Chapter 3 – Chip Planning

Original Authors:
Andrew B. Kahng, Jens Lienig, Igor L. Markov, Jin Hu
Chapter 3 – Chip Planning

3.1 Introduction to Floorplanning
3.2 Optimization Goals in Floorplanning
3.3 Terminology
3.4 Floorplan Representations
   3.4.1 Floorplan to a Constraint-Graph Pair
   3.4.2 Floorplan to a Sequence Pair
   3.4.3 Sequence Pair to a Floorplan
3.5 Floorplanning Algorithms
   3.5.1 Floorplan Sizing
   3.5.2 Cluster Growth
   3.5.3 Simulated Annealing
   3.5.4 Integrated Floorplanning Algorithms
3.6 Pin Assignment
3.7 Power and Ground Routing
   3.7.1 Design of a Power-Ground Distribution Network
   3.7.2 Planar Routing
   3.7.3 Mesh Routing
3.1 Introduction

 ENTITY test is
 port a: in bit;
 end ENTITY test;

 DRC
 LVS
 ERC

 Circuit Design

 Physical Design

 Physical Verification and Signoff

 Fabrication

 Packaging and Testing

 Chip

 System Specification

 Architectural Design

 Functional Design and Logic Design

 Placement

 Clock Tree Synthesis

 Signal Routing

 Timing Closure

 Partitioning

 Chip Planning

 Fabrication

 Packaging and Testing

 Chip
3.1 Introduction

Chip Planning

Floorplan

I/O Pads

Supply Network

GND

VDD

Module a

Module b

Module c

Module d

Module e

Block a

Block b

Block c

Block d

Block e

Block Pins
Example
Given: Three blocks with the following potential widths and heights
Block A: \( w = 1, h = 4 \) or \( w = 4, h = 1 \) or \( w = 2, h = 2 \)
Block B: \( w = 1, h = 2 \) or \( w = 2, h = 1 \)
Block C: \( w = 1, h = 3 \) or \( w = 3, h = 1 \)
Task: Floorplan with minimum total area enclosed
3.1 Introduction

Example
Given: Three blocks with the following potential widths and heights

Block A: \( w = 1, h = 4 \) or \( w = 4, h = 1 \) or \( w = 2, h = 2 \)
Block B: \( w = 1, h = 2 \) or \( w = 2, h = 1 \)
Block C: \( w = 1, h = 3 \) or \( w = 3, h = 1 \)

Task: Floorplan with minimum total area enclosed
Example
Given: Three blocks with the following potential widths and heights
Block A: $w = 1, h = 4$ or $w = 4, h = 1$ or $w = 2, h = 2$
Block B: $w = 1, h = 2$ or $w = 2, h = 1$
Block C: $w = 1, h = 3$ or $w = 3, h = 1$

Task: Floorplan with minimum total area enclosed

Solution:
Aspect ratios
Block A with $w = 2, h = 2$; Block B with $w = 2, h = 1$; Block C with $w = 1, h = 3$

This floorplan has a global bounding box with minimum possible area (9 square units).
3.2 Optimization Goals in Floorplanning

- Area and shape of the global bounding box
  - Global bounding box of a floorplan is the minimum axis-aligned rectangle that contains all floorplan blocks.
  - Area of the global bounding box represents the area of the top-level floorplan
  - Minimizing the area involves finding \((x,y)\) locations, as well as shapes, of the individual blocks.

- Total wirelength
  - Long connections between blocks may increase signal propagation delays in the design.

- Combination of area \(\text{area}(F)\) and total wirelength \(L(F)\) of floorplan \(F\)
  - Minimize \(\alpha \cdot \text{area}(F) + (1 - \alpha) \cdot L(F)\)
    where the parameter \(0 \leq \alpha \leq 1\) gives the relative importance between \(\text{area}(F)\) and \(L(F)\)

- Signal delays
  - Static timing analysis is used to identify the interconnects that lie on critical paths.
3.3 Terminology

- A **rectangular dissection** is a division of the chip area into a set of *blocks* or non-overlapping rectangles.

- A **slicing floorplan** is a rectangular dissection
  - Obtained by repeatedly dividing each rectangle, starting with the entire chip area, into two smaller rectangles
  - Horizontal or vertical cut line.

- A **slicing tree** or **slicing floorplan tree** is a binary tree with *k* leaves and *k* − 1 internal nodes
  - Each leaf represents a block
  - Each internal node represents a horizontal or vertical cut line.
3.3 Terminology

Slicing floorplan and two possible corresponding slicing trees
3.3 Terminology

Polish expression

- Bottom up: $V \rightarrow \ast$ and $H \rightarrow +$
- Length $2n-1$ ($n =$ Number of leaves of the slicing tree)
3.3 Terminology

Non-slicing floorplans (wheels)
3.3 Terminology

Floorplan tree: Tree that represents a hierarchical floorplan

- **Horizontal division** (objects to the top and bottom)
- **Vertical division** (objects to the left and right)
- **Wheel** (4 objects cycled around a center object)
In a **vertical constraint graph (VCG)**, node weights represent the heights of the corresponding blocks.
- Two nodes \( v_i \) and \( v_j \), with corresponding blocks \( m_i \) and \( m_j \), are connected with a directed edge from \( v_i \) to \( v_j \) if \( m_i \) is below \( m_j \).

In a **horizontal constraint graph (HCG)**, node weights represent the widths of the corresponding blocks.
- Two nodes \( v_i \) and \( v_j \), with corresponding blocks \( m_i \) and \( m_j \), are connected with a directed edge from \( v_i \) to \( v_j \) if \( m_i \) is to the left of \( m_j \).

The longest path(s) in the VCG / HCG correspond(s) to the minimum vertical / horizontal floorplan span required to pack the blocks (floorplan height / width).

A **constraint-graph pair** is a floorplan representation that consists of two directed graphs – **vertical constraint graph** and **horizontal constraint graph** – which capture the relations between block positions.
3.3 Terminology

Constraint graphs

Horizontal Constraint Graph

Vertical Constraint Graph
3.3 Terminology

Sequence pair

- Two permutations represent geometric relations between every pair of blocks

- Example: \((ABDCE, CBAED)\)

- Horizontal and vertical relations between blocks \(A\) and \(B\):
  
  \[(\ldots A \ldots B \ldots, \ldots A \ldots B \ldots) \rightarrow A \text{ is left of } B\]
  
  \[(\ldots A \ldots B \ldots, \ldots B \ldots A \ldots) \rightarrow A \text{ is above } B\]
  
  \[(\ldots B \ldots A \ldots, \ldots A \ldots B \ldots) \rightarrow A \text{ is below } B\]
  
  \[(\ldots B \ldots A \ldots, \ldots B \ldots A \ldots) \rightarrow A \text{ is right of } B\]
3.4 Floorplan Representations

3.1 Introduction to Floorplanning

3.2 Optimization Goals in Floorplanning

3.3 Terminology

3.4 Floorplan Representations
  3.4.1 Floorplan to a Constraint-Graph Pair
  3.4.2 Floorplan to a Sequence Pair
  3.4.3 Sequence Pair to a Floorplan

3.5 Floorplanning Algorithms
  3.5.1 Floorplan Sizing
  3.5.2 Cluster Growth
  3.5.3 Simulated Annealing
  3.5.4 Integrated Floorplanning Algorithms

3.6 Pin Assignment

3.7 Power and Ground Routing
  3.7.1 Design of a Power-Ground Distribution Network
  3.7.2 Planar Routing
  3.7.3 Mesh Routing
• Create nodes for every block. In addition, create a source node and a sink one.
• Create nodes for every block. In addition, create a source node and a sink one.
• Add a directed edge \((A,B)\) if Block \(A\) is below/left of Block \(B\). (HCG)
3.4.1 Floorplan to a Constraint-Graph Pair

- Create nodes for every block. In addition, create a source node and a sink one.
- Add a directed edge \((A,B)\) if Block \(A\) is below/left of Block \(B\). (HCG)
- Remove the redundant edges that cannot be derived from other edges by transitivity.
### 3.4.2 Floorplan to a Sequence Pair

- Given two blocks $A$ and $B$ with
  - Locations: $A = (x_A, y_A)$ and $B = (x_B, y_B)$
  - Dimensions: $A = (w_A, h_A)$ and $B = (w_B, h_B)$
- If $x_A + w_A \leq x_B$ and $y_A + h_A \leq y_B$ or $y_B + h_B \leq y_A$, then $A$ is **left of** $B$
- If $y_A + h_A \leq y_B$ and $x_A + w_A \leq x_B$ or $x_B + w_B \leq x_A$, then $A$ is **below** $B$

![Diagram](image)

$S_+ : < acdbe >$

$S_- : < cdaeb >$
3.4.3 Sequence Pair to a Floorplan

- Start with the bottom left corner
- Define a *weighted sequence* as a sequence of blocks based on width
  - Each block $B$ has its own width $w(B)$

- Old (traditional) algorithm: find the longest path through edges ($O(n^2)$)
- New algorithm: find the *longest common subsequence* (LCS)
  - Given two weighted sequences $S_1$ and $S_2$, the $LCS(S_1, S_2)$ is the longest sequence found in both $S_1$ and $S_2$
  - The length of $LCS(S_1, S_2)$ is the summation of weights

- For block placement:
  - $LCS(S_+^R, S_+)$ returns the $x$-coordinates of all blocks
  - $LCS(S_+^R, S_)$ returns the $y$-coordinates of all blocks ($S_+^R$ is the reverse of $S_+$)
  - The length of $LCS(S_+^R, S_+)$ and $LCS(S_+^R, S_)$ is the width and height, respectively
Algorithm: Longest Common Subsequence (LCS)

Input: sequences $S_1$ and $S_2$, weights of $n$ blocks $weights$

Output: positions of each block $positions$, total span $L$

1. for ($i = 1$ to $n$)  // initialization
2. $block\_order[S_2[i]] = i$
3. $lengths[i] = 0$
4. for ($i = 1$ to $n$)
5. $block = S_1[i]$  // current block
6. $index = block\_order[block]$
7. $positions[block] = lengths[index]$  // compute block position
8. $t\_span = positions[block] + weights[block]$  // finds length of sequence
9. for ($j = index$ to $n$)  // from beginning to block
10. if ($t\_span > lengths[j]$) $lengths[j] = t\_span$
11. else break
12. $L = lengths[n]$  // total length is stored here
Example: \( S_1 = <acdbe>, S_2 = <cdaeb> \),
\[
\begin{align*}
\text{widths}[a b c d e] &= [8 4 4 4 4], \\
\text{heights}[a b c d e] &= [4 2 5 5 6]
\end{align*}
\]

Find x-coordinates – go by \( S_1 \)'s order:

Initial:
\[
\begin{align*}
\text{block\_order}[a b c d e] &= [3 5 1 2 4], \\
\text{lengths} &= [0 0 0 0 0]
\end{align*}
\]

Iteration 1 – block = \( a \), index = 3:
\[
\begin{align*}
\text{positions}[a] &= \text{lengths}[3] = 0, \\
t\_span &= 8, \\
\text{lengths} &= [0 0 8 8 8]
\end{align*}
\]

Iteration 2 – block = \( c \), index = 1:
\[
\begin{align*}
\text{positions}[c] &= \text{lengths}[1] = 0, \\
t\_span &= 4, \\
\text{lengths} &= [4 4 8 8 8]
\end{align*}
\]

Iteration 3 – block = \( d \), index = 2:
\[
\begin{align*}
\text{positions}[d] &= \text{lengths}[2] = 4, \\
t\_span &= 8, \\
\text{lengths} &= [4 8 8 8 8]
\end{align*}
\]

Iteration 4 – block = \( b \), index = 5:
\[
\begin{align*}
\text{positions}[b] &= \text{lengths}[5] = 8, \\
t\_span &= 12, \\
\text{lengths} &= [4 8 8 8 12]
\end{align*}
\]

Iteration 5 – block = \( e \), index = 4:
\[
\begin{align*}
\text{positions}[e] &= \text{lengths}[4] = 8, \\
t\_span &= 12, \\
\text{lengths} &= [4 8 8 12 12]
\end{align*}
\]

\[
\begin{align*}
\text{positions}[a b c d e] &= [0 8 0 4 8], \\
\text{total\_width} &= \text{lengths}[n = 5] = 12
\end{align*}
\]
### 3.4.3 Sequence Pair to a Floorplan

Example: \( S_1 = <acdbe>, S_2 = <cdaeb>, \)

\[
\text{widths}[a \ b \ c \ d \ e] = [8 \ 4 \ 4 \ 4], \quad \text{heights}[a \ b \ c \ d \ e] = [4 \ 2 \ 5 \ 5 \ 6]
\]

Find \( y \)-coordinates – go by \( S_1^R \)'s order:

Initial: \( \quad \text{block\_order}[a \ b \ c \ d \ e] = [3 \ 5 \ 1 \ 2 \ 4], \quad \text{lengths} = [0 \ 0 \ 0 \ 0 \ 0] \)

Iteration 1 – block = \( e \), index = 4:

\[
\text{positions}[e] = \text{lengths}[4] = 0, \quad t_\text{span} = 6, \quad \text{lengths} = [0 \ 0 \ 0 \ 6 \ 6]
\]

Iteration 2 – block = \( b \), index = 5:

\[
\text{positions}[b] = \text{lengths}[5] = 6, \quad t_\text{span} = 9, \quad \text{lengths} = [0 \ 0 \ 6 \ 9]
\]

Iteration 3 – block = \( d \), index = 2:

\[
\text{positions}[d] = \text{lengths}[2] = 0, \quad t_\text{span} = 5, \quad \text{lengths} = [0 \ 5 \ 5 \ 6 \ 9]
\]

Iteration 4 – block = \( c \), index = 1:

\[
\text{positions}[c] = \text{lengths}[1] = 0, \quad t_\text{span} = 5, \quad \text{lengths} = [5 \ 5 \ 5 \ 6 \ 9]
\]

Iteration 5 – block = \( a \), index = 3:

\[
\text{positions}[a] = \text{lengths}[3] = 5, \quad t_\text{span} = 9, \quad \text{lengths} = [5 \ 5 \ 9 \ 9 \ 9]
\]

\[
\text{positions}[a \ b \ c \ d \ e] = [5 \ 6 \ 0 \ 0 \ 0], \quad \text{total\_height} = \text{lengths}[n = 5] = 9
\]
3.5 Floorplanning Algorithms

3.1 Introduction to Floorplanning
3.2 Optimization Goals in Floorplanning
3.3 Terminology
3.4 Floorplan Representations
   3.4.1 Floorplan to a Constraint-Graph Pair
   3.4.2 Floorplan to a Sequence Pair
   3.4.3 Sequence Pair to a Floorplan
3.5 Floorplanning Algorithms
   3.5.1 Floorplan Sizing
   3.5.2 Cluster Growth
   3.5.3 Simulated Annealing
   3.5.4 Integrated Floorplanning Algorithms
3.6 Pin Assignment
3.7 Power and Ground Routing
   3.7.1 Design of a Power-Ground Distribution Network
   3.7.2 Planar Routing
   3.7.3 Mesh Routing
3.5 Floorplanning Algorithms

Common Goals

- To minimize the total length of interconnect, subject to an upper bound on the floorplan area

or

- To simultaneously optimize both wire length and area
3.5.1 Floorplan Sizing

Shape functions

Legal shapes

\[ h \geq w \geq A \]

Block with minimum width and height restrictions

3.5.1 Floorplan Sizing

Shape functions

Discrete \((h,w)\) values

Hard library block

3.5.1 Floorplan Sizing

Corner points
3.5.1 Floorplan Sizing

Algorithm

This algorithm finds the **minimum floorplan area** for a given slicing floorplan in polynomial time. For non-slicing floorplans, the problem is NP-hard.

- Construct the shape functions of all individual blocks
- Bottom up: Determine the shape function of the top-level floorplan from the shape functions of the individual blocks
- Top down: From the corner point that corresponds to the minimum top-level floorplan area, trace back to each block’s shape function to find that block’s dimensions and location.
Step 1: Construct the shape functions of the blocks

Block A:

- Height: 3
- Width: 5

Block B:

- Height: 4
- Width: 2
Step 1: Construct the shape functions of the blocks
3.5.1 Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks

Block A:

- 3 x 5
- 2 x 6

Block B:

- 4 x 2
- 2 x 4

Graph showing the shape functions with coordinates (w, h)
3.5.1 Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks

Block A:

Block B:

$A(w)$
3.5.1 Floorplan Sizing – Example

Step 1: Construct the shape functions of the blocks

Block A:

Block B:
3.5.1 Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (vertical)
Step 2: Determine the shape function of the top-level floorplan (vertical)
3.5.1 Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (vertical)
3.5.1 Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (vertical)
3.5.1 Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (vertical)
Step 2: Determine the shape function of the top-level floorplan (vertical)

Minimum top-level floorplan with vertical composition
3.5.1 Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (horizontal)

Minimum top-level floorplan with horizontal composition
3.5.1 Floorplan Sizing – Example

Step 3: Find the individual blocks’ dimensions and locations

(1) Minimum area floorplan: 5 x 5

Horizontal composition
3.5.1 Floorplan Sizing – Example

Step 3: Find the individual blocks’ dimensions and locations

(1) Minimum area floorplan: $5 \times 5$

(2) Derived block dimensions: $2 \times 4$ and $3 \times 5$

Horizontal composition
Step 3: Find the individual blocks’ dimensions and locations

(1) Minimum area floorplan: 5 x 5

(2) Derived block dimensions: 2 x 4 and 3 x 5

Horizontal composition
3.5.1 Floorplan Sizing – Example

Resulting slicing tree

V

B

A

5 x 5

B

A

2 x 4

3 x 5
3.5.2 Cluster Growth

- Iteratively add blocks to the cluster until all blocks are assigned
- Only the different orientations of the blocks instead of the shape / aspect ratio are taken into account
- Linear ordering to minimize total wirelength of connections between blocks
3.5.2 Cluster Growth – Linear Ordering

- **New nets** have no pins on any block from the partially-constructed ordering.
- **Terminating nets** have no other incident blocks that are unplaced.
- **Continuing nets** have at least one pin on a block from the partially-constructed ordering and at least one pin on an unordered block.
3.5.2 Cluster Growth – Linear Ordering

- Gain of each block $m$ is calculated:

$$Gain_m = \text{(Number of terminating nets of } m) - \text{(New nets of } m)$$

- The block with the maximum gain is selected to be placed next.

Diagram:
- $N_1$ to $A$
- $N_4$ to $B$
- $Gain_B = 1 - 1 = 0$
Given:

- Netlist with five blocks $A$, $B$, $C$, $D$, $E$ and six nets
  - $N_1 = \{A, B\}$
  - $N_2 = \{A, D\}$
  - $N_3 = \{A, C, E\}$
  - $N_4 = \{B, D\}$
  - $N_5 = \{C, D, E\}$
  - $N_6 = \{D, E\}$

- Initial block: $A$

Task: Linear ordering with minimum netlength
Continuing Nets
Gain Terminating Nets
New Nets
Block Iteration #

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Block</th>
<th>New Nets</th>
<th>Terminating Nets</th>
<th>Gain</th>
<th>Continuing Nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>$N_1, N_2, N_3$</td>
<td>--</td>
<td>-3</td>
<td>--</td>
</tr>
</tbody>
</table>

Initial block

$Gain_A = (\text{Number of terminating nets of } A) - (\text{New nets of } A)$
Continuing Nets
Gain
Terminating Nets
New Nets
Block
Iteration #

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Block</th>
<th>New Nets</th>
<th>Terminating Nets</th>
<th>Gain</th>
<th>Continuing Nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>$N_1, N_2, N_3$</td>
<td>--</td>
<td>-3</td>
<td>--</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>$N_4$</td>
<td>$N_1$</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>$N_5$</td>
<td>--</td>
<td>-1</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>$N_4, N_5, N_6$</td>
<td>$N_2$</td>
<td>-2</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>$N_5, N_6$</td>
<td>--</td>
<td>-2</td>
<td>$N_3$</td>
</tr>
</tbody>
</table>
Continuing Nets
Terminating Nets
Gain
New Nets
Block
Iteration 

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Block</th>
<th>New Nets</th>
<th>Terminating Nets</th>
<th>Gain</th>
<th>Continuing Nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>$N_1, N_2, N_3$</td>
<td>$N_1$</td>
<td>-3</td>
<td>--</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>$N_4$</td>
<td>$N_1$</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>$N_5$</td>
<td>$N_1$</td>
<td>-1</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>$N_4, N_5, N_6$</td>
<td>$N_2$</td>
<td>-2</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>$N_5, N_6$</td>
<td>$N_2, N_4$</td>
<td>0</td>
<td>$N_3$</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>$N_5$</td>
<td>$N_2, N_4$</td>
<td>-1</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>$N_5, N_6$</td>
<td>$N_2, N_4$</td>
<td>0</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>$N_5, N_6$</td>
<td>$N_2, N_4$</td>
<td>-2</td>
<td>$N_3$</td>
</tr>
</tbody>
</table>
## Table 1: Example Flow Graph with Iteration Information

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>Block</th>
<th>New Nets</th>
<th>Terminating Nets</th>
<th>Gain</th>
<th>Continuing Nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>$N_1, N_2, N_3$</td>
<td>--</td>
<td>-3</td>
<td>--</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>$N_4, N_5$</td>
<td>$N_1$</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>$N_4, N_5, N_6$</td>
<td>--</td>
<td>-1</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>$N_4, N_5, N_6$</td>
<td>--</td>
<td>-2</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>$N_4, N_5, N_6$</td>
<td>--</td>
<td>-2</td>
<td>$N_3$</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>$N_5$</td>
<td>--</td>
<td>-1</td>
<td>$N_3$</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>$N_5, N_6$</td>
<td>$N_2, N_4$</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>$N_5, N_6$</td>
<td>--</td>
<td>-2</td>
<td>$N_3$</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>--</td>
<td>--</td>
<td>0</td>
<td>$N_3, N_5$</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>--</td>
<td>$N_6$</td>
<td>1</td>
<td>$N_3, N_5$</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>--</td>
<td>$N_3, N_5$</td>
<td>2</td>
<td>--</td>
</tr>
</tbody>
</table>
3.5.2 Cluster Growth – Linear Ordering (Example)
3.5.2 Cluster Growth – Algorithm

**Input:** set of all blocks $M$, cost function $C$

**Output:** optimized floorplan $F$ based on $C$

$F = \emptyset$

$\text{order} = \text{LINEAR\_ORDERING}(M) \quad \text{// generate linear ordering}$

$\text{for } (i = 1 \text{ to } |\text{order}|) \quad \text{// find location and orientation}$

$\text{curr\_block} = \text{order}[i] \quad \text{// of curr\_block that causes}$

$\text{ADD\_TO\_FLOORPLAN}(F, \text{curr\_block}, C) \quad \text{// smallest increase based on}$

$\text{// C while obeying constraints}$
3.5.2 Cluster Growth

**Analysis**

- The objective is to minimize the total wirelength of connections blocks.

- Though this produces mediocre solutions, the algorithm is easy to implement and fast.

- Can be used to find the initial floorplan solutions for iterative algorithms such as *simulated annealing*.
3.5.3 Simulated Annealing

- Simulated Annealing (SA) algorithms are iterative in nature.

- Begins with an initial (arbitrary) solution and seeks to incrementally improve the objective function.

- During each iteration, a local neighborhood of the current solution is considered. A new candidate solution is formed by a small perturbation of the current solution.

- Unlike greedy algorithms, SA algorithms can accept candidate solutions with higher cost.
3.5.3 Simulated Annealing

- Solution states
- Cost
- Initial solution
- Local optimum
- Global optimum
3.5.3 Simulated Annealing

What is annealing?

- Definition (from material science): controlled cooling process of high-temperature materials to modify their properties.

- Cooling changes material structure from being highly randomized (chaotic) to being structured (stable).

- The way that atoms settle in low-temperature state is probabilistic in nature.

- Slower cooling has a higher probability of achieving a perfect lattice with minimum-energy
  - Cooling process occurs in steps
  - Atoms need enough time to try different structures
  - Sometimes, atoms may move across larger distances and create (intermediate) higher-energy states
  - Probability of the accepting higher-energy states decreases with temperature
3.5.3 Simulated Annealing

Simulated Annealing

- Generate an initial solution $S_{\text{init}}$, and evaluate its cost.
- Generate a new solution $S_{\text{new}}$ by performing a random walk.
- $S_{\text{new}}$ is accepted or rejected based on the temperature $T$
  - Higher $T$ means a higher probability to accept $S_{\text{new}}$ if $\text{COST}(S_{\text{new}}) > \text{COST}(S_{\text{init}})$
  - $T$ slowly decreases to form the final solution.

- Boltzmann acceptance criterion, where $r$ is a random number [0,1)

$$
\frac{\text{COST}(S_{\text{init}}) - \text{COST}(S_{\text{new}})}{e^{\frac{T}{T}}} > r
$$
3.5.3 Simulated Annealing

Simulated Annealing

- Generate an initial solution and evaluate its cost
- Generate a new solution by performing a random walk
- Solution is accepted or rejected based on a temperature parameter $T$
- Higher $T$ indicates higher probability to accept a solution with higher cost
- $T$ slowly decreases to form the finalized solution.

- Boltzmann acceptance criterion:
  \[
  e^{-\frac{\text{cost}(\text{curr}_\text{sol}) - \text{cost}(\text{next}_\text{sol})}{T}} > r
  \]
  \begin{align*}
  \text{curr}_\text{sol} & : \text{current solution} \\
  \text{next}_\text{sol} & : \text{new solution after perturbation} \\
  T & : \text{current temperature} \\
  r & : \text{random number between (0, 1) from normal distr.}
  \end{align*}
3.5.3 Simulated Annealing – Algorithm

![Progress of Simulated Annealing](image)
3.5.3 Simulated Annealing – Algorithm

**Input:** initial solution `init_sol`  
**Output:** optimized new solution `curr_sol`

\[
T = T_0 \quad \text{// initialization}
\]
\[
i = 0
\]
\[
curr_sol = init_sol
\]
\[
curr_cost = \text{COST}(curr_sol)
\]

**while** (\(T > T_{\text{min}}\))

**while** (stopping criterion is not met)

\[
i = i + 1
\]
\[
(a_i, b_i) = \text{SELECTPAIR}(curr_sol) \quad \text{// select two objects to perturb}
\]
\[
trial_sol = \text{TRY_MOVE}(a_i, b_i) \quad \text{// try small local change}
\]
\[
trial_cost = \text{COST}(trial_sol)
\]
\[
\Delta cost = trial_cost - curr_cost
\]

**if** (\(\Delta cost < 0\))

\[
curr_cost = trial_cost \quad \text{// if there is improvement,}
\]
\[
curr_sol = \text{MOVE}(a_i, b_i) \quad \text{// execute the move}
\]

**else**

\[
r = \text{RANDOM}(0,1) \quad \text{// random number \([0,1]\)}
\]
\[
\text{if } (r < e^{-\Delta cost/T}) \quad \text{// if it meets threshold,}
\]
\[
curr_cost = trial_cost \quad \text{// update the cost and}
\]
\[
curr_sol = \text{MOVE}(a_i, b_i) \quad \text{// execute the move}
\]

\[
T = \alpha \cdot T \quad \text{// } 0 < \alpha < 1, \text{ } T \text{ reduction}
\]
3.6 Pin Assignment

3.1 Introduction to Floorplanning
3.2 Optimization Goals in Floorplanning
3.3 Terminology
3.4 Floorplan Representations
   3.4.1 Floorplan to a Constraint-Graph Pair
   3.4.2 Floorplan to a Sequence Pair
   3.4.3 Sequence Pair to a Floorplan
3.5 Floorplanning Algorithms
   3.5.1 Floorplan Sizing
   3.5.2 Cluster Growth
   3.5.3 Simulated Annealing
   3.5.4 Integrated Floorplanning Algorithms

3.6 Pin Assignment

3.7 Power and Ground Routing
   3.7.1 Design of a Power-Ground Distribution Network
   3.7.2 Planar Routing
   3.7.3 Mesh Routing
During pin assignment, all nets (signals) are assigned to unique pin locations such that the overall design performance is optimized.
Given: Two sets of pins

(1) Determine the circles
(2) Determine the points
(2) Determine the points
(3) Determine initial mapping
(3) Determine initial mapping and (4) optimize the mapping (complete rotation)
(3) Determine initial mapping and (4) optimize the mapping (complete rotation)
(4) Best mapping (shortest Euclidean distance)
3.6 Pin Assignment – Example

(4) Best mapping

Final pin assignment
3.6 Pin Assignment

Pin assignment to an external block $B$

Pin assignment to two external blocks A and B

3.7 Power and Ground Routing

3.1 Introduction to Floorplanning
3.2 Optimization Goals in Floorplanning
3.3 Terminology
3.4 Floorplan Representations
  3.4.1 Floorplan to a Constraint-Graph Pair
  3.4.2 Floorplan to a Sequence Pair
  3.4.3 Sequence Pair to a Floorplan
3.5 Floorplanning Algorithms
  3.5.1 Floorplan Sizing
  3.5.2 Cluster Growth
  3.5.3 Simulated Annealing
  3.5.4 Integrated Floorplanning Algorithms
3.6 Pin Assignment

3.7 Power and Ground Routing
  3.7.1 Design of a Power-Ground Distribution Network
  3.7.2 Planar Routing
  3.7.3 Mesh Routing
3.7 Power and Ground Routing

Power-ground distribution for a chip floorplan

Power and ground rings per block or abutted blocks

Trunks connect rings to each other or to top-level power ring
3.7 Power and Ground Routing

Planar routing

Hamiltonian path
Planar routing

Step 1: Planarize the topology of the nets
  - As both power and ground nets must be routed on one layer, the design should be split using the Hamiltonian path

Step 2: Layer assignment
  - Net segments are assigned to appropriate routing layers

Step 3: Determining the widths of the net segments
  - A segment’s width is determined from the sum of the currents from all the cells to which it connects
3.7 Power and Ground Routing

Planar routing

Generating topology of the two supply nets

Adjusting widths of the segments with regard to their current loads
Mesh routing

Step 1: Creating a ring
  - A ring is constructed to surround the entire core area of the chip, and possibly individual blocks.

Step 2: Connecting I/O pads to the ring

Step 3: Creating a mesh
  - A power mesh consists of a set of stripes at defined pitches on two or more layers

Step 4: Creating Metal1 rails
  - Power mesh consists of a set of stripes at defined pitches on two or more layers

Step 5: Connecting the Metal1 rails to the mesh
3.7 Power and Ground Routing

Mesh routing

Power rail Connector

Ring Pad Mesh

© 2011 Springer Verlag
3.7 Power and Ground Routing

Mesh routing

M1-to-M4 connection

VDD rail

M4-to-M6 connection

GND rail

M6-to-M8 connection

VDD rail

© 2011 Springer Verlag
Summary of Chapter 3 – Objectives and Terminology

- **Traditional floorplanning**
  - Assumes area estimates for top-level circuit modules
  - Determines shapes and locations of circuit modules
  - Minimizes chip area and length of global interconnect

- **Additional aspects**
  - Assigning/placing I/O pads
  - Defining channels between blocks for routing and buffering
  - Design of power and ground networks
  - Estimation and optimization of chip timing and routing congestion

- **Fixed-outline floorplanning**
  - Chip size is fixed, focus on interconnect optimization
  - Can be applied to individual chip partitions (hierarchically)

- **Structure and types of floorplans**
  - Slicing versus non-slicing, the wheels
  - Hierarchical
  - Packed
  - Zero-deadspace
Summary of Chapter 3 – Data Structures for Floorplanning

- Slicing trees and Polish expressions
  - Evaluating a floorplan represented by a Polish expression

- Horizontal and vertical constraint graphs
  - A data structure to capture (non-slicing) floorplans
  - Longest paths determine floorplan dimensions

- Sequence pair
  - An array-based data structure that captures the information
  - contained in H+V constraint graphs
  - Makes constraint graphs unnecessary in practice

- Floorplan sizing
  - Shape-function arithmetic
  - An algorithm for slicing floorplans
Summary of Chapter 3 – Algorithms for Floorplanning

- **Cluster growth**
  - Simple, fast and intuitive
  - Not competitive in practice

- **Simulated annealing**
  - Stochastic optimization with hill-climbing
  - Many details required for high-quality implementation (e.g., temperature schedule)
  - Difficult to debug, fairly slow
  - Competitive in practice

- **Pin assignment**
  - Peripheral I/Os versus area-array I/Os
  - Given "ideal locations", project them onto perimeter and shift around, while preserving initial ordering

- **Power and ground routing**
  - Planar routing in channels between blocks
  - Can form rings around blocks to increase current supplied and to improve reliability
  - Mesh routing