Chapter 5
Placement

Formulation

Given: A set of cells (modules) of fixed dimensions and the interconnections between them.

Find: Position of each cell, such that
(i) no overlap (and enough routing space)
(ii) minimize routing area (congestion)
(iii) minimize total interconnections
Interconnection Cost

(a) Steiner Tree
Rectilinear Length = 14

(b) Steiner Tree with Trunk
Rectilinear Length = 15

(c) Minimum Spanning Tree
Rectilinear Length = 16

(d) Chain
Rectilinear Length = 17

(e) Complete Graph
Rectilinear Length = 42

Approximation: half perimeter of the bounding box

Measure of Congestion (Routing Area)

(a) Two Tracks Required. All connections routed

(b) Shorter Wire Length. Three Tracks Required.
A failure occurs if only two tracks are available
Placement Methods

I. Constructive methods.
   - Cluster growth algorithm
   - Force-directed method
   - Algorithm by Goto
   - Min-cut based method

II. Iterative improvement approaches
   - Pairwise exchange
   - Simulated annealing - Timberwolf
   - Genetic algorithm

III. Analytical methods

Constructive Method

Cluster Growth Algorithms

Select unplaced components and places them in slots.

(i) **SELECT** choose the unplaced component that is most strongly connected to all of the placed component (or: most strongly connected to any single placed component).

(ii) **PLACE** place the selected component at a slot such that a certain "cost" of the partial placement is minimized.
Goto’s Algorithm


Overview

اسلوب 1: Constructive Placement
اسلوب 2: Iterative Improvement

Cost Function

- For a given set of points to be interconnected, 1/2 of the perimeter of the minimal bounding box is an estimation of the total wire length.
- Cost of a placement:
  \[ \frac{1}{2} \sum P_i \]
  where \( P_i \) is the perimeter of the bounding box for net \( i \)
Median of a Set of Modules

M -- a module in the placement
For each net i connected to M:
Consider repositioning M at (x,y)
Define

\[ F(x, y) = \sum_{i=1}^{r} (f_i(x) + f_i(y)) \]

\[ f_i(x) = \begin{cases} 
    x_i^a - x & x < x_i^a \\
    0 & x_i^a \leq x \leq x_i^b \\
    x - x_i^b & x > x_i^b 
\end{cases} \]

\[ f_i(y) = \begin{cases} 
    y_i^a - y & y < y_i^a \\
    0 & y_i^a \leq y \leq y_i^b \\
    y - y_i^b & y > y_i^b 
\end{cases} \]

Why \( F(x,y) \)?

1/2 x perimeter of B' = 1/2 x perimeter of B + f(x) + f(y)

Find (x,y) to minimize

\[ F(x, y) = \sum_{i=1}^{r} (f_i(x) + f_i(y)) \]

\[ B_i(x) + f_i(x) = \sum \left\{ |x - x_i^a| + |x - x_i^b| + |x - y_i^a| + |x - y_i^b| \right\} \]

Can independently minimize

\[ \sum_i \left( |x - x_i^a| + |x - x_i^b| \right) \]

and

\[ \sum_i \left( |y - y_i^a| + |y - y_i^b| \right) \]
Median of a Set of Modules (Cont’d)

Find \( x \) such that
\[
|x - x_1^a| + |x - x_2^a| + |x - x_3^a| + \ldots \text{ is minimized}
\]

**Problem**
Let \( x_i \) be the set of points on position \( i \) (\( i = 1, 2, \ldots, n \)) along a line.

Find \( x \) such that \( \sum_{i=1}^{g} \alpha_i |x - x_i| \) is minimum.

**Fact**
Assume \( x_1 \leq x_2 \leq \ldots \leq x_n \), \( x \) is the “median” \( g \) of the given points set. i.e.
\[
\sum_{i=1}^{g} \alpha_i \leq \frac{N}{2} \leq \sum_{i=1}^{g} \alpha_i
\]

Find medium of the points \( x_i^a \)'s, \( x_i^b \)'s

Find median of the points \( y_i^a \)'s, \( y_i^b \)'s

\( \varepsilon \)-neighborhood of a module M

Let \( F_i(x) = \sum_i (|x - x_i^a| + |x - x_i^b|) \)
\[
F_i(x') \leq F_i(x') \leq F_i(x'') \leq \ldots
\]

This first \( \varepsilon \) \( x \)'s in this list is the \( \varepsilon \)-neighborhood of M
Iterative Placement Improvement Using $\varepsilon$-neighborhood

Let $\varepsilon = 3$

$\{B, C, D\} = 3$-neighborhood of $A$

Summary of Goto’s Method

I. Initial Placement:

Place modules one by one. For each module

- compute the $\varepsilon$-neighborhood $H$
- pick the best un-taken place from $H$

II. Improvement

$\lambda$-exchange based on $\varepsilon$-neighborhood

Jason Cong
**Force-Directed Method**


\[
K_{ij} = K_{ji}
\]

Map the graph to the layout surface

**Force-Directed Method**

(Cont’d)

Hooke’s Law

\[
\Delta S_{ij} = \begin{pmatrix} \Delta x_{ij} \\ \Delta y_{ij} \end{pmatrix} ; \quad \Delta S_{ij} = |\Delta x_{ij}| + |\Delta y_{ij}|
\]

\[
\overline{F}_{ij} = -K_{ij} \Delta S_{ij} = -K_{ij} \begin{pmatrix} \Delta x_{ij} \\ \Delta y_{ij} \end{pmatrix}
\]

* \( \overline{F}_{x_{ij}} = -K_{ij} \Delta x_{ij} \) (x-component of \( \overline{F}_{ij} \))
* \( \overline{F}_{y_{ij}} = -K_{ij} \Delta y_{ij} \) (y-component of \( \overline{F}_{ij} \))
Force-Directed Method
(Cont’d)

**Repulsion** (Among not adjacent cells)

\[
\bar{F}_{ij} = \frac{R}{\Delta S_{ij}} \begin{pmatrix} \Delta x_{ij} \\ \Delta y_{ij} \end{pmatrix}
\]

\( R = \text{Constant} \)

**Movable cell**

Sum of forces acting on cell equals 0

\[ \text{sum of horizontal forces} = 0 \]
\[ \text{sum of vertical forces} = 0 \]

---

Force-Directed Method
(Cont’d)

**Semi-movable cells**

sum of horizontal forces acting on cell equals 0, or
sum of vertical forces acting on cell equals 0

**Permanent cell**

Fixed location
\( x, y: \text{fixed} \)

Example of permanent cells

---

Jason Cong 17

Jason Cong 18
**Force-Directed Method (Cont’d)**

- **C₄, C₅**: movable
  - \(-k_{24}(x_i - x_j) - k_{23}(x_j - x_j) + R \frac{\Delta S_{24}}{\Delta S_{23}} (x_i - x_j) - k_{34}(y_i - y_j) = 0\)
  - \(-k_{34}(x_j - y_j) + R \frac{\Delta S_{34}}{\Delta S_{33}} (x_j - y_j) = 0\)
  - \(-k_{45}(x_i - y_i) + R \frac{\Delta S_{45}}{\Delta S_{44}} (x_i - y_i) = 0\)
  - \(-k_{54}(x_j - y_j) + R \frac{\Delta S_{54}}{\Delta S_{55}} (x_j - y_j) = 0\)
  - \(-k_{55}(x_j - y_j) = 0\)

**Solving the Non-linear Equations**

- Use an iterative method
- Use of a simple heuristic at iteration:

\[
\sum_{j \neq i}^N F_{x_{ij}} = 0 \quad \text{(for each i)}
\]

Let \(F_{x_i}\) be a single var. function w.r.t \(x_i\)

\[
\Rightarrow x_i^{\text{new}} = x_i - \frac{1}{2} \frac{F_{x_i}}{F_{x_i}} \quad \text{(heuristic)}
\]

Solving the equations for the y-dimension similarly
Force-Directed Method (Cont’d)

After solving the equation, we get components are not in the “right” position.

Minimum distortion using linear assignment

Placement solution

Force-Directed Method (Cont’d)

Construct a bipartite graph

Linear assignment or bipartite matching

Weight of edge \( \text{distance between } c_i \text{ and } l_i \) = distance between \( c_i \) and \( l_i \)

\( C_i \) is matched with \( l_i \)

\( c_i \) is placed at \( l_i \)
Min-Cut Based Placement

"A procedure for placement of standard-call VLSI circuits"
A.E. Dunlop, B.W. Kernighan, IEEE Trans. on CAD, vol CAD-4 No.1, Jan 1985

Min-cut with terminal propagation
area of each half \( \propto \) area of the cells

Min-Cut Based Placement (Cont’d)
This process continues until there are only a few cells in each group (\( \approx 6 \))

Assign cells in each group close together in the same row or nearly in adjacent rows

Each group has \( \leq 6 \) cells
Terminal Propagation

We should use the fact that s is in L₁!

Fictitious cell of net s

Assuming located at center

p will stay in R₁ for the rest of partitioning

When not to use p to bias partitioning

In this case, p should not be used to bias the solution in either direction

Do not use p

use p

Net s has cells in many groups:

Minimum cost rectilinear sterner tree

P₂ should be ignored! too close to the partition line

Jason Cong
Terminal Propagation (Cont’d)

- Terminal propagation reduces overall area by ~30%.
- Partitioning must be done breadth-first, not depth first.

Creating Rows

<table>
<thead>
<tr>
<th>Row 1</th>
<th>Row 2</th>
<th>Row 3</th>
<th>Row 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cells in C1</th>
<th>row1</th>
</tr>
</thead>
<tbody>
<tr>
<td>cells in C3</td>
<td>row1</td>
</tr>
<tr>
<td>cells in C2</td>
<td></td>
</tr>
</tbody>
</table>

Row 1: Row 2
Row 3: Row 4

Choose \( \alpha \) and \( \beta \) preferably to balance row length (During re-arrangement).

Partitioning of circuit into 32 groups. Each group is either assigned to a single row or divided into 2 rows.

Creating Rows

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,3</td>
<td>4,3</td>
<td>1,2</td>
</tr>
<tr>
<td>2</td>
<td>2,3</td>
<td>2,3</td>
</tr>
<tr>
<td>2,3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

A four-row standard cell design.
Experimental Results

CMOS Chip with 453 nets and 412 cells

- Manual Solution:
  track density=147; feedthroughs=184
- without terminal propagation: t.d.=313; f.t.=591
  (t.d. reduced to 235 by iterative interchanges)
- with terminal propagation: t.d.=186; f.t.=182
  (t.d. reduced to 152 by iterative interchanges)
CPU time=3230 secs VAX 11/780

- Iterative Interchange Refinement is helpful

Experimental Results (Cont’d)

- The program is in production use as part of an automatic placement system in AT&T Bell Lab.
- Solutions within 10% of the best hand layout

| Cells | Manual | | | Automatic | | | | Tech |
|-------|--------|--------|--------|--------|--------|--------|
|       | Tracks | Fthru | Tracks | Fthru | Tracks | Fthru |       |
| 290   | 91     | 0      | 93     | 51    | CMOS   |
| 412   | 147    | 184    | 152    | 182   | CMOS   |
| 1158  | 444    | 268    | 484    | 501   | CMOS   |
| 1198  | 581    | 410    | 384    | 441   | CMOS   |

Jason Cong
Remarks on Min-Cut Based Placement

Also implemented F-M partitioning method. Much faster but solutions appeared to be not as good as K-L

Use S-A to do partitioning. Much slower. if restricted to a reasonable CPU time, solutions are of similar quality of those by F-M method. Easy to implement

Seeking an elegant way to force some cells to be in particular positions

Investigate other algorithms for terminal propagation. Terminal propagation is the bottleneck of CPU time

Placement Methods

I. Constructive methods.
   - Cluster growth algorithm
   - Force-directed method
   - Algorithm by Goto
   - Min-cut based method

II. Iterative improvement approaches
   - Pairwise exchange
   - Simulated annealing - Timberwolf
   - Genetic algorithm

III. Analytical methods
Iterative Improvement

Pairwise interchanges

\( \lambda = 2 \) consider all possible swapping of pairs of modules systematically.

\[
\begin{align*}
\text{e.g.} & & \text{swap}(1,2), \text{swap}(1,3), \ldots, \text{swap}(1,n) \\
& & \text{swap}(2,3), \text{swap}(2,4), \ldots \ldots \\
& & \ldots \ldots \\
1 \text{ pass} & & \binom{n}{2}
\end{align*}
\]

\( \lambda = 3 \) swapping 3 modules systematically

\[ \binom{n}{3} \]

\( \lambda = 4 \) swapping 4 module systematically

\[ \binom{n}{4} \]

\( \lambda = k \)

expensive!

Simulated Annealing Based Placement


Timber wolf

Stage 1

- Modules are moved between different rows as well as within the same row
- Modules overlaps are allowed
- When the temperature is reduced below a certain value, stage 2 begins

Stage 2

- Remove overlaps
- Annealing process continues, but only interchanges adjacent modules within the same row
Solution Space

All possible arrangements of modules into rows possibly with overlaps

Neighboring Solutions

Three types of moves:

M1: Displace a module to a new location

M2: Interchange two modules

M3: Change the orientation of a module
Move Selection

- Timber wolf first try to select a move between M1 and M2
  
  \[ \text{Prob}(M1) = \frac{4}{5} \]  
  \[ \text{Prob}(M2) = \frac{1}{5} \]  

- If a move of type M1 is chosen (for certain module) and it is rejected, then a move of type M3 (for the same module) will be chosen with probability 1/10

Restriction on:
- How far a module can be displaced
- What pairs of modules can be interchanged

Move Restriction

Range Limiter

- At the beginning, R is very large, big enough to contain the whole chip
- Window size shrinks slowly as the temperature decreases. In fact, height and width of R are proportional to \( \log(T) \)
- Stage 2 begins when window size are so small that no inter-row modules interchanges are possible

Rectangular window R
Cost Function

\[ \Psi = C_1 + C_2 + C_3 \]

\[ C_i = \sum_i (\alpha_i w_i + \beta_i h_i) \]

\( \alpha_i, \beta_i \) are horizontal and vertical weights, respectively

\( \alpha_i = 1, \beta_i = 1 \Rightarrow 1/2 \cdot \text{perimeter of bounding box} \)

\( \bowtie \) Critical nets: Increase both \( \alpha_i \) and \( \beta_i \)

\( \bowtie \) Double metal technology: Over-the-cell routing is possible. Fewer feed through cells are needed.

\[ \therefore \text{vertical wirings are "cheaper" than horizontal wirings. use smaller vertical weights i.e. } \beta_i < \alpha_i \]
Annealing Schedule

- $T_k = r(k) \cdot T_{k-1}$, $k = 1, 2, 3, \ldots$
- $r(k)$ increases from 0.8 to max value 0.94 and then decreases to 0.1
- At each temperature, a total number of $K \cdot n$ attempts is made
  - $n$: number of modules
  - $K$: user specified constant

Experimental Results

- Compared with the CIPAR standard cell placement package developed by American Microsystems
- Test circuits: (800-2700 modules)
  - Total wirelength reduction 45% – 66%
- Running time:
  - 2700 modules example
    - 75 million moves 6.5 CPU hours on an IBM 3081 (Equivalent to 84 CPU hours on a VAX 780)
- Recently there are many efficient general annealing schedule that can substantially speed up the algorithm
SA-Based Placement without Cell Overlapping


Two Main features:

- No module overlaps
- Approximated cost calculation

The moves:

- 

- It is more expensive to update the cost function because each move involves movement of possibly many cells

Approximated Cost Computation

- Only update positions of the cells every U moves

- if \( \delta = \) error in the cost computation

\[
\delta \leq \alpha T \Rightarrow e^{-(\Delta C \pm \delta) / T} \approx e^{-\Delta C / T}
\]

Algorithm is not significantly disturbed

As long as \( U < R \frac{\alpha^2 T^2}{2mW^2} \)

We have the expected error \( < \alpha T \)

**Example** A typical circuit

\( w = 75 \mu m, R=20, m=1500, \) initial temp=10^6

For \( \alpha = 1 \ U=10^4 \) (at the beginning)
Approximated Cost Computation (Cont’d)

T: temperature
M: # of nets
R: # rows
w: Ave. cell width
U: # moves before updating s.t. $\delta < T$
Assume $\Delta n$: Ave # cell moves at each row

\[
\text{exp error of a cell position} = \sqrt{\Delta n} \times w
\]
\[
\text{exp error of a net cost} = \sqrt{2\Delta n} \times w
\]
\[
\text{exp error of total cost} = \sqrt{2\Delta n \times m} \times w
\]

\[
\Rightarrow \sqrt{2\frac{U}{R\times m}} \times w < T
\]
\[
\Rightarrow U < R \times \frac{T^2}{2mW^2}
\]

Experimental Results

- Compared with AT&T Bell Labs’ LTx2 (min-cut based)
- 8 problems: 1000-4000 standard cells
- Improvement of 20-40% in routing area
- 1.5-2 CPU hours on an Amdahl 5870 for a 3000 cell example (min-cut based algorithm takes about 0.5-1 hour)
- 5 times longer if we use exact cost computation. Also with about the same quality of solutions.
Placement Methods

I. Constructive methods.
   - Cluster growth algorithm
   - Force-directed method
   - Algorithm by Goto
   - Min-cut based method

II. Iterative improvement approaches
   - Pairwise exchange
   - Simulated annealing - Timberwolf
   - Genetic algorithm

III. Analytical methods

Analytical Placement

Gordian package:
Ref. 1: Gordian: VLSI Placement by Quadratic Programming and slicing Optimization
J. M. Kleinhans, G. Sigl, F. M. Johannes, K. J. Antreich

Ref. 2: Analytical Placement: A Linear or a Quadratic Objective Function?
G. Sigl, K. Doll, F. M. Johannes DAC'91 pp 427-423
Gordian: A Quadratic Placement Approach

- Global optimization: solves a sequence of quadratic programming problems
- Partitioning: enforces the non-overlap constraints

Overview of Gordian Package

GORDIAN with repartitioning procedure

Procedure Gordian

l:=1;
global-optimize(l);
while(∃M_i > k)
  for each ρ ∈ R(l)
    partition(ρ, ρ', ρ'');
  endfor
  l:=l+1;
  setup-constraints(l);
global-optimize(l);
repartition(l);
endwhile
final-placement(l);
endprocedure
Problem Definition

Squared wire length of net $v$

\[ L_v = \sum_{u \in M_v} \left( (x_{uv} - x_v)^2 + (y_{uv} - y_v)^2 \right) \]

\[ (x_{uv} = x_u + \xi_{uv}; y_{uv} = y_u + y_{vu}) \]

Cost Function

Overall objective

\[ \phi = \frac{1}{2} \sum_{v \in N} L_v w_v \]

\[ \phi (x,y) = X^T C X + d_x^T X + y^T C Y + d_y^T Y + \delta_{uv} \]

\[ \phi (x) = X^T C X + d^T X \]
Global Placement and Constraints

The center of gravity constraints
At level \( l \), chip is divided into \( q(\leq 2^l) \) regions
For region \( p \), the center coordinates: \((u_p, v_p)\)

Constraint:
\[
\sum_{u \in M_p} F_u x_u = u_p \sum_{u \in M_p} F_u
\]

\((M_p: \text{set of modules in region } p)\)

Matrix from for all regions
\[
A^{(l)}X = u^{(l)}, \quad a_{pu} = \begin{cases} 
F_i / \sum_{i \in M_p} F_i & \text{if } i \in M_p \\
0 & \text{otherwise}
\end{cases}
\]

Problem Formulation

\[
\begin{array}{c}
E_{(u_p, v_p)} \\
F
\end{array} + \begin{array}{c}
A \end{array} + \\
B_{(u_p, v_p)} \\
C
\]

Linearly constrained quadratic programming problem

\[
\text{LQP:} \min \{ \Phi(x) = X^T C X + d^T X \} \quad \text{s.t.} \ A^{(l)} X = u^{(l)} \}
\]
Solution Method

\[ A_{q \times m} = [D_{q \times q} B_{q \times (m-q)}] \]

\[ [D, B]_{m \times m} X_d = u \]

\[ X_d = -D^{-1}B X_i + D^{-1}u \]

\[ X = \begin{bmatrix} -D^{-1} B & D^{-1} u \end{bmatrix} \begin{bmatrix} X_d \\ X_i \end{bmatrix} = ZX + x_0 \]

Unconstrained quadratic programming problem

\[ UQP: \min_{x_i, q \in \mathbb{R}^{m-q}} \{ \psi(x_i) = X^T Z^T C Z X + C^T X_i \} \quad (C^T = CX_0 + d) \]

Solved by conjugate gradient method

Partitioning

• Recursive partitioning is needed to resolve module overlap in global placement

\[ M_p \rightarrow M_p', M_p'' \]

\[ x_i, x''_{u''} \quad u'^{\prime} \in M_p', u'' \in M_{p''} \]

\[ \alpha = \frac{\sum F_u}{\sum F_u} \approx 0.5 \]

cut value: \[ C_p(\alpha) = \sum_{v \in N_c} \psi_v \]

• Global placement problem will be solved again with two additional center-of-gravity constraints
Partitioning Improvement and Repartitioning

- Module exchange after each cut to improve cut size (terminal propagation using global placement positions)
- Repartitioning (to ‘undo’ the mistake made at the previous level):

  ```
  Procedure repartition(l)
  if overlap exists
    for each \( \rho \in R(l-1) \)
      merge-regions(\( \rho, \rho', \rho'' \));
      partition (\( \rho, \rho', \rho'' \));
    endfor
    setup-constraints(l);
    global-optimize(l);
  endif
  endprocedure
  ```

Summary of Gordian

- **Global Optimization**
  - minimization of wire length
  - module coordinates

- **Partitioning**
  - of the module set and dissection of the placement region
  - position constraints
  - Regions with \( \leq k \) modules

- **Final Placement**
  - adaption of style dependent constraints
  - module coordinates

Data flow in the placement procedure GORDIAN

**Complexity**
- space: \( O(m) \)
- time: \( Q(m^{1.5} \log^2 m) \)

**Final placement**
- standard cell
- macro-cell & SOG
Experimental Results

Comparison of Results for Standard Cell Blocks

<table>
<thead>
<tr>
<th>Circuit</th>
<th>GORDIAN</th>
<th>Min-Cut</th>
<th>Annealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>scb1</td>
<td>2.7</td>
<td>5.1</td>
<td>2.6</td>
</tr>
<tr>
<td>scb2</td>
<td>5.8</td>
<td>5.3</td>
<td>5.0</td>
</tr>
<tr>
<td>scb3</td>
<td>15.7</td>
<td>25.6</td>
<td>9.1</td>
</tr>
<tr>
<td>scb4</td>
<td>14.0</td>
<td>16.9</td>
<td>13.2</td>
</tr>
<tr>
<td>scb5</td>
<td>10.6</td>
<td>11.3</td>
<td>10.9</td>
</tr>
<tr>
<td>scb6</td>
<td>11.3</td>
<td>12.7</td>
<td>12.8</td>
</tr>
<tr>
<td>scb7</td>
<td>16.4</td>
<td>20.2</td>
<td>18.8</td>
</tr>
<tr>
<td>scb8</td>
<td>51.7</td>
<td>89.2</td>
<td>59.5</td>
</tr>
<tr>
<td>scb9</td>
<td>54.0</td>
<td>98.6</td>
<td>80.0</td>
</tr>
</tbody>
</table>

| CPU-time scb8 | 120s | 366s | 39851s |
| CPU-time scb9 | 135s | 440s | 34709s |
| ratio         | 1:3  | .:3  | .:300  |

Linear vs. Quadratic Objective Function

Differences between linear and quadratic objective function

\[
\begin{align*}
\phi_q &= l_\alpha^2 + l_\beta^2 + l_\gamma^2 = 2(l - l_\gamma)^2 + l_\gamma^2 \\
\phi_q &= -4(l - l_\gamma) + 2l_\gamma = 0 \Rightarrow l_\gamma = \frac{2}{3} l \\
\phi_q &= l_\alpha + l_\beta + l_\gamma
\end{align*}
\]

Jason Cong 59
Linear vs. Quadratic Objective Function (Cont’d)

- Quadratic objective function tends to make very long net shorter than linear objective function does, and let short nets become slightly longer.

Optimizing Linear Objective Function

Global Placement with linear objective function

\[ \phi_q = \sum_{v \in N} \sum_{u \in M_v} (x_{uv} - x_v)^2 \]  
Linear equation

\[ \phi = \sum_{v \in N} \sum_{u \in M_v} |x_{uv} - x_v| \]  
Linear programming

- use quadratic programming to minimize linear objective function

\[ \phi_l = \sum_{v \in N} \sum_{u \in M_{v}} (x_{uv} - x_v)^2 = \sum \sum (x_{uv} - x_v)^2 \]

\[ g_{uv} = |x_{uv} - x_v| \]

\[ g_v = \sum_{u \in M_v} |x_{uv} - x_v| \]
Effect of Linear Objective Function

(a) emphasizes on the nets with 2-3 terminals
(b) the force on modules close to the net node is reduced
(c) lower bound of $g_v$: $\omega_0$ (average module width)

Analytical Placement Results

Figure: Sum of wire lengths versus #pins
Analytical Placement Results

Table 1: Results (area in mm², CPU-time in seconds on VAX 8650)

<table>
<thead>
<tr>
<th>circuit</th>
<th>#modules</th>
<th>#nets</th>
<th>Gordian area</th>
<th>Gordian cpu</th>
<th>GordianL area</th>
<th>GordianL cpu</th>
<th>Vpr(CPLRT) area</th>
<th>Vpr(CPLRT) cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary1 struct</td>
<td>752</td>
<td>904</td>
<td>23.4</td>
<td>40</td>
<td>22.7</td>
<td>203</td>
<td>21.8</td>
<td>767</td>
</tr>
<tr>
<td>primary2</td>
<td>1888</td>
<td>1920</td>
<td>9.2</td>
<td>113</td>
<td>6.7</td>
<td>435</td>
<td>7.1</td>
<td>788</td>
</tr>
<tr>
<td>biomed</td>
<td>2907</td>
<td>3028</td>
<td>97.3</td>
<td>260</td>
<td>85.5</td>
<td>1180</td>
<td>90.1</td>
<td>2559</td>
</tr>
<tr>
<td>C1355</td>
<td>554</td>
<td>595</td>
<td>2.8</td>
<td>20</td>
<td>2.2</td>
<td>98</td>
<td>2.3</td>
<td>120</td>
</tr>
<tr>
<td>C5313</td>
<td>2330</td>
<td>2508</td>
<td>18.1</td>
<td>162</td>
<td>15.4</td>
<td>764</td>
<td>17.2</td>
<td>1440</td>
</tr>
<tr>
<td>S9234</td>
<td>5597</td>
<td>5844</td>
<td>46.6</td>
<td>250</td>
<td>38.0</td>
<td>1918</td>
<td>37.7</td>
<td>5575</td>
</tr>
<tr>
<td>20x20</td>
<td>6419</td>
<td>6464</td>
<td>16.5</td>
<td>839</td>
<td>15.1</td>
<td>3276</td>
<td>16.5</td>
<td>12791</td>
</tr>
<tr>
<td>AVQ</td>
<td>21046</td>
<td>21316</td>
<td>5067</td>
<td>86.9</td>
<td>22989</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Quadratic objective function
Linear objective function

(a) Global placement with 1 region
(b) Global placement with 4 region

(c) Final placements
Summary

- Commonly used placement techniques
  - Constructive (e.g. min-cut based placement)
  - Iterative (e.g. simulated annealing based -- Timberwolf)
  - Analytical (e.g. quadratic placement -- Gordian)

- Modern placement algorithms need to consider timing constraints and interconnect delays

Appendix: Conjugate Gradient Method:

Objective: solve $Ax = b$ using iterative method

Assume that $x_0 = 0$ (without loss of generality)

Residual vector: $r_i = b - Ax_i$

Krylov subspace $K^i(A; r_0) = \{r_0, Ar_0, \ldots, A^{i-1}r_0\}$

A-inner product: $(x, y)_A = (x, Ay)$

A-norm: $\|x\|_A^2 = (x, x)_A$

Basic idea: iterative

find $x_i \in K^i(A; r_0)$ such that $\|x_i - x\|_A$ is minimal

$\Rightarrow \quad x_i - x \perp_A K^i(A; r_0)$

$\Rightarrow \quad r_i \perp K^i(A; r_0)$
Conjugate Gradient: $Ax = b$

Nice property (proof by induction):

\[
\begin{align*}
\alpha_{j+1} r_{j+1} &= Ar_j - \beta_j r_j - \gamma_j r_{j-1} \\
\beta_j &= (r_j, Ar_j)/(r_j, r_j) \\
\gamma_j &= (r_{j-1}, Ar_j)/(r_{j-1}, r_{j-1}) \\
\alpha_{j+1} + \beta_j + \gamma_j &= 0
\end{align*}
\]

In matrix form:

\[
\begin{align*}
R_i &= (r_i, \ldots, r_{i+1}) \\
A R_i &= R_i T_i + \alpha_{j+1} r_i e_i^T
\end{align*}
\]

$T_i$ is an $i$ by $i$ tridiagonal matrix, $e_i$ is the vector with only one non-zero element at $i$-th position.

---

Conjugate Gradient: $Ax = b$

Find $x_i \in \mathcal{K}(A; r_0)$ such that $||x_i - x||_A$ is minimal:

\[
\begin{align*}
x_i \in \mathcal{K}(A; r_0) & \Rightarrow x_i = R_i y \\
||x_i - x||_A \text{ is minimal} & \Rightarrow R_i^T (Ax_i - b) = 0 \\
& \Rightarrow R_i^T A R_i y - R_i^T b = 0 \\
& \Rightarrow R_i^T T_i y = ||r_0||^2 e_i \quad (A R_i = R_i T_i + \alpha_{j+1} r_i e_i^T)
\end{align*}
\]

Notice $R_i^T R_i$ is a diagonal matrix, it’s trivial to solve for $y$, therefore we can obtain $x_i = R_i y$. 
Conjugate Gradient (standard alg)

\[ x_0 = 0; \quad r_0 = b - Ax_0; \quad p_1 = 0; \]
\[ \beta_1 = 0; \quad \rho_0 = (r_0, r_0) \]

for \( i = 0, 1, 2, \ldots \)

\[ p_i = r_i + \beta_{i-1}p_{i-1}; \]
\[ \alpha_i = \rho_i/(p_i, Ap_i); \]
\[ x_i = x_{i-1} + \alpha_i p_i; \]
\[ r_{i+1} = r_i - \alpha_i Ap_i; \]

if \( x_{i+1} \) accurate enough then quit;

\[ \rho_{i+1} = (r_{i+1}, r_{i+1}); \]
\[ \beta_i = \rho_{i+1} / \rho_i; \]

end;

Conjugate Gradient: Preconditioning

Convergence heavily depends on the condition number of \( A \)

Basic Idea of Conditioning:

- solve an equivalent system with better condition number
- find a pos-def. matrix \( K \approx A \), while \( Kw = r \) is cheap to compute

Derive the pre-conditioned algorithm:

- solve a new system \( K^{-1/2} A K^{-1/2} x^\Theta = K^{-1/2} b^\Theta \)
- apply the standard algorithm, replace
  \[ x \text{ by } x^\Theta, \quad b \text{ by } b^\Theta = K^{-1/2} b, \]
  \[ A \text{ by } K^{-1/2} A K^{-1/2}, \text{ etc.} \]
- get the recursive equation for the original system

Only a solver of type \( Kw = r \) is needed! No need for \( K^{-1/2} \).
Conjugate Gradient (Preconditioned)

\[ x_0 = 0; \quad r_0 = b - Ax_0; \quad p_0 = 0; \quad \beta_0 = 0; \]

Solve \( Kw_0 = r_0 \) for \( w_0 \); \( \rho_0 = (w_0, r_0) \)

For \( i = 0, 1, 2, \ldots \)

\[
\begin{align*}
\beta_i &= \rho_i / (\rho_i, Ap_i); \\
x_{i+1} &= x_i + \alpha_i p_i; \\
r_{i+1} &= r_i - \alpha_i Ap_i; \\
\end{align*}
\]

if \( x_{i+1} \) accurate enough then quit;

solve \( Kw_{i+1} = r_{i+1} \) for \( w_{i+1} \)

\[
\begin{align*}
\rho_{i+1} &= (r_{i+1}, w_{i+1}); \\
\beta_{i+1} &= \rho_{i+1} / \rho_i; \\
\end{align*}
\]

end;

Preconditioning Techniques for Sparse Matrix

Incomplete LU factorization (ILU):

\[ A = LU - P \]

\( L \) – sparse lower triangle; \( U \) – sparse upper triangle

Use \( K = LU \)

General method:

perform Gauss elimination as the complete LU factorization, drop some elements at pre-determined non-diagonal positions
Preconditioning Techniques for Sparse Matrix

General ILU:

given static non-zero pattern \( Z = \{(i, j) | i \neq j; 1 \leq i, j \leq n\} \)

for \( k=1, \ldots, n-1 \)

for \( i=k+1, n \) and if \( (i, k) \in Z \)

\[ a_{ik} = a_{ik}/a_{kk} \]

for \( j=k+1, \ldots, n \) and for \( (i, j) \in Z \)

\[ a_{ij} = a_{ij} - a_{ik} * a_{kj} \]

Matrix \( A \) will be modified to be the preconditioning matrix

ILU(0): pattern \( Z \) is the same as the original matrix \( A \).