Ch. 4 Placement, Assignment and Floorplanning
B. T. Preas and P. G. Karger

1 Introduction
2 Description of the Placement Problem
3 Abstract Models of Placement
4 Constructive Placement Algorithms
5 Iterative Placement Algorithms
6 Pin Assignment and Gate Assignment
7 Floorplanning
8 Applications of Placement Algorithms
9 Conclusion
Ch. 4 Placement, Assignment and Floorplanning
B. T. Preas and P. G. Karger

[Ref]

1. Ohtsuki, Ch. 2

2. Micheli, et al. ed., Ch. 5 "Automatic Layout of IC"
Introduction

• **Floorplanning.**

Given

- a set of modules with variable geometries, pins with variable positions, estimated delay and power consumption;

- constraints on the positions of some of the modules;

- constraints on the total estimated area and on the aspect ratio of the chip;

- constraints on the aspect ratios of the modules;

- constraints on the positions of the pins;

- a netlist specifying which pins have to be interconnected;
Introduction

- **Floorplanning (Cont.)**
  - constraints on total estimated power dissipation;
  - estimated delay per unit length of interconnection;
  - constraints on estimated delays at the chip level;

Determine geometries, locations, aspect ratios and pin positions for all the modules, so that all the constraints are satisfied, and a weighted sum of total estimated area, net length, delay, and power dissipation is minimized.
**Introduction**

- **Placement**

*Given*

- a set of modules with fixed geometric, fixed pins, delay and power consumption;

- constraints on the positions of some of the modules;

- constraints on the total area and on the aspect ratio of the chip;

- a netlist specifying which pins have to be interconnected;

- constraints on total power dissipation;

- estimated delay per unit length of interconnection;

- constraints on delays at the chip level;
Introduction

- **Placement Goal**

  *Determine locations for all the modules so that all the constraints are satisfied, and a weighted sum of total area, net length, delay, and power dissipation is minimized.*
Description of the Placement Problem

Layout = Placement + Routing

- **Placement** has three senses:
  - *Component placement*
  - *Resulting placement*
  - *Comprehensive placement* which includes component placement, floorplanning, pin assignment, and gate assignment.
Abstract Models of Placement

- **Cell** is the description of a circuit element.

  - Cell definition of the D-latch (and *component* is an instance of a cell).
Abstract Models of Placement

- **Interconnection Rules**

(a) Min. Steiner Tree $l = 10$

(b) Min. Spanning Tree $l = 11$
Abstract Models of Placement

(c) Min. Chain \( l = 12 \)

(d) Min. Source to Sink Connection \( l = 13 \)

(e) Complete Graph \( l = \frac{2}{4}(8+3+4+5+3+6) = 14.5 \)

\[
l = \frac{2}{\rho} \sum_{i=1}^{\rho\rho'(\rho-1)} (\text{length of wires})
\]

(f) Half Perimeter \( l = 9 \)
Measures of the Quality of a Placement

(a) Steiner Tree (14)
(b) Steiner Tree with Trunk (15)
(c) Minimum Spanning Tree (16)
(d) Chain (17)
(e) Complete Graph (42)
Abstract Models of Placement

§ Placement Goals

In the placement stage, locations of individual blocks are decided on a chip to satisfy a number of constraints in actual cases.

• The routing length of a signal net must be kept within tolerable bounds.
• The heat dissipation or power dissipation level had to be preserved.
• Signal cross-talk must be eliminated.
Abstract Models of Placement

§ Objective function

• Net Metrics
  - wire length: Manhattan, Euclidean distance
  - half perimeter
  - complete graph

• Congestion metrics
  - cut number
  - track density
  - routability
Abstract Models of Placement – Objective Functions

- (1) **Total wire length**

\[
T(P) = \sum_{ij} (w(i,j)d[P(i),P(j)])
\]

- (2) **Maximum Cut Line**

\[
X(P) = \text{Max } C_p(x_i)
\]

\[
Y(P) = \text{Max } C_p(y_i)
\]

/* # of signal nets crossing line \(x_i\) */

\[
T(P) = \sum_i C_p(x_i) + \sum_i C_p(y_i)
\]

/* # of signal nets crossing line \(y_i\) */
Abstract Models of Placement

- (3) **Maximum Density**

\[
D(P) = \text{Max } d_{pi}(e_i) = \text{Max } \frac{f_{pi}(e_i)}{C_{pi}(e_i)}
\]

\[
= \frac{\text{# of nets}}{\text{Capacity}}
\]
Abstract Models of Placement

Example.

Place all blocks (A, B, ..., G) on a 3x3 cell. \( S_1 = \{A_1, B_1\}, S_2 = \{A_2, B_2\}, S_3 = \{A_3, C_1\}, S_4 = \{C_2, D_1\}, S_5 = \{E_1, F_1, G_1\}, S_6 = \{H_1, I_1\} \)

\[
\begin{array}{ccc}
  A & B & E \\
  C & D & F \\
  H & I & G \\
\end{array}
\quad
\begin{array}{ccc}
  D & E & F \\
  A & C & G \\
  B & H & I \\
\end{array}
\]

\( T(P_1) = 7, \ X(P_1) = 4, \ Y(P_1) = 2, \ D(P_1) = 2 \) \quad \( T(P_2) = 8, \ X(P_2) = 2, \ Y(P_2) = 2, \ D(P_2) = 2 \)
Abstract Models of Placement

T(P3)=8(10), X(P3)=2(3), T(P4)=7(9), X(P4)=2(3), Y(P3)=3(3), D(P3)=2(1), Y(P4)=2(2), D(P4)=2(1)
Abstract Models of Placement

§ Layout surface (carrier)
- Geometric Model used in gate array, PCB layout.
- Topological Model used in standard cell and general cell IC design. (Fig. 4.5)

§ Minimum Length Algorithm
- (Problem) Given a set of blocks with signal nets defined on subsets of these blocks, and a set of cells, place all the blocks on the cells so that the total routing length over all signal nets is minimum.
- Block medium is defined as a position where the routing length associated with the block is minimum.
Abstract Models of Placement – Total Wire Length

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

where \( r \) is the number of nets connected to a block \( M \).

\[
\begin{align*}
  f_i(x) &= x_i^a - x \quad \text{if } x < x_i^a; \\
  f_i(x) &= 0 \quad \text{if } x_i^a \leq x \leq x_i^b; \\
  f_i(x) &= x - x_i^b \quad \text{if } x > x_i^b.
\end{align*}
\]

or \( f_i(x) = \frac{1}{2} \left( |x - x_i^a| + |x - x_i^b| - (x_i^b - x_i^a) \right) \)

\[
\begin{align*}
  f_i(y) &= y_i^a - y \quad \text{if } y < y_i^a; \\
  f_i(y) &= 0 \quad \text{if } y_i^a \leq y \leq y_i^b; \\
  f_i(y) &= y - y_i^b \quad \text{if } y > y_i^b.
\end{align*}
\]

\[
\min F(x,y) = \min \left( \sum_{i=1}^{r} (|x - x_i^a| + |x - x_i^b|) \right) + \sum_{i=1}^{r} (|y - y_i^a| + |y - y_i^b|) \right)
\]
Constructive Placement Algorithms

- Select seed blocks based on an evaluation function (IOC = I - O) then decide which cell the selected blocks will be placed.

- **Cluster Growth or Greedy Algorithm** (Bottom-up)

  /* initialization */
  seeds := findSeeds[unplacedComp]
  currentPlace := PLACE[seeds, currentPlace]

  while all components not placed do
    selectedComp := SELECT[currentPlace, unplacedComp]
    currentPlace := PLACE[selectedComp, currentPlace]
  end while
Constructive Placement Algorithms

• Partitioning-Based Placement (Top-down)

Partitioning Algorithm

- [Kernighan & Lin, 70] edge-cut algorithm
- [Schweikert & Kernighan, 72] net-cut algorithm
- [Fiduccia & Mattheyses, 82] linear net-cut algorithm
Partitioning

- Given a graph, \( G \), with \( n \) nodes with sizes (weights) \( w \):

\[
0 < w_i \leq p, \ i = 1, \ldots, n
\]

with costs on its edges, partition the nodes of \( G \) into \( k \) subsets, \( k > 0 \), no larger than a given maximum size, \( p \), so as to minimize the total cost of the edges cut.

- Define:\( C = (c_{ij}), \ i, j = 1, \ldots, n \)
as a weighted connectivity matrix describing the edges of \( G \).

- A \emph{k-way partition} of \( G \) is a set of non-empty, pairwise-disjoint subsets of \( G \), \( v_1, \ldots, v_k \), such that \( \bigcup_{i=1}^{k} v_i = G \)

- A partition is said to be admissible if \( |v_i| \leq p, i = 1, \ldots, k \)

- Problem: Find a minimal-cost permissible partition of \( G \)
**Exact Solutions**

- $n$ nodes, $k$ subsets of size $p$: $kp=n$
- $\binom{n}{p}$ ways to choose the first subset
- $\binom{n-p}{p}$ ways to choose the second, etc.
- $\frac{1}{k!}\binom{n}{p}\binom{n-p}{p}\cdots\binom{2p}{p}\binom{p}{p}$ ways total

- $n=40$, $p=10 > 10^{20}$
- In general, problems where $T_n \propto n^\beta$, $\beta > 2$
  are impractical for real circuits (>1,000,000 gates)
**n-Way Partitioning**

- Hard problem and no really good heuristics for $n>2$
  - **Direct Methods:** Start with seed node for each partition and assign nodes to each partition using some criterion (e.g. sum of weighted connections into partition)
  - **Group Migration Methods:** Start with (random) initial partition and migrate nodes among partitions via some heuristic
  - **Metric Allocation Methods:** uses metrics other than connection graph and then clusters nodes based on metric other than explicit connectivity.
  - **Stochastic Optimization Approaches:** Use a general-purpose stochastic approach like simulated annealing or genetic algorithms
- Usually apply two-way partitioning (Kernighan-Lin or Fiduccia-Mattheyses) recursively, or simulated annealing
Two-Way Partitioning
(Kernighan & Lin)

- Consider the set $S$ of $2n$ vertices, all of equal size for now, with an associated cost matrix $C = (c_{ij}), i, j = 1, \ldots, 2n$
- Assume $C$ is symmetric and $c_{ii} = 0 \forall i$
- We want to partition $S$ into two subsets $A$ and $B$, each with $n$ points, such that the external cost $T = \sum_{A \times B} C_{ab}$ is minimized
- Start with any arbitrary partition $[A, B]$ of $S$ and try to decrease the initial cost $T$ by a series of interchanges of subsets of $A$ and $B$
- When no further improvement is possible, the resulting partition $[A', B']$ is a local minimum and has a fairly high probability of being a global minimum with this scheme
Two-Way Partitioning
(Kernighan & Lin)

◆ For each $a \in A$:
  ▲ external cost $E_a = \sum_{y \in B} c_{ay}$ (same for $E_b$)
  ▲ internal cost $I_a = \sum_{x \in A} c_{ax}$ (same for $I_b$)

$D_z = E_z - I_z \forall z \in S$

◆ If $a \in A$ and $b \in B$ are interchanged, then the gain:

$g = D_a + D_b - 2c_{ab}$

◆ Proof: If $Z$ is the total cost of connections between $A$ and $B$, excluding $a$ and $b$, then:

$T_{a,b} = Z + E_a + E_b - c_{ab}$
$T_{b,a} = Z + I_a + I_b + c_{ab}$

$gain = T_{a,b} - T_{b,a} = D_a + D_b - 2c_{ab}$
Two-Way Partitioning  
(Kernighan & Lin)

(1) Compute all $D$ values in $S$
(2) Choose $a_i, b_i$ such that $g_i = D_{a_i} + D_{b_j} - 2c_{a_ib_j}$ is maximized
(3) Set $a_i$ and $b_i$ aside and call them $a'_i$ and $b'_i$
(4) Recalculate the D values for all the elements of $A - \{a_i\}, B - \{b_j\}$

\[
D'_x = D_x + 2c_{xa_i} - 2c_{xb_j}, x \in A - \{a_i\}
\]

\[
D'_y = D_y + 2c_{yb_j} - 2c_{ya_i}, y \in B - \{b_j\}
\]
Two-Way Partitioning (Kernighan & Lin)

- Repeat (2)-(4) on a new pair until all nodes exhausted

\[(a_1', b_1'), (a_2', b_2'), \ldots, (a_n', b_n')\]

- If sum to \(m > 0\), some gain, so repeat until sum to \(m=0\)

- What is the time and memory complexity of this algorithm?
Two-Way Partitioning
(Fiduccia & Mattheyses)

- Move one cell at a time from one side of the partition to the other in an attempt to minimize the cutset of the final partition
  - base cell -- cell to be moved
  - gain $g(i)$ -- no. of nets by which the cutset would decrease if cell $i$ were moved from partition $A$ to partition $B$ (may be negative)

- To prevent thrashing, once a cell is moved it is locked for an entire pass

- Claim is $O(n)$ time
Two-Way Partitioning  
(Fiduccia & Mattheyses)

◆ Steps:
  (1) Choose a cell
  (2) Move it
  (3) Update the g(i)’s of the neighbors
◆ O(n) neighbors, each cell recomputed each time neighbor moved, and must recompute for each pin (assume #pins=K.n, Amdahl 470 K~2.7)

\[ n_i^2 + n_i^2 + \cdots + n_i^2 = O \left( \text{# pins}^2 \right) / \text{pass} \]
Two-Way Partitioning  
(Fiduccia & Mattheyses)

- If $p(i) =$ no. of pins on cell $i$: $-p(i) < g_i < p(i)$
- Bin-sort cells on $g_i$

- Time required to maintain each bucket array $O(P)/\text{pass}$
Two-Way Partitioning  
(Fiduccia & Mattheyses)

◆ Move the Cell

(1) Find the first cell of highest gain that is not locked and such that moving it would not cause an imbalance
   ▲ Break tie by choosing the one that gives the best balance
(2) Choose this as the base cell. Remove it from the bucket list and place it on the LOCKED list. Update it to the other partition.

◆ Updating Cell Gains

Critical net

▲ Given a partition \((A|B)\), we define the distribution of \(n\) as an ordered pair of integers \((A(n),B(n))\), which represents the number of cells net \(n\) has in blocks \(A\) and \(B\) respectively (can be computed in \(O(P)\) time for all nets)
Two-Way Partitioning
(Fiduccia & Mattheyses)

- Net is critical if there exists a cell on it such that if it were moved it would change the net’s cut state (whether it is cut or not).
- Net is critical if $A(n) = 1$ or $B(n) = 1$ or $=0$
- Gain of cell depends only on its critical nets:
  - If a net is not critical, its cutstate cannot be affected by the move
  - A net which is not critical either before or after a move cannot influence the gains of its cells
- This is the basis of the linear-time claim
Two-Way Partitioning  
(Fiduccia & Mattheyses)

- Let $F$ be the *from* partition of cell $i$ and $T$ the *to* partition
- $g(i) = FS(i) - TE(i)$, where:
  - $FS(i) =$ no. of nets which have cell $i$ as their only $F$ cell
  - $TE(i) =$ no. of nets which contain $i$ and have an empty $T$ side
Two-Way Partitioning  
(Fiduccia & Mattheyses)

- Compute the initial gains of all unlocked cells:
  ```plaintext
  foreach(free cell i) {
    g(i) = 0;
    F = the “from” partition of cell i;
    T = the “to” partition of cell i;
    foreach(net n on cell i) {
      if(F(n) = 1) g(i)--;  
      if(T(n) = 0) g(i)++;
    }
  }
  
  Requires O(P) work to intialize
  ▲ net is critical before the move iff F(n)=1 or T(n)=0 or T(n) =1
  ▲ net is critical after the move iff T(n)=1 or F(n)=0 or F(n)=1
  ```
Two-Way Partitioning
(Fiduccia & Mattheyses)

- **Main loop:**
  - lock base cell;
  - foreach(net n on base cell) {
    - if(T(n) == 0) increment gains of all free cells on net n;
    - else if(T(n) == 1) decrement gains of the T cell on net n
      if it is free;
    - F(n)--;
    - T(n)++;
    - /* check critical nets after the move */
    - if(F(n)== 0) decrement gains of all free cells on net n;
    - else if(F(n) == 1) increment gain of the only F cell on
      net n if it is free;
  }

- **Time complexity O(nlog(n))?**
Constructive Placement Algorithms

**Min-Cut** Algorithm [Breuer 77]
- quadrature
- slice/bisection

**Partitioning based** placement [Lauther 79]
Initial Placement: Min-Cut

(a) 

(b) 

(c)
Mincut Initial Placement

- Represent the collection of cells by an arrangement on non-overlapping rectangles.
- Represent the collection of rectangles by a pair of mutually-dual, polar (directed, acyclic, one source one sink) graphs $G_X = (V_X, E_X)$ and $G_Y = (V_Y, E_Y)$
- $(e^i_X, e^i_Y)$ represents a rectangle with $L(e^i_X) \cdot L(e^i_Y) = A$
- To start $L(e^i_X) \cdot L(e^i_Y) = \sqrt{\sum_i a^i_X a^i_Y}$
- Repeat the process recursively, using K&L to determine elements of each partition
- Replace $L(e^i_X), L(e^i_Y)$ by actual $a^i_X, a^i_Y$ (true length)
- Compute longest path from source to sink (both graphs)
- Allocate location (float non-critical cells)
Mincut Initial Placement

- Apply rotation, mirror, and “squeeze”
- Iterate blocks to improve wiring congestion
Constructive Placement Algorithms

- **Minimum Density Algorithm** [Ohtsuki, Ch.2]
  - $C_{ij}^x$ ($C_{ij}^y$): number of signal nets that x-direction segment $e_{ij}^x$ (or y-dir segment $e_{ij}^y$) can accommodate.
  - $X_{ij}$ ($Y_{ij}$): number of signal nets assigned to segment $e_{ij}^x$ ($e_{ij}^y$).
  - $f(X_{ij}, r)$: cost of $e_{ij}^x$ when $X_{ij}$ signal nets are assigned to segment $e_{ij}^x$, where $r$ is a congestion parameter.
  - $X_{ij}/C_{ij}^x$ ($Y_{ij}/C_{ij}^y$): segment density.
Constructive Placement Algorithms

- Cost function must satisfy:

\[
\lim_{r \to 0} f(x_{ij}, r) = 0 \quad \text{if } x_{ij} \leq C_{ij}^x
\]

\[
\lim_{r \to \infty} f(x_{ij}, r) = \infty \quad \text{if } x_{ij} > C_{ij}^x.
\]

e.g. \( f(x_{ij}, r) = \left( \frac{x_{ij}}{C_{ij}} \right)^r \)

- Cost for the entire segment

\[
F_r = \left( \sum_{i=1}^{m} \sum_{j=1}^{n-1} f(x_{ij}, r) + \sum_{i=1}^{m-1} \sum_{j=1}^{n} f(y_{ij}, r) \right)^{1/r}
\]

\[
= \left( \sum_{i=1}^{m} \sum_{j=1}^{n-1} \left( \frac{x_{ij}}{C_{ij}} \right)^r + \sum_{i=1}^{m-1} \sum_{j=1}^{n} \left( \frac{y_{ij}}{C_{ij}} \right)^r \right)^{1/r}
\]
Constructive Placement Algorithms

For example, if we let \( r=1 \) and \( C_{ij}^x = C_{ij}^y = C \)

\[
F_1 = \frac{1}{C} \left( \sum_{i=1}^{m} \sum_{j=1}^{n-1} X_{ij} + \sum_{i=1}^{m} \sum_{j=1}^{n} Y_{ij} \right)
\]

=> Total wire length

If we let \( r \rightarrow \infty \), then

\[
F_{\infty} = \text{MAX} \left( \frac{X_{ij}}{C_{ij}^x}, \frac{Y_{ij}}{C_{ij}^y} \right)
\]

• Calculation for Density

\( L_{AB} \): number of routing paths connecting blocks A and B.

\[
L_{AB} = 1 \quad \text{if } x_A = x_B \text{ or } y_A = y_B
\]

\[
L_{AB} = |x_B - x_A| + |y_B - y_A| \quad \text{if } x_A \neq x_B \text{ and } y_A \neq y_B
\]
Constructive Placement Algorithms

\( l(e_{ij}^x) \): number of signal nets passing through segment \( e_{ij}^x \).

1. If \( x_A = x_B \) then \( l(e_{ij}^x) = 0 \)
2. If \( x_A \neq x_B \) and \( y_A = y_B \) then
   \( l(e_{ij}^x) = 1 \) if \( x_A \leq i \leq x_B - 1 \) and \( j = y_A \);
   \( l(e_{ij}^x) = 0 \) otherwise.

3. If \( x_A \neq x_B \) and \( y_A = y_B \) then
   \( l(e_{ij}^x) = x_B - i \) if \( x_A \leq i \leq x_B - 1 \) and \( j = y_A \);
   \( l(e_{ij}^x) = i - x_A + 1 \) if \( x_A \leq i \leq x_B - 1 \) and \( j = y_B \);
   \( l(e_{ij}^x) = 1 \) if \( x_A \leq i \leq x_B - 1 \) and \( y_A < j < y_B \).
   \( l(e_{ij}^x) = 0 \) otherwise.
Constructive Placement Algorithms

Given, \( A=(X_A,Y_A)=(1,3) \), \( B=(X_B,Y_B)=(4,1) \)
for \( m=3 \), \( n=4 \), \( l_{AB} = |4-1| + |1-3| = 5 \)

\( l(e_{13}^x) = 4-1 = 3 \)
\( l(e_{11}^x) = 1-1+1 = 1 \)
\( l(e_{23}^x) = 4-2 = 2 \)
\( l(e_{21}^x) = 2-1+1 = 2 \)
\( l(e_{33}^x) = 4-3 = 1 \)
\( l(e_{31}^x) = 3-1+1 = 3 \)
Constructive Placement Algorithms

- **Global Placement**
  - **Quadratic Assignment**

  Given cost matrix \( C = [c_{ij}] \) and distance matrix \( D = [d_{kl}] \), try to minimize
  \[
  \sum_{ij} c_{ij} d_{p_i p_j}
  \]
  over all permutations \( P \) of the component positions.

- **Convex Function Optimization**

  1. force analogy
  2. electrical circuit
  3. eigenvalue.
Constructive Placement Algorithms

- Branch and Bound Placement

\[
\text{bestScore} := \text{scoreOf}[\text{knownPlace}]
\]
\[
\text{remainSolSet} := \text{universe}
\]
\[
\text{incumbentSolSet} := \text{knownPlace}
\]

while remainSolSet not empty do

activeSet := chooseASet[remainSolSet]

remainSolSet := remainSolSet - activeSet

activeSubSet := partition[activeSet]

/* branch step */

for each element in activeSubSet do

Lb := Lbof[element]

if Lb < bestScore then /* bound step */
Constructive Placement Algorithms

begin
    if element is fathomed then
        begin
            bestScore := Lb
            incumbentSol := element
            incumbentSol := element
        end
    else
        remainSolSet := remainSolSet + element
    end
end for
end while
Iterative Placement Algorithms

- Iterative Improvement Algorithms
  - PI: Pairwise Interchange
  - NI: Neighborhood Interchange
  - FDI: Force-Directed Interchange
  - FDR: Force-Directed Relaxation
  - GFDR: General Force-Directed Relaxation
  - FDPR: Force-Directed Pairwise Relaxation
Iterative Placement Algorithms

• Generic Iterative Placement Algorithm

    currentScore := SCORE[currentPlace]
    until stopping criterion satisfied do
        selectedComp := SELECT[currentPlace]
        trialPlace := MOVE[selectedComp, currentPlace]
        trialScore := SCORE[trialPlace]
        if trialScore < currentScore then
            currentScore := trialScore
            currentPlace := trialPlace
        else
            currentPlace := MOVE[selectedComp, trialPlace]
        end
**Iterative Placement Algorithms**

- **Interchange Algorithms**
  - PI: Pairwise Interchange
  - NI: Neighborhood Interchange
Pairwise Interchange

while( cost > TARGET_COST) {
    select two modules;
    interchange them;
    compute new cost;
    if( new cost > old cost) replace them;
}

- n modules, n(n-1)/2 interchanges/cycle.
- Neighborhood Interchange
  ▲ Exchange only in a neighborhood

![Diagram showing cost decrease over iterations towards a target cost]
Iterative Placement Algorithms

- **Force-Directed Methods** $F_{ij} = -C_{ij} d_{ij}$

  FDI: Force-Directed Interchange,

  FDR: Force-Directed Relaxation,

  FDPR: Force-Directed Pairwise Relaxation.
Force-Directed Relaxation

- Sort all N modules into a stack S such that top module has max. no. of nets connected to it.

while(S is not empty) {
    m=pop(S);
    compute forces on m due to blocks to which it is connected;
determine target location, t, which minimizes all forces;
    if(not occupied) move m to target and lock it;
    else if(occupied but not locked) {
        replace t with m and lock m;
        push(t, S);
    }
    else if(occupied and locked) {
        check local area for a free cell;
        if(none found) leave m where it was;
    }
**Force-Directed Pairwise Relaxation**

As above, only after computing target location, for all \( t \)'s in an \( \varepsilon \)-neighborhood of target, find one such that \( m \) is in the \( \varepsilon \)-neighborhood of \( t \). Only interchange if true.

**Force-Directed Pairwise Relaxation**

compute forces on all modules;

foreach( primary module, m) {

  compute target location;

  in an \( \varepsilon \)-neighborhood {

    compute module which if interchanged with \( m \) reduces overall force;

    swap modules;

  }

}
Iterative Placement Algorithms

- **unconnected Sets**
  Select components by dividing them into sets which have no nets in common.

- **Simulated Annealing**
  An altered placement $\Rightarrow$ score change $\Delta s$. If $\Delta s < 0$ the move is accepted, if $\Delta s \geq 0$ the move is accepted with probability $e^{-\Delta s/t}$. 
Iterative Placement Algorithms

• Simulated Annealing Algorithm
  SA (j0, T0)
  /* j0: initial state, T0: initial temperature */
  T := T0
  X := j0
  while (stopping criterion not satisfied) do
    while (inner loop criterion not satisfied) do
      begin
        j := GENERATE(X)
        if (ACCEPT(C(j), C(X), T) X := j
      end
      T := UPDATE(T)
  end while
end while
Iterative Placement Algorithms

ACCEPT(C(j), C(i), T)
/* return 1 if the cost change passes a test */

begin
    ΔCij := C(j) - C(i)
    y := f(ΔCij, T)
    r := RANDOM(0,1)
    if (r < y) return(1)
    else return (0)
end
Pin Assignment and Gate Assignment

- **Pin Assignment**: assign nets among the functionally equivalent pins of a component, and subnets to equipotential pins.
  - Concentric Circle Mapping
  - Topological Pin Assignment
  - Nine-Zone Method
Pin Assignment and Gate Assignment

- **Gate Assignment**: Map the logical gates in the structural description onto the functionally equivalent gates

  - **Algorithm**

  1. Assign all logical gates to component that contains only one physical gate.

  2. If any component with more than one physical gate has an available gate, use it for the logical gate that has the most connections with gates already assigned to this component.

  3. If there is no partially filled component, take the next unused component and assign the first logical gate randomly.
Floorplanning

§ Models of floorplanning

• Three **classes** of cells are used:
  1. cells stored in library;
  2. cells defined, but their layouts are flexible;
  3. cells design unknown and flexible.
Floorplanning

• **Approaches** to estimate shape and area.
  
  1. *experimental*: estimate with empirical formulas.
  2. *analytical*: use wirability and routing area analysis.
  3. *procedural*: sense the cells context and optimize their interface characteristics for the current position.
  4. *knowledge-based*. 
Floorplanning

- **Approaches to Floorplanning**
  
  - *Constructive Floorplanning*
    
    1. Cluster Growth
    
    2. Connectivity Clustering
    
    3. Dual Graph Method
    
    4. Partitioning and slicing.
  
  - *Iterative Floorplanning*
    
    1. Interchange Floorplan Improvement
    
    2. Floorplanning by Relaxation
    
Applications of Placement Algorithms

§ **Large Number of Components**
- To deal with large number of components, specialized techniques must be used, such as, partitioning, hierarchical placement, hardware accelerators, or multiple processors.

§ **Variations in Component Shapes**
- Classical placement algorithms assume that placement quality is a function only of the interconnections and is not affected by the shapes of components. However, algorithms that take component shapes into consideration provide more accurate placement models.
Applications of Placement Algorithms

§  Layout design Styles

• The demands placed on and the design of placement algorithms vary greatly depending on the design styles such as Gate Array, Standard Cells, General Cells, and PCB.

§  Performance-Based Placement

• The goal of performance-based placement is to increase the speed of the circuit being designed by manipulation of the placement. Two factors must be considered: switching delays and interconnection delays.
Conclusion

- Inclusion of superior placement techniques is essential to the success of a design automation system.