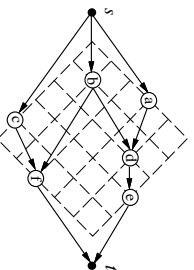
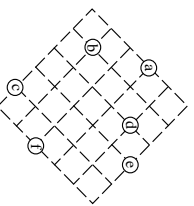
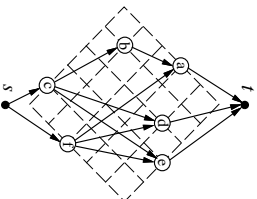


(Γ_+, Γ_-) -Packing

- For every sequence pair (Γ_+, Γ_-) , there is a (Γ_+, Γ_-) packing.
- **Horizontal constraint graph** $G_H(Y, E)$ (similarly for $G_V(Y, E)$):
 - V : source s , sink t , m vertices for modules.
 - E : (s, x) and (x, t) for each module x , and (x, x') iff x must be left-to x' .
 - **Vertex weight**: 0 for s and t , **width** of module x for the other vertices.



Packing for sequence pair: (abcdcf, cjbade)
Horizontal constraint graph (Transitive edges are not shown)



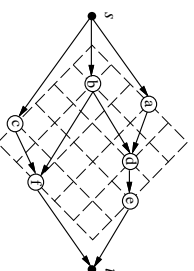
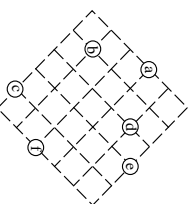
Vertical constraint graph (Transitive edges are not shown)

Summary: Floorplanning

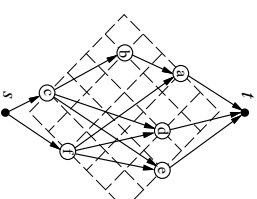
- Floorplanning objectives: (1) minimize area (2) meet timing constraints ((3) determine shapes of flexible blocks)
- Discussed approaches: Simulated annealing, mathematical programming, cluster growth, sequence pair.
- Other approaches
 - Simulated annealing for general floorplans: Sechen, DAC-88; Otten, ICCAD-84.
 - Min-cut: Lauther, J. Digital Syst., 1980.
 - Force-directed: Quinn, DAC-75.
 - Branch and bound: Onodera, 1991; Wimer, TCAD, Feb. 1989.
 - Hybrid: Whipflier, 1982 (force-directed + min-cut), DAC-82
- Other issues: Floorplanning with obstacles, soft/hard blocks.
- Open problems: Repeater floorplanner, position-constrained floor-planner

Optimal (Γ_+, Γ_-) -Packing

- **Optimal** (Γ_+, Γ_-) -Packing can be obtained in $O(m^2)$ time by applying a longest path algorithm on a vertex-weighted directed acyclic graph.
 - G_H and G_V are independent.
 - The X and Y coordinates of each module are determined as the minimum by assigning the longest path length between s and the vertex of the module in G_H and G_V , respectively.
- The set of all sequence pairs is a P-admissible solution space.



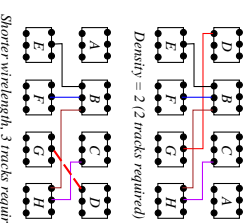
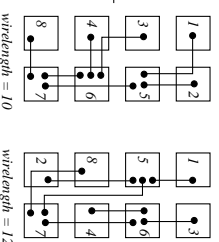
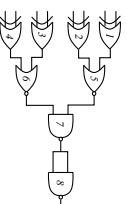
Packing for sequence pair: (abcdcf, cjbade)
Horizontal constraint graph (Transitive edges are not shown)



Vertical constraint graph (Transitive edges are not shown)

Placement

- The process of arranging the circuit components on a layout surface.
- Inputs: A set of fixed modules, a netlist.
- Goal: Find the best position for each module on the chip according to appropriate cost functions.
 - Considerations: **routability/channel density**, **wirelength**, cut size, performance, thermal issues, I/O pads.



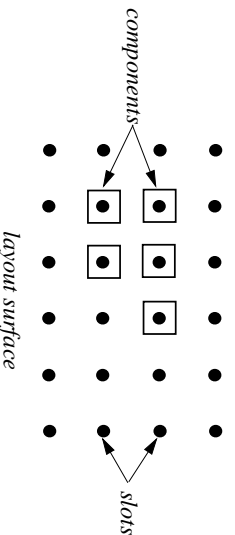
Estimation of Wirelength

- **Semi-perimeter method:** Half the perimeter of the bounding rectangle that encloses all the pins of the net to be connected. Most widely used approximation!
- **Complete graph:** Since #edges in a complete graph ($\frac{n(n-1)}{2}$) is $\frac{n}{2} \times \#$ of tree edges ($n-1$), $wirelength \approx \frac{2}{n} \sum_{(i,j) \in net} dist(i,j)$.
- **Minimum chain:** Start from one vertex and connect to the closest one, and then to the next closest, etc.
- **Source-to-sink connection:** Connect one pin to all other pins of the net. Not accurate for uncongested chips.
- **Steiner-tree approximation:** Computationally expensive.
- **Minimum spanning tree**

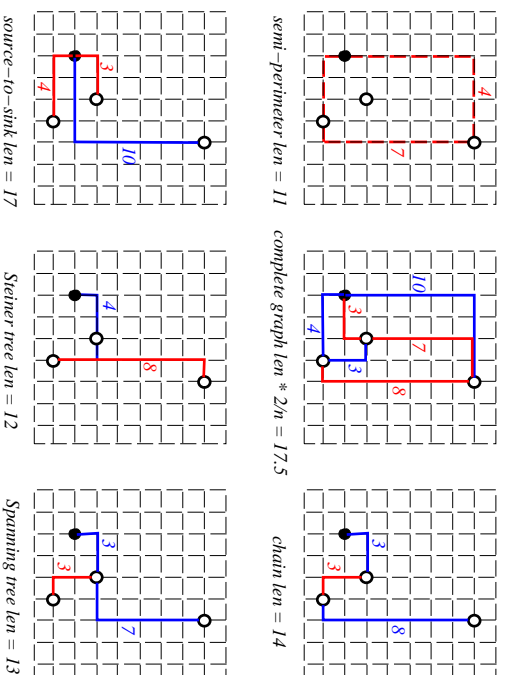
127

Placement by Cluster Growth

- Greedy method: Selects unplaced components and places them in available slots.
 - **SELECT:** Choose the unplaced component that is most strongly connected to all of the placed components (OR: most strongly connected to any single placed component).
 - **PLACE:** Place the selected component at a slot such that a certain "cost" of the partial placement is minimized.



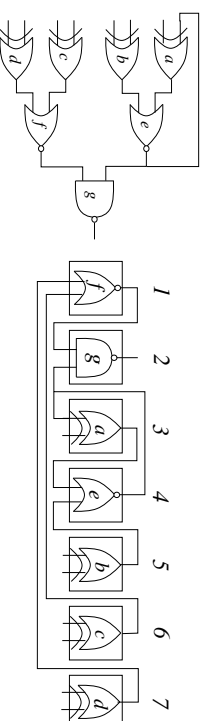
Estimation of Wirelength (cont'd)



128

Placement by Cluster Growth (cont'd)

- # of other terminals connected: $c_a = 3$, $c_b = 1$, $c_c = 1$, $c_d = 1$, $c_e = 4$, $c_f = 3$, and $c_g = 3 \Rightarrow e$ has the most connectivity.
- Place e in the center, slot 4.
- a, b, g are connected to e , and $\bar{c}_{ae} = 2, \bar{c}_{be} = \bar{c}_{eg} = 1 \Rightarrow$ Place a next to e (say, slot 3).
- Continue until all cells are placed.
- For performance consideration, we should place all cells in a sub-region close to chip I/Os if # of cells \ll # of slots.



129

130

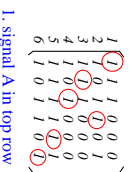
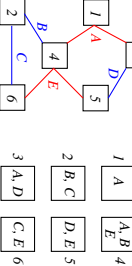
Linear Assignment

• Problem Definition:

- Instance: An $n \times n$ matrix $D = ((d_{ij}))$ with value from \mathbb{R}^+ .
- Configurations: All permutations $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$
- Solutions: All configurations
- Maximize or Minimize $D(\pi) \leftarrow \sum_{i=1}^n d_{i,\pi(i)}$

• D can be interpreted as an edge-weighted complete bipartite graph G with n vertices on each side.

• The linear assignment problem can be solved in $O(n^3)$ time (known fastest algorithm by Fredman & Targan: $O(n^2 \log n + mn)$ time).



a	b	c
d	e	f

a	b	c	d	e	f
1	0	1	1	0	0
2	1	0	1	0	0
3	1	0	1	0	0
4	1	0	1	0	0
5	1	1	1	1	0
6	1	0	1	0	0

1. signal A in top row
2. even #'s only in corners

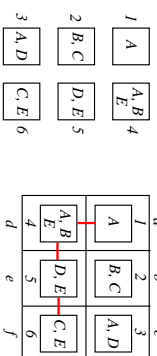
1. signal A in top row
2. even #'s only in corners
3. signal C not on right

No feasible assignment!

Placement by Linear Assignment

• Akers, "On the use of linear assignment algorithm in module placement," DAC-81.

• Total # of signal adjacencies for the placement = 3 (Signal A: between modules 1 and 4; Signal E: modules 4 and 5; Signal E: modules 5 and 6)



1	2	3	4	5	6
1	0	0	1	1	0
2	0	0	1	0	1
3	1	0	0	1	1
4	1	1	1	0	1
5	0	1	1	0	1
6	0	1	0	1	1

a	b	c	d	e	f
1	1	0	0	1	1
2	1	0	1	0	2
3	1	2	0	2	1
4	1	3	2	2	2
5	1	1	0	2	1
6	2	1	1	1	2

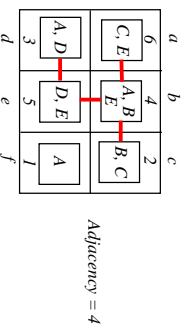
- J_{ij} : # of common signals between modules i and j .
- A_{ix} : # of signal adjacencies between module i and its neighbors if we move it to location x .
- Example: $A_{6a} = J_{62} + J_{64} = 2$; $A_{ae} = J_{a2} + J_{a4} + J_{a6} = 2$.

Linear Assignment (cont'd)

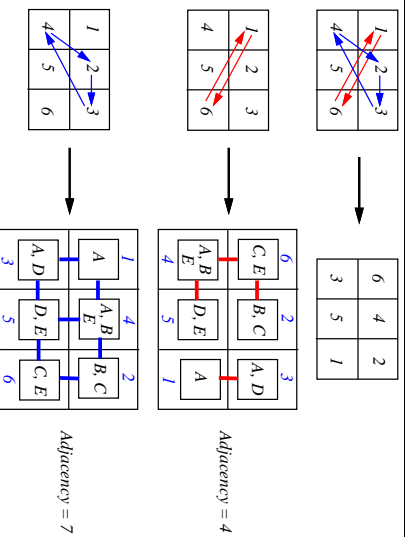
a	b	c	d	e	f
1	1	0	0	1	0
2	1	0	0	0	2
3	1	2	0	0	1
4	1	0	2	2	2
5	1	1	0	0	1
6	2	1	1	1	2

max linear assignment = $1+1+2+3+2+2 = 11$

Bad approach: Rearrange the placement according to this solution



Linear Assignment: Better Approach

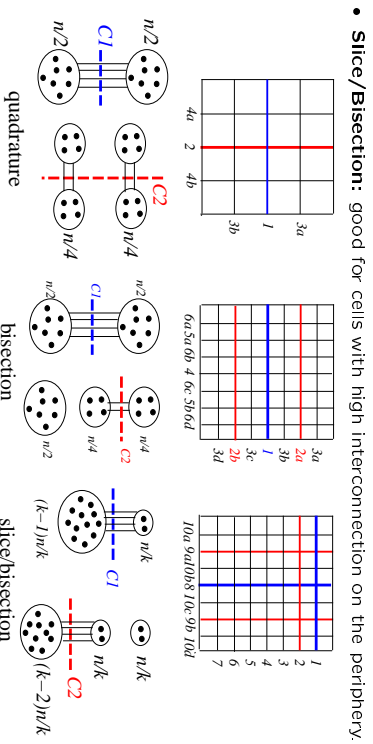


- Minor improvement.
- Reason: When a module reached a "good" position, the module previously adjacent to that position had gone elsewhere!

- Modify the placement based on a cycle that yields maximum improvement.
- Repeat the process all over again.

Min-Cut Placement

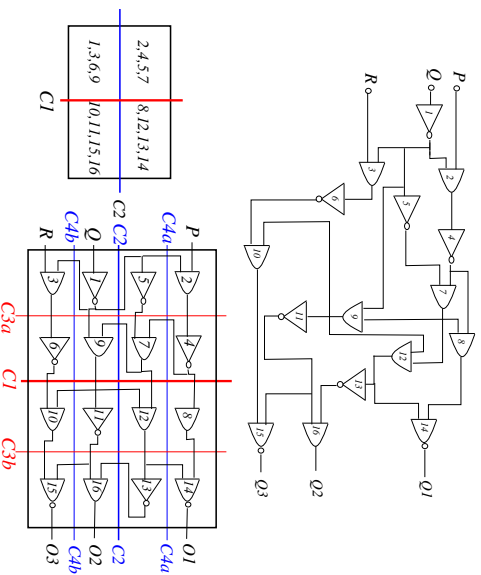
- Breuer, "A class of min-cut placement algorithms." DAC-77.
- **Quadrature:** suitable for circuits with high density in the center.
- **Bisection:** good for standard-cell placement.



135

Quadrature Placement Example

- Apply K-L heuristic to partition + Quadrature Placement: Cost $C_1 = 4$, $C_{2L} = C_{2R} = 2$, etc.



137

Algorithm for Min-Cut Placement

```

Algorithm: Min_Cut_Placement( $N, n, C$ )
/*  $N$ : the layout surface */
/*  $n$ : # of cells to be placed */
/*  $n_0$ : # of cells in a slot */
/*  $C$ : the connectivity matrix */

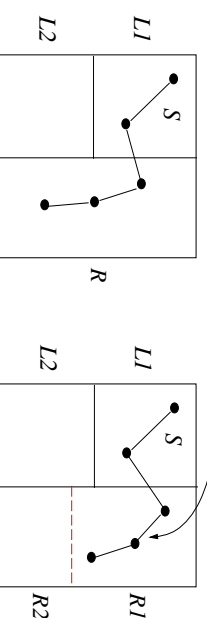
1 begin
2 if ( $n \leq n_0$ ) then PlaceCells( $N, n, C$ );
3 else
4   ( $N_1, N_2$ )  $\leftarrow$  Cutsurface( $N$ );
5   ( $n_1, C_1$ ), ( $n_2, C_2$ )  $\leftarrow$  Partition( $n, C$ );
6   Call Min_Cut_Placement( $N_1, n_1, C_1$ );
7   Call Min_Cut_Placement( $N_2, n_2, C_2$ );
8 end

```

136

Min-Cut Placement with Terminal Propagation

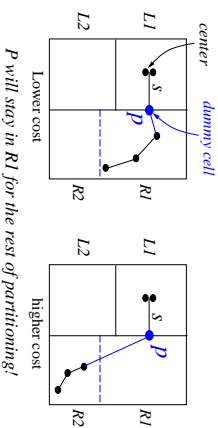
- Dunlop & Kernighan, "A procedure for placement of standard-cell VLSI circuits," IEEE TCAD, Jan. 1985.
- Drawback of the original min-cut placement: Does not consider the positions of terminal pins that enter a region.
 - What happens if we swap {1, 3, 6, 9} and {2, 4, 5, 7} in the previous example?



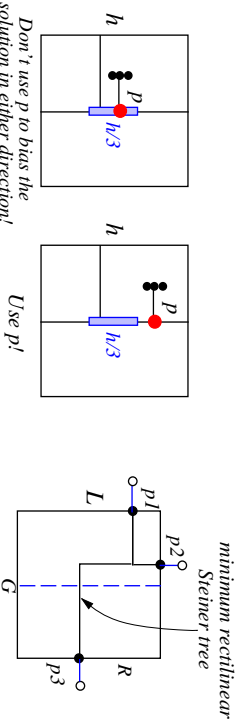
138

Terminal Propagation

- We should use the fact that s is in L_1 !



- When not to use p to bias partitioning? Net s has cells in many groups?



139

Placement by Simulated Annealing

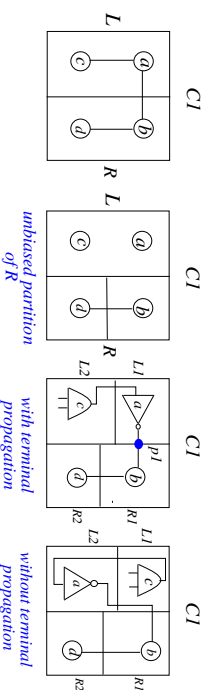
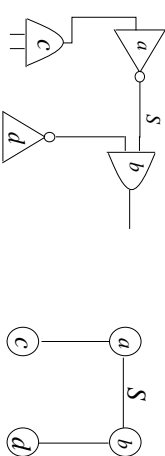
- Sechen and Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. Solid-State Circuits*, Feb. 1985; "TimberWolf 3.2: A new standard cell placement and global routing package," DAC-86.

- TimberWolf: Stage 1
 - Modules are moved between different rows as well as within the same row.
 - Modules overlaps are allowed.
 - When the temperature is reached below a certain value, stage 2 begins.
- TimberWolf: Stage 2
 - Remove overlaps.
 - Annealing process continues, but only interchanges adjacent modules within the same row.

141

Terminal Propagation Example

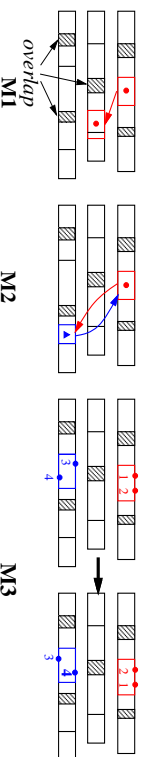
- Partitioning must be done breadth-first, not depth-first.



140

Solution Space & Neighborhood Structure

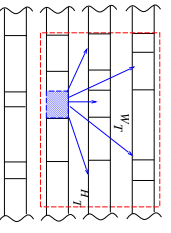
- **Solution Space:** All possible arrangements of the modules into rows, possibly with overlaps.
- **Neighborhood Structure:** 3 types of moves
 - M_1 : Displace a module to a new location.
 - M_2 : Interchange two modules.
 - M_3 : Change the orientation of a module.



142

Neighborhood Structure

- TimberWolf first tries to select a move between M_1 and M_2 : $Prob(M_1) = 0.8$, $Prob(M_2) = 0.2$.
- If a move of type M_1 is chosen and it is rejected, then a move of type M_3 for the same module will be chosen with probability 0.1.
- Restrictions: (1) what row for a module can be displaced? (2) what pairs of modules can be interchanged?
- **Key: Range Limiter**
 - At the beginning, (W_T, H_T) is very large, big enough to contain the whole chip.
 - Window size shrinks slowly as the temperature decreases. Height and width $\propto \log(T)$.
 - Stage 2 begins when window size is so small that no inter-row module interchanges are possible.



Cost Function

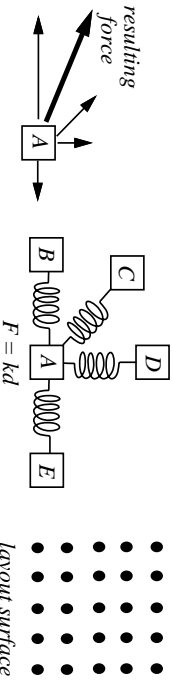
- Cost function: $C = C_1 + C_2 + C_3$.
- **C_1 : total estimated wirelength.**
 - $C_1 = \sum_{i \in Nets} (\alpha_i w_i + \beta_i h_i)$
 - α_i, β_i are horizontal and vertical weights, respectively. ($\alpha_i = 1, \beta_i = 1 \Rightarrow \frac{1}{2} \times$ perimeter of the bounding box of Net i .)
 - Critical nets: Increase both α_i and β_i .
 - If vertical wirings are "cheaper" than horizontal wirings, use smaller vertical weights: $\beta_i < \alpha_i$.
- **C_2 : penalty function for module overlaps.**
 - $C_2 = \gamma \sum_{i \neq j} O_{ij}^2$, γ : penalty weight.
 - O_{ij} : amount of overlaps in the x -dimension between modules i and j .
- **C_3 : penalty function that controls the row length.**
 - $C_3 = \delta \sum_{r \in Rows} |L_r - D_r|^p$, δ : penalty weight.
 - D_r : desired row length.
 - L_r : sum of the widths of the modules in row r .

Annealing Schedule

- $T_k = r_k T_{k-1}, k = 1, 2, 3, \dots$
- r_k increases from 0.8 to max value 0.94 and then decreases to 0.8.
- At each temperature, a total # of n_P attempts is made. n : # of modules; P : user specified constant.
- Termination: $T < 0.1$.

Placement by the Force-Directed Method

- Hanan & Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems*, Breuer, Ed, 1972.
- Quinn, Jr. & Breuer, "A force directed component placement procedure for printed circuit boards," *IEEE Trans. Circuits and Systems*, June 1979.
- Reducing the placement problem to solving a set of simultaneous linear equations to determine equilibrium locations for cells.
- Analogy to Hooke's law: $F = kd$, F : force, k : spring constant, d : distance.
- Goal: Map cells to the layout surface.



Finding the Zero-Force Target Location

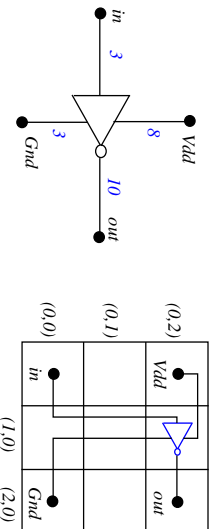
- Cell i connects to several cells j 's at distances d_{ij} 's by wires of weights w_{ij} 's.
- Total force: $F_i = \sum_j w_{ij}d_{ij}$

- The zero-force target location (\hat{x}_i, \hat{y}_i) can be determined by equating the x - and y -components of the forces to zero:

$$\sum_j w_{ij} \cdot (x_j - \hat{x}_i) = 0 \Rightarrow \hat{x}_i = \frac{\sum_j w_{ij}x_j}{\sum_j w_{ij}}$$

$$\sum_j w_{ij} \cdot (y_j - \hat{y}_i) = 0 \Rightarrow \hat{y}_i = \frac{\sum_j w_{ij}y_j}{\sum_j w_{ij}}$$

- In the example, $\hat{x}_i = \frac{8 \times 0 + 10 \times 2 + 3 \times 0 + 3 \times 2}{8 + 10 + 3 + 3} = 1.083$ and $\hat{y}_i = 1.50$.



147

Force-Directed Placement

- Approach I (Constructive):
 - Start with an initial placement.
 - Compute the zero-force locations for all cells.
 - Apply **linear assignment** to determine the "ideal" locations for the cells.
- Approach II (can be constructive or iterative):
 - Start with an initial placement.
 - Select a "most profitable" cell p (e.g., maximum F , critical cells) and place it in its zero-force location.
 - "Fix" placement if the zero-location has been occupied by another cell q .
- Popular options to fix:
 - Ripple move:** place p in the occupied location, compute a new zero-force location for q , ...
 - Chain move:** place p in the occupied location, move q to an adjacent location, ...
 - Move p to a free location close to q .

148

Algorithm: Force-Directed Placement

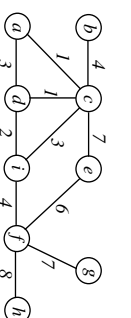
```

1 begin
2 Compute the connectivity for each cell;
3 Sort the cells in decreasing order of their connectivities into list  $L$ ;
4 while ( $IterationCount < IterationLimit$ ) do
5   Seed  $\leftarrow$  next module from  $L$ ;
6   Declare the position of the seed vacant;
7   while ( $EndRipple = FALSE$ ) do
8     Compute target location of the seed;
9     case the target location
10    VACANT:
11      Move seed to the target location and lock;
12       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow 0$ ;
13    SAME AS PRESENT LOCATION:
14       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow 0$ ;
15    LOCKED:
16      Move selected cell to the nearest vacant location;
17       $EndRipple \leftarrow TRUE$ ;  $AbortCount \leftarrow AbortCount + 1$ ;
18      if ( $AbortCount > AbortLimit$ ) then
19        Unlock all cell locations;
20         $IterationCount \leftarrow IterationCount + 1$ ;
21    OCCUPIED AND NOT LOCKED:
22      Select cell as the target location for next move;
23      Move seed cell to target location and lock the target location;
24       $EndRipple \leftarrow FALSE$ ;  $AbortCount \leftarrow 0$ ;
25 end

```

Placement by the Genetic Algorithm

- Cohon & Paris, "Genetic placement," ICCAD-86.
- Genetic algorithm:** A search technique that emulates the biological evolution process to find the optimum.
- Generic approaches:
 - Start with an initial set of random configurations (**population**); each individual is a string of symbol (symbol string \leftrightarrow **chromosome**: a solution to the optimization problem, symbol \leftrightarrow **gene**).
 - During each iteration (**generation**), the individuals are evaluated using a **fitness** measurement.
 - Two fitter individuals (**parents**) at a time are selected to generate new solutions (**offsprings**).
 - Genetic operators: **crossover**, **mutation**, **inversion**
- In the example, string = [aghdhdef]; fitness value = $1 / \sum_{(i,j) \in E} w_{ij}d_{ij} = 1/85$.



6	7	8
3	4	5
0	1	2

d	e	f
c	b	i
a	g	h

string: aghdhdef

149

150

Two More Crossover Operations

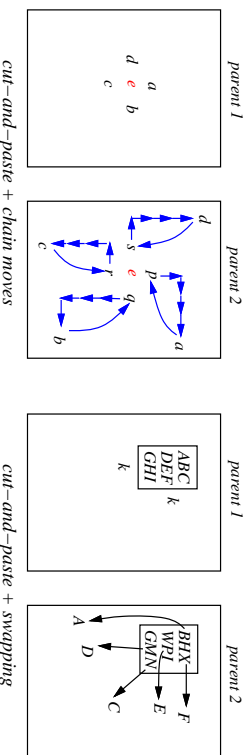
- Main genetic operator: Operate on two individuals and generates an offspring.
 - $[b|d|e|f|a|g|h|a] (\frac{1}{86}) + [b|d|e|f|i|g|h|a] (\frac{1}{110}) \rightarrow [b|d|e|f|g|h|a] (\frac{1}{93})$.
 - Need to avoid repeated symbols in the solution string!
- **Partially mapped crossover** for avoiding repeated symbols:
 - $[b|d|e|f|g|h|a] (\frac{1}{86}) + [a|g|h|c|b|i|d|e|f] (\frac{1}{88}) \rightarrow [b|g|h|a|i|d|e|f]$.
 - Copy $idef$ to the offspring; scan $[b|d|e|f|g|h|a]$ from the left, and then copy all unrepeated genes.

- **Cut-and-paste + Chain moves:**

- Copy a randomly selected cell e and its four neighbors from parent 1 to parent 2.
- The cells that earlier occupied the neighboring locations in parent 2 are shifted outwards.

- **Cut-and-paste + Swapping**

- Copy $k \times k$ square modules from parent 1 to parent 2 (k : random # from a normal distribution with mean 3 and variance 1).
- Swap cells not in both square modules.



Genetic Operator: Crossover

- Main genetic operator: Operate on two individuals and generates an offspring.
 - $[b|d|e|f|a|g|h|a] (\frac{1}{86}) + [b|d|e|f|i|g|h|a] (\frac{1}{110}) \rightarrow [b|d|e|f|g|h|a] (\frac{1}{93})$.
 - Need to avoid repeated symbols in the solution string!
- **Partially mapped crossover** for avoiding repeated symbols:
 - $[b|d|e|f|g|h|a] (\frac{1}{86}) + [a|g|h|c|b|i|d|e|f] (\frac{1}{88}) \rightarrow [b|g|h|a|i|d|e|f]$.
 - Copy $idef$ to the offspring; scan $[b|d|e|f|g|h|a]$ from the left, and then copy all unrepeated genes.

Genetic Operators: Mutation & Inversion

- **Mutation:** prevents loss of diversity by introducing new solutions.
 - Incremental random changes in the offspring generated by the crossover.
 - A commonly used mutation: pairwise interchange.
- **Inversion:** $[b|d|i|e|f|g|h|a] \rightarrow [b|i|d|h|c|g|f|e|a]$.
- Apply mutation and inversion with probability P_μ and P_i , respectively.

```

Algorithm: Genetic Placement( $N_p, N_g, N_o, P_i, P_\mu$ )
/*  $N_p$ : population size; */
/*  $N_g$ : # of generation; */
/*  $N_o$ : # of offspring; */
/*  $P_i$ : inversion probability; */
/*  $P_\mu$ : mutation probability; */
1 begin
2 ConstructPopulation( $N_p$ ): /* randomly generate the initial population */
3 for  $j \leftarrow 1$  to  $N_p$ 
4 Evaluate Fitness(population( $N_p$ ));
5 for  $i \leftarrow 1$  to  $N_g$ 
6   for  $j \leftarrow 1$  to  $N_o$ 
7     /* choose parents with probability proportional to fitness value */
8     ( $x, y$ )  $\leftarrow$  ChooseParents;
9     /* perform crossover to generate offspring */
10    offspring( $j$ )  $\leftarrow$  GenerateOffspring( $x, y$ );
11    for  $h \leftarrow 1$  to  $N_p$ 
12      With probability  $P_i$ , apply Mutation(population( $h$ ));
13      for  $h \leftarrow 1$  to  $N_p$ 
14        With probability  $P_i$ , apply Inversion(population( $h$ ));
15      Evaluate Fitness(offspring( $j$ ));
16      population  $\leftarrow$  Select(population, offspring,  $N_p$ );
17 return the highest scoring configuration in population;
18 end
    
```

Genetic Placement Experiment: GINIE

- Termination condition: no improvement in the best solution for 10,000 generations.
- Population size: 50. (Each generation: 50 unchanged throughout the process.)
- Each generation creates 12 offsprings.
- Comparisons with simulated annealing:
 - Similar quality of solutions and running time.

155

Summary: Placement

- Discussed methods: min-cut (w/ terminal propagation), force-directed, simulated annealing (SA), genetic algorithm (GA).
- Better results: min-cut w/ terminal propagation + force-directed + simulated annealing.
- Other approaches
 - Simulated evolution (GA + data structure): Kling & Bannerjee, DAC-87.
 - Force-directed: Goto, IEEE TCS, Jan. 1981.
 - Fuzzy logic: Lin & Shragowitz, DAC-92.
- Other issues: timing-driven placement (Dunlop, DAC-84; Marek-Sadowska, ICCAD-89; Srinivasan, TCS, Nov. 1992), thermal consideration (Chu & Wong, ISPD-97), interaction with partitioning/clustering and routing.
- A survey: Shahooar & Mazumder, "VLSI cell placement techniques," *ACM Computing Surveys*, June 1991.

156