pool.push(O_b);
23:    Board.last.append(c_i, p_i^0 → p_b);
24:    if Pool is empty then
25:        Compute ΔWL for Board.last;
26:    end if
27: end while
28: BacktraceToBestEntry(C, Board);
```

Table 1: Notations used in Chain Move Algorithm

$Pool$	The cell pool.
$Board$	The scoreboard.
p_i^0	Initial position of Cell c_i.
p_i	Candidate position of cell c_i.
O_i	The set of cells overlapping with cell c_i at p_i.
$cost_i$	The cost of cell c_i at p_i.
$p_b, O_b, cost_b$	Correspond to best $p_i, O_i, cost_i$, respectively.

Search region in terms of HPWL for cell ci

Extraction of candidate legal locations inside the optimal region (consider power line alignment, blockages, fixed macros, visited cell)

For each candidate location, the movement cost is computed, the overlapping cells are extracted and the best location in terms of cost is selected

Move the examining cell to the new position and insert the overlapping cell into the Cellpool

Update the Scoreboard

Revert to the placement from the Scoreboard with the best cost improvement

7

Chain Move Algorithm – Details

- Chain-move Constraints
  - Each cell is allowed to be moved only once during one DP iteration
  - A cell may fail to find any legal position in its search region
    - When all cells in a dense region have already been moved in this pass
    - → Discard current chain move and recover all the movement in the chain
  - Too many cells in the chain
    - Difficulty in search for legal positions for the last cells
    - → Set an upper bound to the size of the chain (10K cells)
      - Any chain exceeding the upper bound will trigger the discarding process
  - No overlaps with larger cells are allowed
    - Each overlapped cell must be no larger than the current moving cell
      - Otherwise, it is more difficult to find a legal position for those overlapping cells
- Visiting order of cells
  - Fixed cell order for each iteration → the cost improvement saturates quickly and fails to achieve further optimisation
  - Overlap reduction mode →MRHCs and large cells have a higher priority
  - Cost Minimisation → Cells far away from their optimal regions are granted high priority
    - Higher gain can be achieved
    - Different ordering across each DP iteration

8

Chain Move Algorithm – Search Region Computation (1)

- Optimal Region
  - step 1: get the nets that the examining cell belongs to (blue dashed lines)
  - step 2: get the net bounding boxes (BBs) ignoring the examining cell (red rectangles)
  - step 3: Sort the coordinates of the BBs for x and y axis
    - $\{x_l[1], x_l[2], x_r[1], x_r[2], x_l[3], x_r[3]\}$
    - $\{y_u[1], y_u[3], y_l[1], y_u[2], y_l[3], y_l[2]\}$
  - step 4: Compute the centre-of-mass for x and y (since the number of x and y coordinates is odd, the centre-of-mass will be a region)
    - $COM[x] = \{x_r[1], x_r[2]\}$
    - $COM[y] = \{y_l[1], y_u[2]\}$
  - Optimal region → the one with BB $(x_r[1], y_l[1]) - (x_r[2], y_u[2])$ (cyan highlighted region)

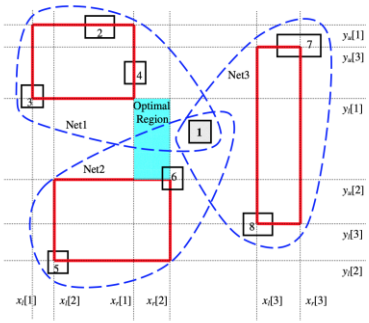


Fig. 2. Optimal Region.

9

Chain Move Algorithm – Search Region Computation (2)

- Search Region
  - Optimal region is often quite congested
- Extend the optimal region
  - Any bin intersecting with the search region will be considered for collection of candidate positions.

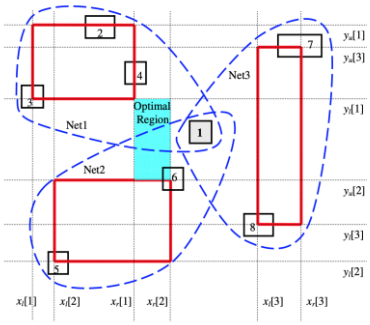


Fig. 2. Optimal Region.

10

Chain Move Algorithm – Cost Function

- Move Cost Function
  - $cost = \Delta WL \times (1 + \alpha \times c_d) + \beta \times c_{ov}$
    - ΔWL → wirelength cost
    - c_d → density cost
    - c_{ov} → overlap cost
  - If a cell is connected to some cells in the cellpool
    - such connections are ignored
  - Consider both area density and pin density
  - The overlap cost is defined as the total area of overlapped cells times the total number of pins divided by row height.

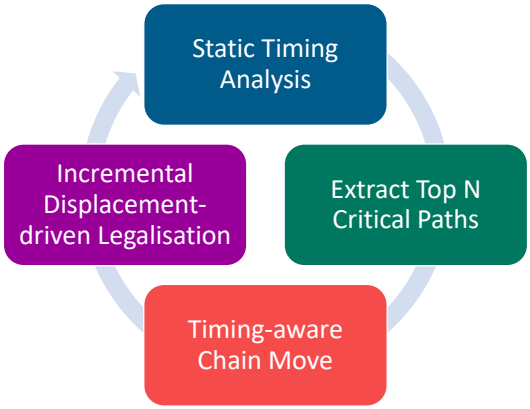
11

Timing-aware Chain Move

12

Timing-aware Detailed Placement

- A more important metric to improve during the Detailed Placement step is the circuit timing.
- ➔ Timing-aware Chain Move



13

Timing-aware Chain Move – Features

- Visiting order of cells
  - Consider the cells of the top N critical paths
  - The critical cells are sorted based on their distance from their centre of mass or by slack
    - starting from the most critical path to the lowest one
- Handling overlapping cells
  - Placing a cell in its new location may result in overlaps with other cells
  - In contrast to HPWL-driven Chain Move
    - The overlapping cell is inserted into the cellpool only if it exists in the examining critical paths.
    - Otherwise, the cell is handled by the Incremental Legalisation step following the Timing-aware Chain Move
- Visiting order of Cellpool cells
  - In HPWL-driven Cellpool was FIFO
    - Now, the cells in the Cellpool are sorted, like the examining cells, either by their centre of mass or by their slack

14

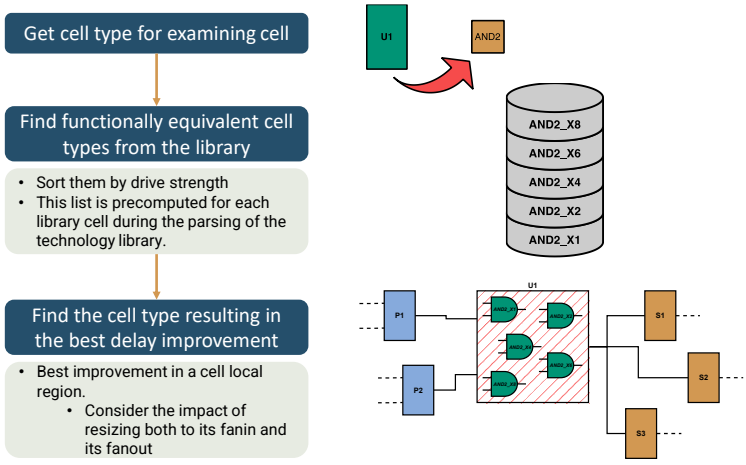
Timing-aware Chain Move – Actions (1)

- Movement:
  - For each cell:
    1. Computes the optimal region for the cell and expands it to the overlapping bins
    2. Extract candidate locations
    3. Attempt placing the cell to a 'valid' location achieving HPWL improvement
       - A location is valid if the constraints presented previously are met.

15

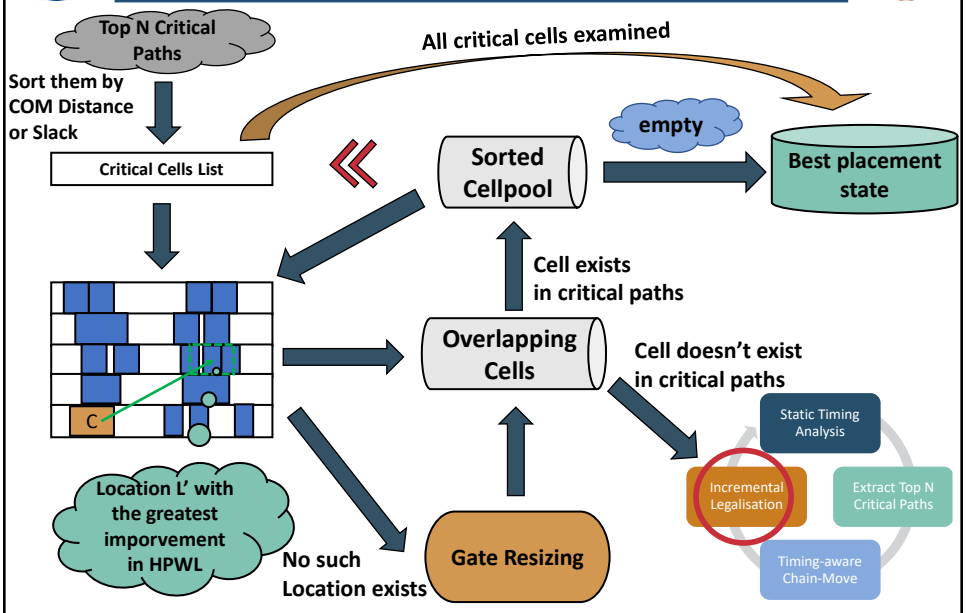
Timing-aware Chain Move – Actions (2)

- Gate Resizing
  - If no movement is achieved, then attempts gate resizing for the examining cell



16

Timing-aware Chain Move – Summary



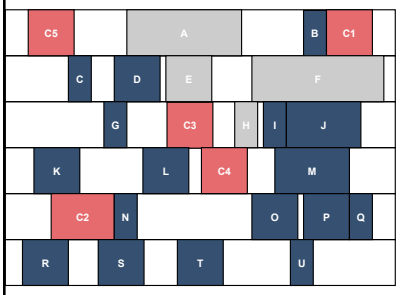
17

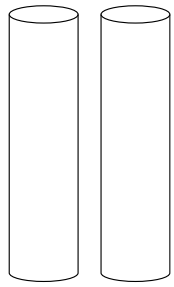
Timing-aware Chain Move

Example

18

Initial Placement





Density Bin Boundaries
Cell Optimal Region
Cell Center-of-Mass Region
C
Fixed cells
Best Placement State

cost1

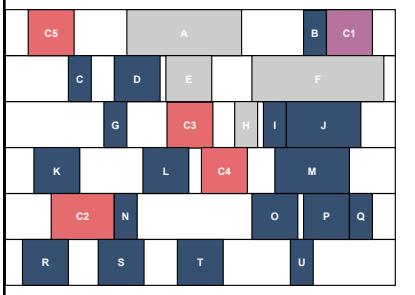
Algorithm 2 Timing-aware Chain Move Algorithm

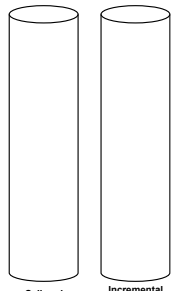
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

19

Place C1





Density Bin Boundaries
Cell Optimal Region
Cell Center-of-Mass Region
C
Fixed cells
Best Placement State

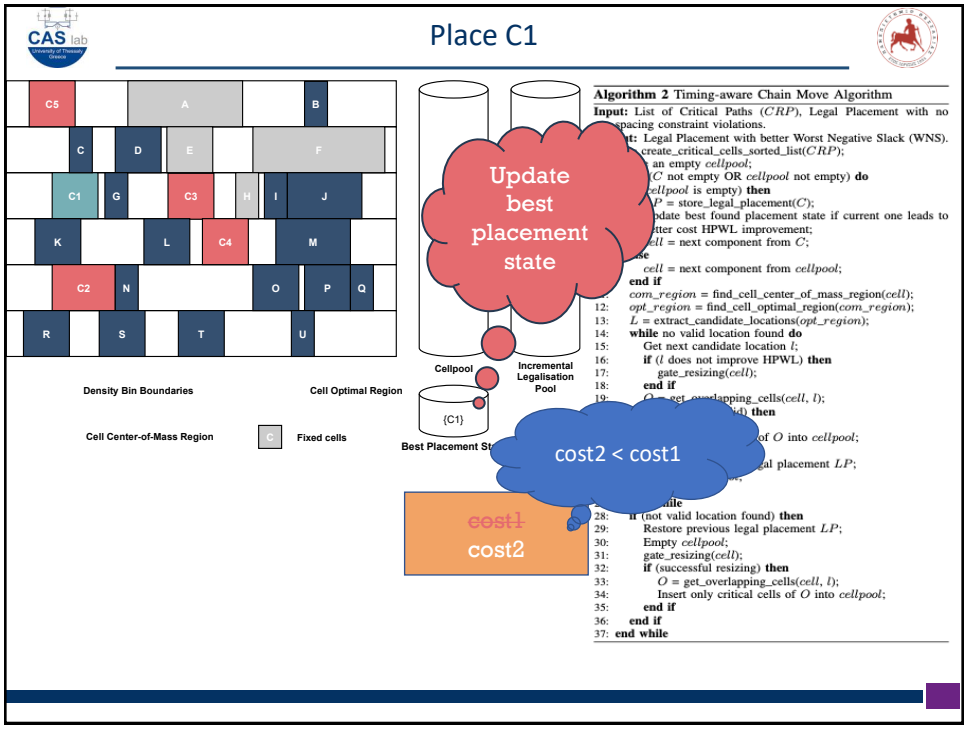
cost1

Algorithm 2 Timing-aware Chain Move Algorithm

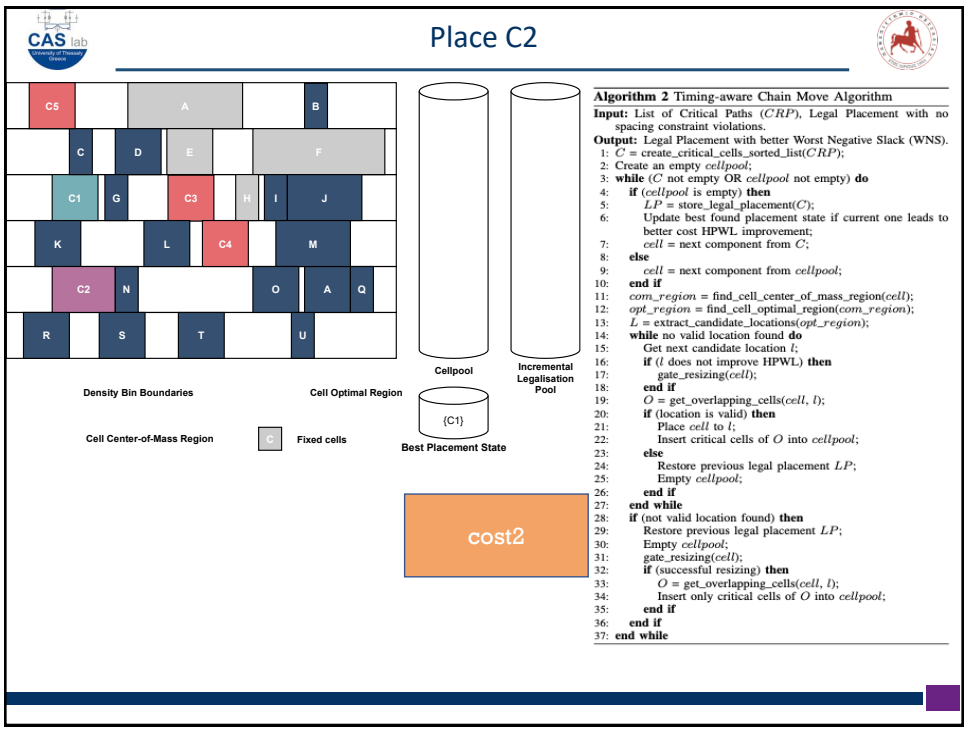
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

20

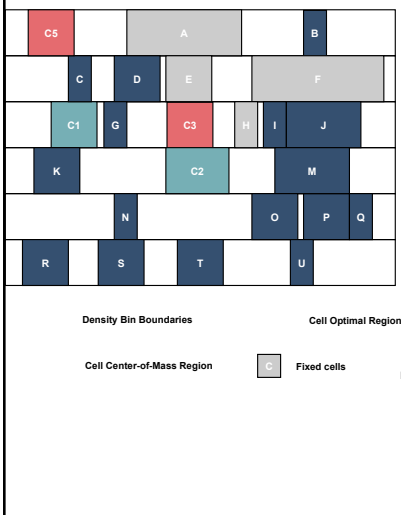

Place C1
Update best placement state
cost2 < cost1
cost1
cost2
Cellpool
Incremental Legalisation Pool
{C1}
Best Placement St
Density Bin Boundaries
Cell Optimal Region
Cell Center-of-Mass Region
Fixed cells
Algorithm 2 Timing-aware Chain Move Algorithm
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).
C = create_critical_cells_sorted_list(CRP);
Create an empty cellpool;
while (C not empty OR cellpool not empty) do
if (cellpool is empty) then
LP = store_legal_placement(C);
Update best found placement state if current one leads to better cost HPWL improvement;
cell = next component from C;
else
cell = next component from cellpool;
end if
com_region = find_cell_center_of_mass_region(cell);
opt_region = find_cell_optimal_region(com_region);
L = extract_candidate_locations(opt_region);
while no valid location found do
Get next candidate location l;
if (l does not improve HPWL) then
gate_resizing(cell);
end if
O = get_overlapping_cells(cell, l);
if (location is valid) then
Place cell to l;
Insert critical cells of O into cellpool;
else
Restore previous legal placement LP;
Empty cellpool;
gate_resizing(cell);
if (successful resizing) then
O = get_overlapping_cells(cell, l);
Insert only critical cells of O into cellpool;
end if
end if
end while
if (not valid location found) then
Restore previous legal placement LP;
Empty cellpool;
gate_resizing(cell);
if (successful resizing) then
O = get_overlapping_cells(cell, l);
Insert only critical cells of O into cellpool;
end if
end if
end while

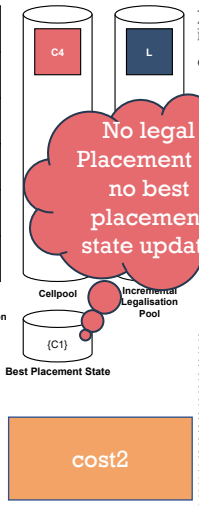
21


Place C2
cost2
Cellpool
Incremental Legalisation Pool
{C1}
Best Placement State
Density Bin Boundaries
Cell Optimal Region
Cell Center-of-Mass Region
Fixed cells
Algorithm 2 Timing-aware Chain Move Algorithm
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).
C = create_critical_cells_sorted_list(CRP);
Create an empty cellpool;
while (C not empty OR cellpool not empty) do
if (cellpool is empty) then
LP = store_legal_placement(C);
Update best found placement state if current one leads to better cost HPWL improvement;
cell = next component from C;
else
cell = next component from cellpool;
end if
com_region = find_cell_center_of_mass_region(cell);
opt_region = find_cell_optimal_region(com_region);
L = extract_candidate_locations(opt_region);
while no valid location found do
Get next candidate location l;
if (l does not improve HPWL) then
gate_resizing(cell);
end if
O = get_overlapping_cells(cell, l);
if (location is valid) then
Place cell to l;
Insert critical cells of O into cellpool;
else
Restore previous legal placement LP;
Empty cellpool;
end if
end while
if (not valid location found) then
Restore previous legal placement LP;
Empty cellpool;
gate_resizing(cell);
if (successful resizing) then
O = get_overlapping_cells(cell, l);
Insert only critical cells of O into cellpool;
end if
end if
end while

22

Place C2





cost2

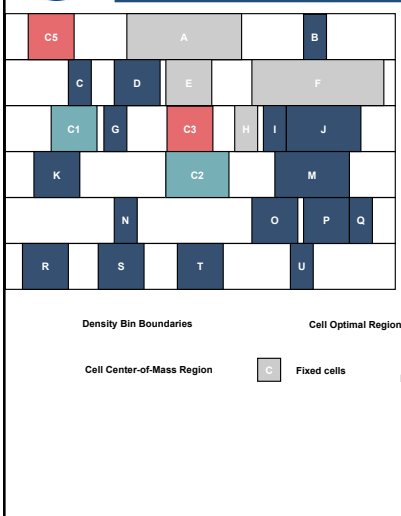
Algorithm 2 Timing-aware Chain Move Algorithm

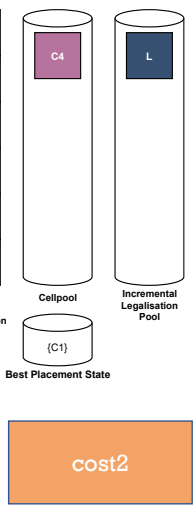
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1: C = create_critical_cells_sorted_list(CRP);
2: Create an empty cellpool;
   (C not empty OR cellpool not empty) do
   lpool is empty) then
   store_legal_placement(C);
   best found placement state if current one leads to
   st HPWL improvement;
   ext component from C;
   xt component from cellpool;
   ion = find_cell_center_of_mass_region(cell);
   region = find_cell_optimal_region(com_region);
   extract_candidate_locations(opt_region);
   hile no valid location found do
      Get next candidate location l;
      if (l does not improve HPWL) then
         gate_resizing(cell);
      end if
      O = get_overlapping_cells(cell, l);
      if (location is valid) then
         Place cell to l;
         Insert critical cells of O into cellpool;
      else
         Restore previous legal placement LP;
         Empty cellpool;
      end if
   end while
   if (not valid location found) then
      Restore previous legal placement LP;
      Empty cellpool;
      gate_resizing(cell);
      if (successful resizing) then
         O = get_overlapping_cells(cell, l);
         Insert only critical cells of O into cellpool;
      end if
   end if
end while
```

23

Place C4





cost2

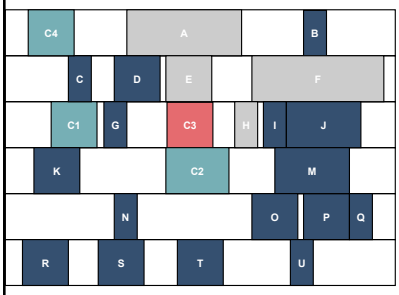
Algorithm 2 Timing-aware Chain Move Algorithm

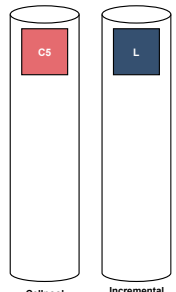
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1: C = create_critical_cells_sorted_list(CRP);
2: Create an empty cellpool;
3: while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:       LP = store_legal_placement(C);
6:       Update best found placement state if current one leads to
         better cost HPWL improvement;
7:       cell = next component from C;
8:    else
9:       cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:      Get next candidate location l;
16:      if (l does not improve HPWL) then
17:         gate_resizing(cell);
18:      end if
19:      O = get_overlapping_cells(cell, l);
20:      if (location is valid) then
21:         Place cell to l;
22:         Insert critical cells of O into cellpool;
23:      else
24:         Restore previous legal placement LP;
25:         Empty cellpool;
26:      end if
27:   end while
28:   if (not valid location found) then
29:      Restore previous legal placement LP;
30:      Empty cellpool;
31:      gate_resizing(cell);
32:      if (successful resizing) then
33:         O = get_overlapping_cells(cell, l);
34:         Insert only critical cells of O into cellpool;
35:      end if
36:   end if
37: end while
```

24

Place C4





cost2

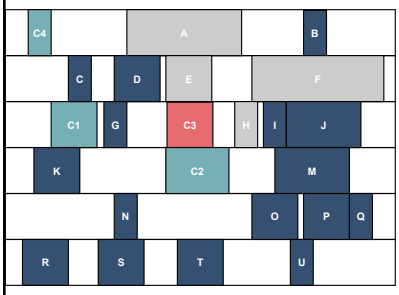
Algorithm 2 Timing-aware Chain Move Algorithm

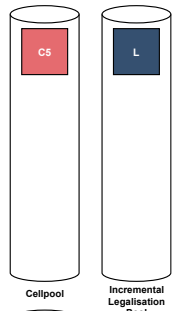
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

25

Resize C4 - Sucessful





cost2

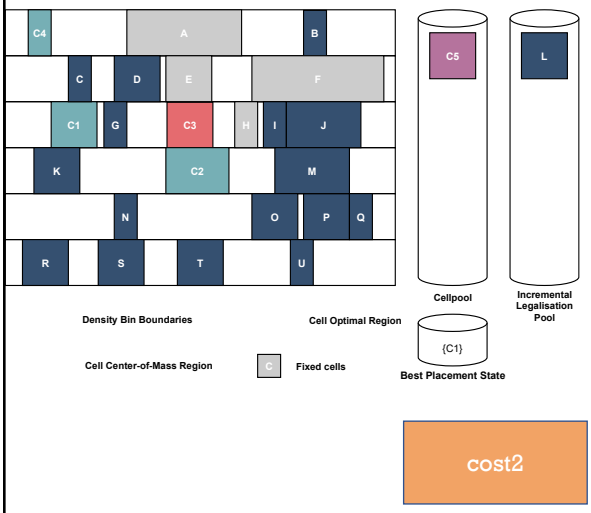
Algorithm 2 Timing-aware Chain Move Algorithm

Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

26

Place C5



Algorithm 2 Timing-aware Chain Move Algorithm

Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

27

Resize C5

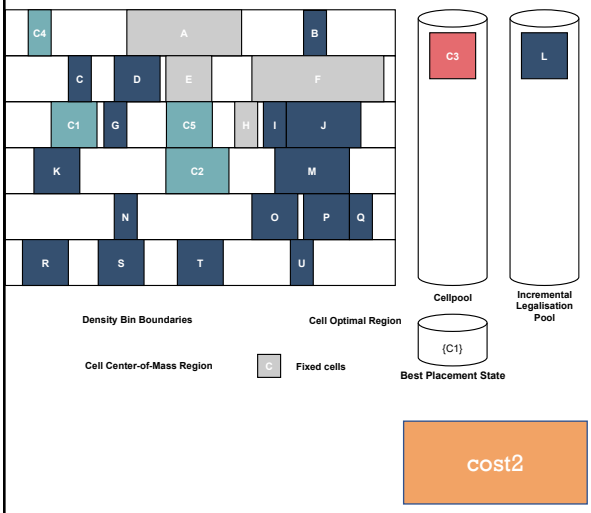
Algorithm 2 Timing-aware Chain Move Algorithm

Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

28

Resize C5 - Unsuccessful



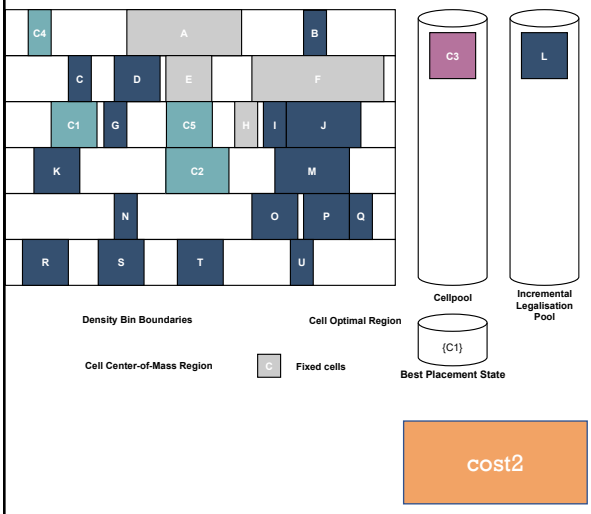
Algorithm 2 Timing-aware Chain Move Algorithm

Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

29

Place C3 – No valid location



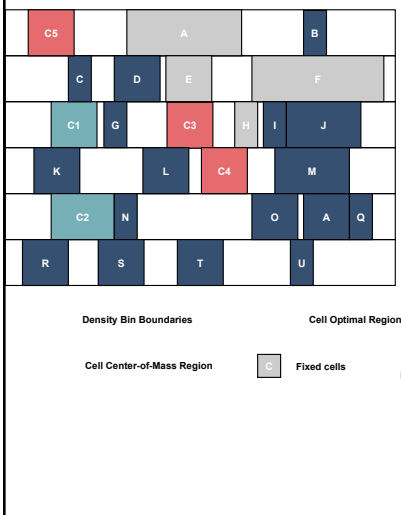
Algorithm 2 Timing-aware Chain Move Algorithm

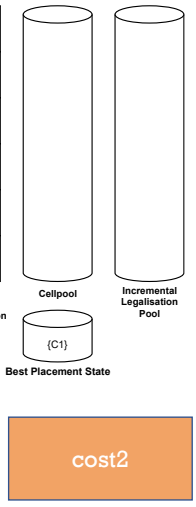
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

30

Place C3 → Revert to the last legal placement





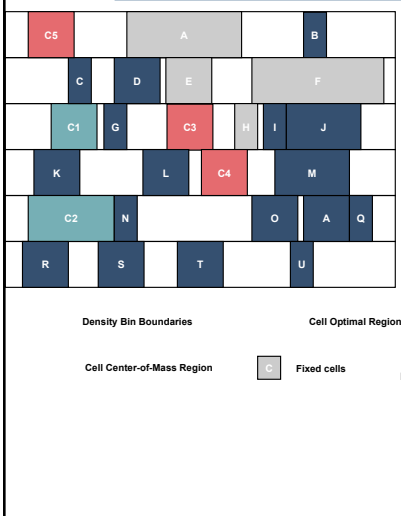
Algorithm 2 Timing-aware Chain Move Algorithm

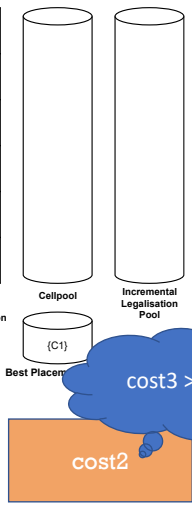
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
C = create_critical_cells_sorted_list(CRP);
Create an empty cellpool;
while (C not empty OR cellpool not empty) do
    if (cellpool is empty) then
        LP = store_legal_placement(C);
        Update best found placement state if current one leads to
        better cost HPWL improvement;
        cell = next component from C;
    else
        cell = next component from cellpool;
    end if
    com_region = find_cell_center_of_mass_region(cell);
    opt_region = find_cell_optimal_region(com_region);
    L = extract_candidate_locations(opt_region);
    while no valid location found do
        Get next candidate location l;
        if (l does not improve HPWL) then
            gate_resizing(cell);
        end if
        O = get_overlapping_cells(cell, l);
        if (location is valid) then
            Place cell to l;
            Insert critical cells of O into cellpool;
        else
            Restore previous legal placement LP;
            Empty cellpool;
        end if
    end while
    if (not valid location found) then
        Restore previous legal placement LP;
        Empty cellpool;
        gate_resizing(cell);
        if (successful resizing) then
            O = get_overlapping_cells(cell, l);
            Insert only critical cells of O into cellpool;
        end if
    end if
end while
```

31

Resize C2 - Successful





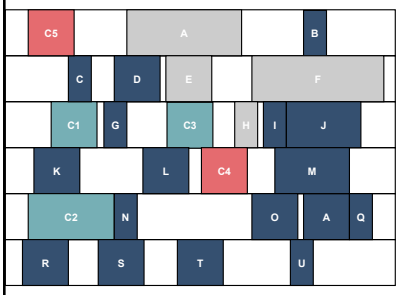
Algorithm 2 Timing-aware Chain Move Algorithm

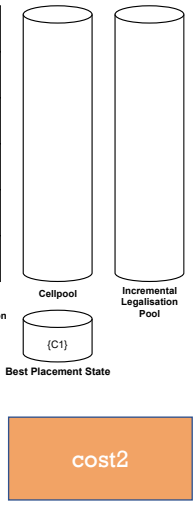
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
C = create_critical_cells_sorted_list(CRP);
Create an empty cellpool;
while (C not empty OR cellpool not empty) do
    if (cellpool is empty) then
        LP = store_legal_placement(C);
        Update best found placement state if current one leads to
        better cost HPWL improvement;
        cell = next component from C;
    else
        cell = next component from cellpool;
    end if
    com_region = find_cell_center_of_mass_region(cell);
    opt_region = find_cell_optimal_region(com_region);
    L = extract_candidate_locations(opt_region);
    while no valid location found do
        Get next candidate location l;
        if (l does not improve HPWL) then
            gate_resizing(cell);
        end if
        O = get_overlapping_cells(cell, l);
        if (location is valid) then
            Place cell to l;
            Insert critical cells of O into cellpool;
        else
            Restore previous legal placement LP;
            Empty cellpool;
        end if
    end while
    if (not valid location found) then
        Restore previous legal placement LP;
        Empty cellpool;
        gate_resizing(cell);
        if (successful resizing) then
            O = get_overlapping_cells(cell, l);
            Insert only critical cells of O into cellpool;
        end if
    end if
end while
```

32

Place C3





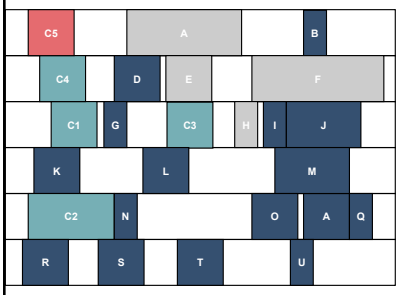
Algorithm 2 Timing-aware Chain Move Algorithm

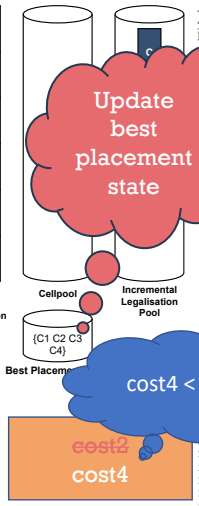
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
C = create_critical_cells_sorted_list(CRP);
Create an empty cellpool;
while (C not empty OR cellpool not empty) do
    if (cellpool is empty) then
        LP = store_legal_placement(C);
        Update best found placement state if current one leads to
        better cost HPWL improvement;
        cell = next component from C;
    else
        cell = next component from cellpool;
    end if
    com_region = find_cell_center_of_mass_region(cell);
    opt_region = find_cell_optimal_region(com_region);
    L = extract_candidate_locations(opt_region);
    while no valid location found do
        Get next candidate location l;
        if (l does not improve HPWL) then
            gate_resizing(cell);
        end if
        O = get_overlapping_cells(cell, l);
        if (location is valid) then
            Place cell to l;
            Insert critical cells of O into cellpool;
        else
            Restore previous legal placement LP;
            Empty cellpool;
        end if
    end while
    if (not valid location found) then
        Restore previous legal placement LP;
        Empty cellpool;
        gate_resizing(cell);
        if (successful resizing) then
            O = get_overlapping_cells(cell, l);
            Insert only critical cells of O into cellpool;
        end if
    end if
end while
```

33

Place C4 – Update Best Placement State





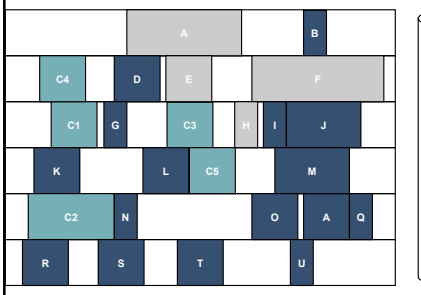
Algorithm 2 Timing-aware Chain Move Algorithm

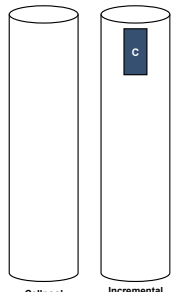
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
C = create_critical_cells_sorted_list(CRP);
Create an empty cellpool;
while (C not empty OR cellpool not empty) do
    if (cellpool is empty) then
        LP = store_legal_placement(C);
        Update best found placement state if current one leads to
        better cost HPWL improvement;
        cell = next component from C;
    else
        cell = next component from cellpool;
    end if
    com_region = find_cell_center_of_mass_region(cell);
    opt_region = find_cell_optimal_region(com_region);
    L = extract_candidate_locations(opt_region);
    while no valid location found do
        Get next candidate location l;
        if (l does not improve HPWL) then
            gate_resizing(cell);
        end if
        O = get_overlapping_cells(cell, l);
        if (location is valid) then
            Place cell to l;
            Insert critical cells of O into cellpool;
        else
            Restore previous legal placement LP;
            Empty cellpool;
        end if
    end while
    if (not valid location found) then
        Restore previous legal placement LP;
        Empty cellpool;
        gate_resizing(cell);
        if (successful resizing) then
            O = get_overlapping_cells(cell, l);
            Insert only critical cells of O into cellpool;
        end if
    end if
end while
```

34

Place C5





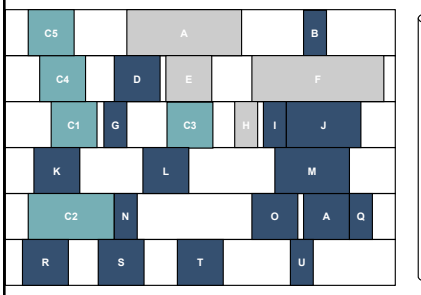
Algorithm 2 Timing-aware Chain Move Algorithm

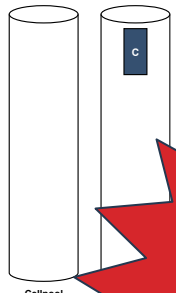
Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:      if (cellpool is empty) then
5:          LP = store_legal_placement(C);
6:          Update best found placement state if current one leads to
            better cost HPWL improvement;
7:          cell = next component from C;
8:      else
9:          cell = next component from cellpool;
10:     end if
11:     com_region = find_cell_center_of_mass_region(cell);
12:     opt_region = find_cell_optimal_region(com_region);
13:     L = extract_candidate_locations(opt_region);
14:     while no valid location found do
15:         Get next candidate location l;
16:         if (l does not improve HPWL) then
17:             gate_resizing(cell);
18:         end if
19:         O = get_overlapping_cells(cell, l);
            is valid) then
                l;
            cells of O into cellpool;
            us legal placement LP;
            ellpool;
            nd while
28:     if (not valid location found) then
29:         Restore previous legal placement LP;
30:         Empty cellpool;
31:         gate_resizing(cell);
32:         if (successful resizing) then
33:             O = get_overlapping_cells(cell, l);
34:             Insert only critical cells of O into cellpool;
35:         end if
36:     end if
37: end while
```

35

Revert to the Best Placement State





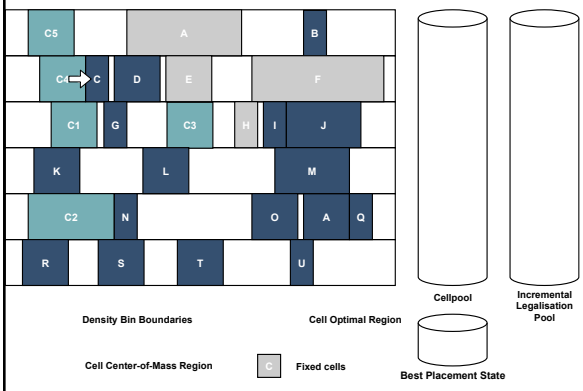
Algorithm 2 Timing-aware Chain Move Algorithm

Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:      if (cellpool is empty) then
5:          LP = store_legal_placement(C);
6:          Update best found placement state if current one leads to
            better cost HPWL improvement;
7:          cell = next component from C;
8:      else
9:          cell = next component from cellpool;
10:     end if
11:     com_region = find_cell_center_of_mass_region(cell);
12:     opt_region = find_cell_optimal_region(com_region);
13:     L = extract_candidate_locations(opt_region);
14:     while no valid location found do
15:         Get next candidate location l;
16:         if (l does not improve HPWL) then
17:             gate_resizing(cell);
18:         end if
19:         O = get_overlapping_cells(cell, l);
            is valid) then
                l;
            cells of O into cellpool;
            us legal placement LP;
            ellpool;
            nd while
28:     if (not valid location found) then
29:         Restore previous legal placement LP;
30:         Empty cellpool;
31:         gate_resizing(cell);
32:         if (successful resizing) then
33:             O = get_overlapping_cells(cell, l);
34:             Insert only critical cells of O into cellpool;
35:         end if
36:     end if
37: end while
```

36

Incremental Displacement-driven Legalisation



Algorithm 2 Timing-aware Chain Move Algorithm

Input: List of Critical Paths (CRP), Legal Placement with no spacing constraint violations.
Output: Legal Placement with better Worst Negative Slack (WNS).

```
1:  C = create_critical_cells_sorted_list(CRP);
2:  Create an empty cellpool;
3:  while (C not empty OR cellpool not empty) do
4:    if (cellpool is empty) then
5:      LP = store_legal_placement(C);
6:      Update best found placement state if current one leads to
        better cost HPWL improvement;
7:      cell = next component from C;
8:    else
9:      cell = next component from cellpool;
10:   end if
11:   com_region = find_cell_center_of_mass_region(cell);
12:   opt_region = find_cell_optimal_region(com_region);
13:   L = extract_candidate_locations(opt_region);
14:   while no valid location found do
15:     Get next candidate location l;
16:     if (l does not improve HPWL) then
17:       gate_resizing(cell);
18:     end if
19:     O = get_overlapping_cells(cell, l);
20:     if (location is valid) then
21:       Place cell to l;
22:       Insert critical cells of O into cellpool;
23:     else
24:       Restore previous legal placement LP;
25:       Empty cellpool;
26:     end if
27:   end while
28:   if (not valid location found) then
29:     Restore previous legal placement LP;
30:     Empty cellpool;
31:     gate_resizing(cell);
32:     if (successful resizing) then
33:       O = get_overlapping_cells(cell, l);
34:       Insert only critical cells of O into cellpool;
35:     end if
36:   end if
37: end while
```

Incremental Displacement-driven Legalisation

37