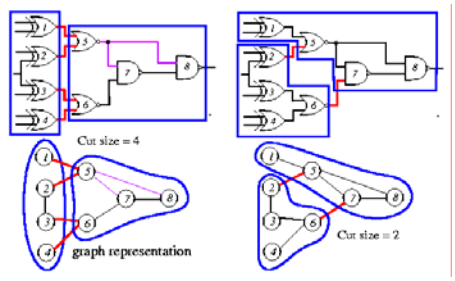
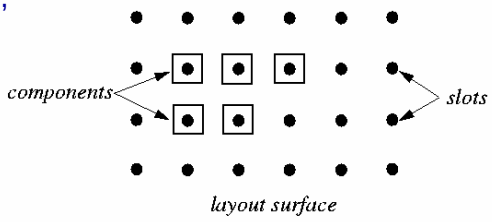
Circuit Partitioning & Placement

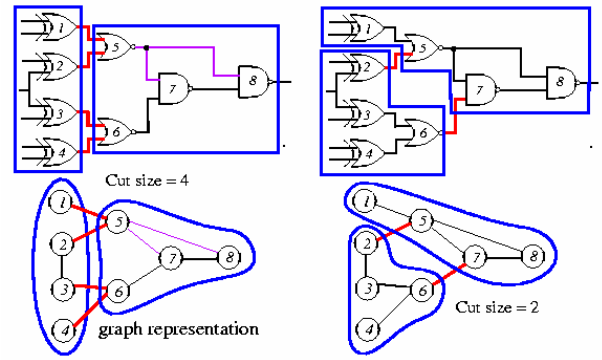
- Course contents:
  - Kernighang-Lin partitioning heuristic
  - Placement metrics
  - Constructive placement: cluster growth, min cut
  - Iterative placement: force-directed method, simulated annealing, genetic algorithm
- Reading
  - Chapter 7





Circuit Partitioning

- **Objective:** Partition a circuit into parts such that every component is within a prescribed range and the # of connections among the components is minimized.
  - More constraints are possible for some applications.
- Cutset? Cut size? Size of a component?



Problem Definition: Partitioning

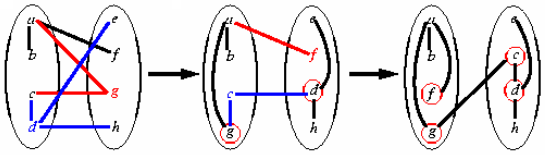
- ***k*-way partitioning:** Given a graph $G(V, E)$, where each vertex $v \in V$ has a **size** $s(v)$ and each edge $e \in E$ has a **weight** $w(e)$, the problem is to divide the set V into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that an objective function is optimized, subject to certain constraints.
- **Bounded size constraint:** The size of the i-th subset is bounded by B_i ($\sum_{v \in V_i} s(v) \leq B_i$).
  - Is the partition balanced?
- **Min-cut cost between two subsets:** Minimize $\sum_{\forall e=(u,v) \wedge p(u) \neq p(v)} w(e)$, where $p(u)$ is the partition # of node u.
- The 2-way, balanced partitioning problem is NP-complete, even in its simple form with identical vertex sizes and unit edge weights.

Kernighan-Lin Heuristic

- Kernighan and Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, Feb. 1970.
- An **iterative**, **2-way**, **balanced** partitioning (bi-sectioning) heuristic.
- Till the cut size keeps decreasing
  - Vertex pairs which give the largest decrease **or the smallest increase** in cut size are exchanged.
  - These vertices are then **locked** (and thus are prohibited from participating in any further exchanges).
  - This process continues until all the vertices are locked.
  - Find the set with the **largest partial sum** for swapping.
  - Unlock all vertices.

K-L Heuristic: A Simple Example

- Each edge has a unit weight.



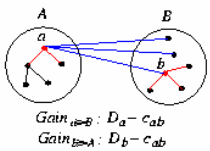
Step #	Vertex pair	Cost reduction	Cut cost
0	-	0	5
1	{d, g}	3	2
2	{c, f}	1	1
3	{b, h}	-2	3
4	{a, e}	-2	5

- Questions: How to compute cost reduction? What pairs to be swapped?
  - Consider the change of internal & external connections.

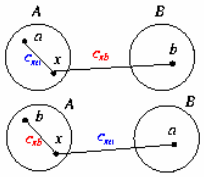
Properties

- Two sets A and B such that $|A| = n = |B|$ and $A \cap B = \emptyset$.
- **External cost** of $a \in A$: $E_a = \sum_{v \in B} c_{av}$.
- **Internal cost** of $a \in A$: $I_a = \sum_{v \in A} c_{av}$.
- D-value of a vertex a: $D_a = E_a - I_a$ (cost reduction for moving a).
- Cost reduction (gain) for swapping a and b: $g_{ab} = D_a + D_b - 2c_{ab}$.
- If $a \in A$ and $b \in B$ are interchanged, then the new D-values, D', are given by

$$D'_x = D_x + 2c_{xa} - 2c_{xb}, \forall x \in A - \{a\}$$
$$D'_y = D_y + 2c_{yb} - 2c_{ya}, \forall y \in B - \{b\}.$$



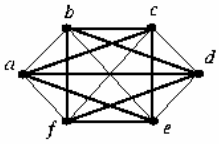
Internal cost vs. External cost



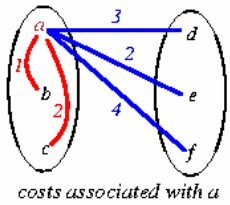
updating D-values

before swap	*after swap*	$\triangle C$
$-c_{xa}$	$+c_{xa}$	$+2c_{xa}$
$+c_{xb}$	$-c_{xb}$	$-2c_{xb}$

K-L Heuristic: A Weighted Example



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



costs associated with *a*

Initial cut cost = (3+2+4)+(4+2+1)+(3+2+1) = 22

- Iteration 1:

$$I_a = 1+2 = 3; \quad E_a = 3+2+4 = 9; \quad D_a = E_a - I_a = 9-3 = 6$$
$$I_b = 1+1 = 2; \quad E_b = 4+2+1 = 7; \quad D_b = E_b - I_b = 7-2 = 5$$
$$I_c = 2+1 = 3; \quad E_c = 3+2+1 = 6; \quad D_c = E_c - I_c = 6-3 = 3$$
$$I_d = 4+3 = 7; \quad E_d = 3+4+3 = 10; \quad D_d = E_d - I_d = 10-7 = 3$$
$$I_e = 4+2 = 6; \quad E_e = 2+2+2 = 6; \quad D_e = E_e - I_e = 6-6 = 0$$
$$I_f = 3+2 = 5; \quad E_f = 4+1+1 = 6; \quad D_f = E_f - I_f = 6-5 = 1$$

Computing the g Value

- Iteration 1:

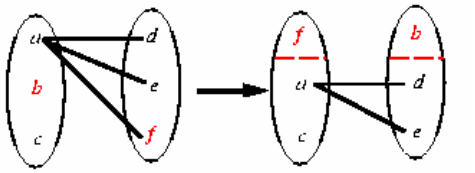
$$I_a = 1+2 = 3; \quad E_a = 3+2+4 = 9; \quad D_a = E_a - I_a = 9-3 = 6$$
$$I_b = 1+1 = 2; \quad E_b = 4+2+1 = 7; \quad D_b = E_b - I_b = 7-2 = 5$$
$$I_c = 2+1 = 3; \quad E_c = 3+2+1 = 6; \quad D_c = E_c - I_c = 6-3 = 3$$
$$I_d = 4+3 = 7; \quad E_d = 3+4+3 = 10; \quad D_d = E_d - I_d = 10-7 = 3$$
$$I_e = 4+2 = 6; \quad E_e = 2+2+2 = 6; \quad D_e = E_e - I_e = 6-6 = 0$$
$$I_f = 3+2 = 5; \quad E_f = 4+1+1 = 6; \quad D_f = E_f - I_f = 6-5 = 1$$

- $g_{xy} = D_x + D_y - 2c_{xy}$.

$$\begin{aligned}
g_{ad} &= D_a + D_d - 2c_{ad} = 6+3-2\times 3 = 3 \\
g_{ae} &= 6+0-2\times 2 = 2 \\
g_{af} &= 6+1-2\times 4 = -1 \\
g_{bd} &= 5+3-2\times 4 = 0 \\
g_{be} &= 5+0-2\times 2 = 1 \\
g_{bf} &= 5+1-2\times 1 = 4 \ (maximum) \\
g_{cd} &= 3+3-2\times 3 = 0 \\
g_{ce} &= 3+0-2\times 2 = -1 \\
g_{cf} &= 3+1-2\times 1 = 2
\end{aligned}$$

- Swap *b* and *f*. $(\hat{g_1} = 4)$

Updating the D Value



- $D'_x = D_x + 2\,c_{xp} - 2\,c_{xq}$, $\forall\, x \in A - \{p\}$ (swap p and q, $p \in A$, $q \in B$)

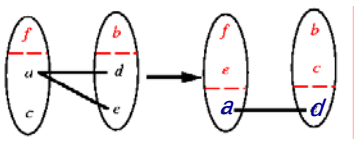
$$\begin{aligned} D'_a &= D_a + 2c_{ab} - 2c_{af} = 6 + 2\times 1 - 2\times 4 = 0 \\ D'_c &= D_c + 2c_{cb} - 2c_{cf} = 3 + 2\times 1 - 2\times 1 = 3 \\ D'_d &= D_d + 2c_{df} - 2c_{db} = 3 + 2\times 3 - 2\times 4 = 1 \\ D'_e &= D_e + 2c_{ef} - 2c_{eb} = 0 + 2\times 2 - 2\times 2 = 0 \end{aligned}$$

- $g_{xy} = D'_x + D'_y - 2c_{xy}$.

$$\begin{aligned} g_{ad} &= D'_a + D'_d - 2c_{ad} = 0 + 1 - 2\times 3 = -5 \\ g_{ae} &= D'_a + D'_e - 2c_{ae} = 0 + 0 - 2\times 2 = -4 \\ g_{cd} &= D'_c + D'_d - 2c_{cd} = 3 + 1 - 2\times 3 = -2 \\ g_{ce} &= D'_c + D'_e - 2c_{ce} = 3 + 0 - 2\times 2 = -1 \ (maximum) \end{aligned}$$

- Swap c and e. $(\hat{g_2} = -1)$

Determining Swapping Pairs



- $D''_x = D'_x + 2\,c_{xp} - 2\,c_{xq}$, $\forall\, x \in A - \{p\}$

$$\begin{aligned} D''_a &= D'_a + 2c_{ac} - 2c_{ae} = 0 + 2\times 2 - 2\times 2 = 0 \\ D''_d &= D'_d + 2c_{de} - 2c_{dc} = 1 + 2\times 4 - 2\times 3 = 3 \end{aligned}$$

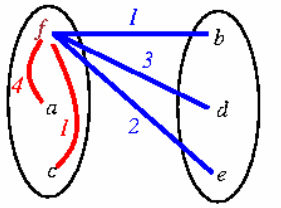
- $g_{xy} = D''_x + D''_y - 2c_{xy}$.

$$g_{ad} = D''_a + D''_d - 2c_{ad} = 0 + 3 - 2\times 3 = -3 (\hat{g_3} = -3)$$

- Note that this step is redundant $(\sum_{i=1}^{n} \hat{g_i} = 0)$.
- Summary: $\hat{g_1} = g_{bf} = 4,\ \hat{g_2} = g_{ce} = -1,\ \hat{g_3} = g_{ad} = -3.$
- Largest partial sum $\max \sum_{i=1}^{k} \hat{g_i} = 4$ (k = 1) $\Rightarrow$ Swap b and f.

Next Iteration

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



Initial cut cost = (1+3+2)+(1+3+2)+(1+3+2) = 18 (22−4)

- Iteration 2: Repeat what we did at Iteration 1 (Initial cost = 22-4 =18).
- Summary: $\hat{g}_1 = g_{ce} = -1$, $\hat{g}_2 = g_{ab} = -3$, $\hat{g}_3 = g_{fd} = 4$.
- Largest partial sum = $\max \sum_{i=1}^{k} \hat{g}_i = 0$ (k = 3) ⇒ Stop!

Kernighan-Lin Heuristic

Algorithm: Kernighan-Lin(G)
Input: $G = (V, E), |V| = 2n$.
Output: Balanced bi-partition A and B with "small" cut cost.

```
1  begin
2  Bipartition G into A and B such that |V_A| = |V_B|, V_A ∩ V_B = ∅,
       and V_A ∪ V_B = V.
3  repeat
4     Compute D_v, ∀v ∈ V.
5     for i = 1 to n do
6        Find a pair of unlocked vertices v_ai ∈ V_A and v_bi ∈ V_B whose
         exchange makes the largest decrease or smallest increase in
         cut cost;
7        Mark v_ai and v_bi as locked, store the gain ĝ_i, and compute
         the new D_v, for all unlocked v ∈ V;
8     Find k, such that G_k = Σ_{i=1}^{k} ĝ_i is maximized;
9     if G_k > 0 then
10       Move v_a1,...,v_ak from V_A to V_B and v_b1,...,v_bk from V_B to V_A;
11    Unlock v, ∀v ∈ V.
12 until G_k ≤ 0;
13 end
```

Time Complexity

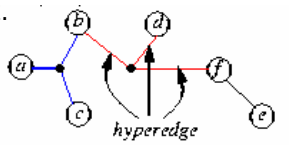
- Line 4: Initial computation of D: $O(n^2)$
- Line 5: The **for**-loop: $O(n)$
- The body of the loop: $O(n^2)$
  - Lines 6--7: Step i takes $(n - i + 1)^2$ time
- Lines 4--11: Each pass of the repeat loop: $O(n^3)$
- Suppose the repeat loop terminates after r passes
- The total running time: $O(rn^3)$
  - Polynomial-time algorithm?

Extensions of K-L Heuristic

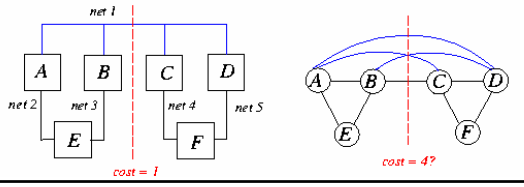
- **Unequal sized subsets** (assume $n_1 < n_2$)
  1. Partition: $|A| = n_1$ and $|B| = n_2$.
  2. Add $n_2 - n_1$ dummy vertices to set A. Dummy vertices have no connections to the original graph.
  3. Apply the Kernighan-Lin algorithm.
  4. Remove all dummy vertices.
- **Unequal sized "vertices"**
  1. Assume that the smallest "vertex" has unit size.
  2. Replace each vertex of size s with s vertices which are fully connected with edges of infinite weight.
  3. Apply the Kernighan-Lin algorithm.
- ***k*-way partition**
  1. Partition the graph into k equal-sized sets.
  2. Apply the Kernighan-Lin algorithm for each pair of subsets.
  3. Time complexity? Can be reduced by recursive bi-partition.
- How to **handle hypergraphs**?
  - Need to handle multi-terminal nets directly.

Coping with Hypergraph

- A hypergraph $H = (N, L)$ consists of a set N of vertices and a set L of hyperedges, where each hyperedge corresponds to a **subset** N_i of distinct vertices with $|N_i| \geq 2$.

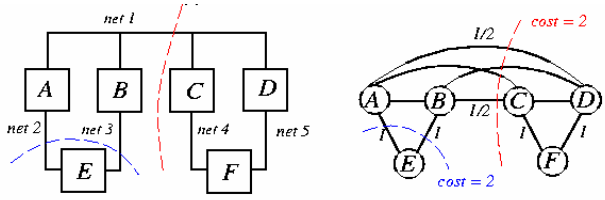


- Schweikert and Kernighan, "A proper model for the partitioning of electrical circuits," 9th Design Automation Workshop, 1972.
- For multi-terminal nets, **net cut** is a more accurate measurement for cut cost (i.e., deal with hyperedges).
  - {A, B, E}, {C, D, F} is a good partition.
  - Should not assign the same weight for all edges.

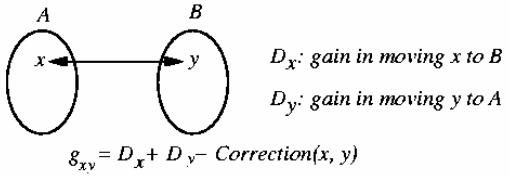


Net-Cut Model

- Let $n(i)$ = # of cells associated with Net i.
- Edge weight $w_{xy} = \frac{2}{n(i)}$ for an edge connecting cells x and y.



- Easy modification of the K-L heuristic.

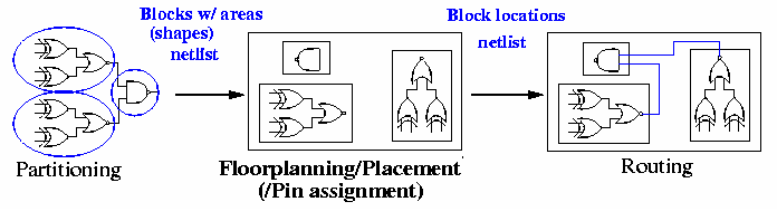


A Linear-Time Heuristic for Partitioning

- Fiduccia and Mattheyses, “A linear time heuristic for improving network partitions,” DAC-82.
- New features to the K-L heuristic:
  - Aims at **reducing net-cut costs**; the concept of cutsize is extended to hypergraphs.
  - Only a **single vertex is moved** across the cut in a single move.
  - Vertices are weighted.
  - Can handle “unbalanced” partitions; a balance factor is introduced.
  - A special data structure is used to select vertices to be moved across the cut to improve running time.
  - **Time complexity *O*(*P*),** where *P* is the total # of terminals.

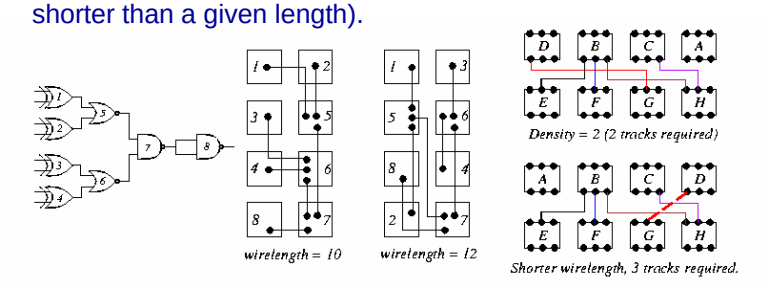
Placement

- **Placement** is the problem of automatically assigning correct positions on the chip to predesigned cells, such that some cost function is optimized.
- Inputs: A set of **fixed** cells/modules, a netlist.
- Goal: Find the best position for each cell/module on the chip according to appropriate cost functions.
  - Considerations: **routability/channel density**, **wirelength**, cut size, performance, thermal issues, I/O pads.



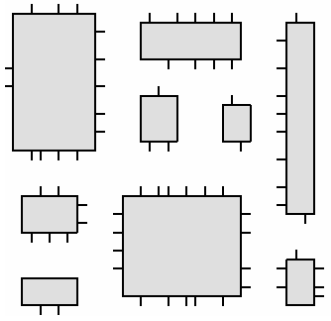
Placement Objectives and Constraints

- What does a placement algorithm try to optimize?
  - the total area
  - the total wire length
  - the number of horizontal/vertical wire segments crossing a line
- Constraints:
  - the placement should be routable (no cell overlaps; no density overflow).
  - timing constraints are met (some wires should always be shorter than a given length).



VLSI Placement: Building Blocks

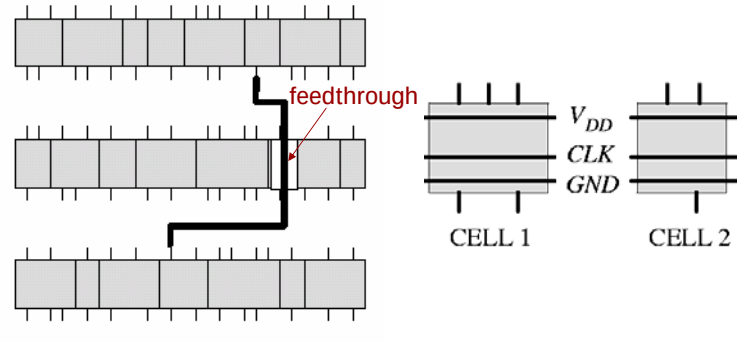
- Different design styles create different placement problems.
  - E.g., building-block, standard-cell, gate-array placement
- Building block: The cells to be placed have arbitrary shapes.



Building block

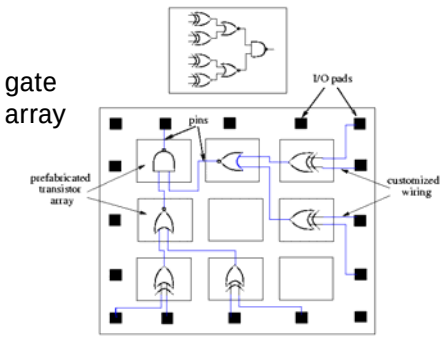
VLSI Placement: Standard Cells

- Standard cells are designed in such a way that power and clock connections run horizontally through the cell and other I/O leaves the cell from the top or bottom sides.
- The cells are placed in rows.
- Sometimes feedthrough cells are added to ease wiring.



Consequences of Fabrication Method

- Full-custom fabrication (building block):
  - Free selection of aspect ratio (quotient of height and width).
  - Height of wiring channels can be adapted to necessity.
- Semi-custom fabrication (gate array, standard cell):
  - Placement has to deal with fixed carrier dimensions.
  - Placement should be able to deal with fixed channel capacities.



Relation with Routing

- Ideally, placement and routing should be performed simultaneously as they depend on each other's results. This is, however, too complicated.
  - P&R: placement and routing
- In practice placement is done prior to routing. The placement algorithm estimates the wire length of a net using some *metric*.

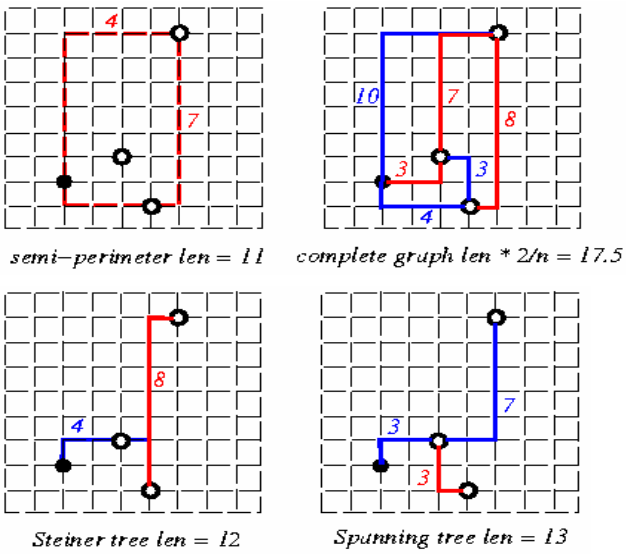
Estimation of Wirelength

- **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!
- **Squared Euclidean distance:** Squares of all pairwise terminal distances in a net using a quadratic cost function

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\gamma_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2]$$

- **Steiner-tree approximation:** Computationally expensive.
- **Minimum spanning tree**: Good approximation to Steiner trees.
- **Complete graph:** Since #edges in a complete graph is $\left(\frac{n(n-1)}{2}\right)$ = $\frac{n}{2}$ × # of tree edges (n-1), *wirelength* ≈ $\frac{2}{n}\sum_{(i,j)\in net} dist(i, j)$.

Estimation of Wirelength (cont'd)

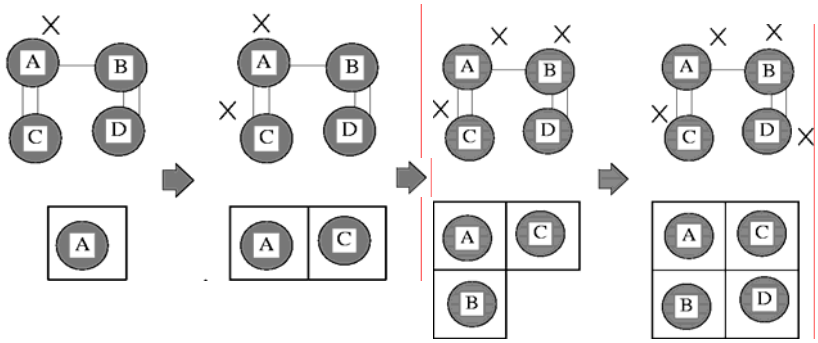


Placement Algorithms

- The placement problem is NP-complete
- Popular placement algorithms:
  - **Constructive algorithms:** once the position of a cell is fixed, it is not modified anymore.
    - Cluster growth, min cut, etc.
  - **Iterative algorithms:** intermediate placements are modified in an attempt to improve the cost function.
    - Force-directed method, etc
  - **Nondeterministic approaches:** simulated annealing, genetic algorithm, etc.
- Most approaches combine multiple elements:
  - Constructive algorithms are used to obtain an initial placement.
  - The initial placement is followed by an iterative improvement phase.
  - The results can further be improved by simulated annealing.

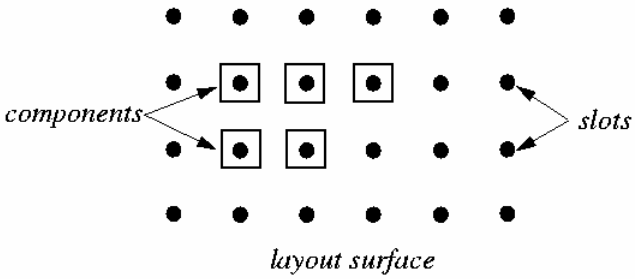
Bottom-Up Placement: Clustering

- Starts with a single cell and finds more cells that share nets with it.



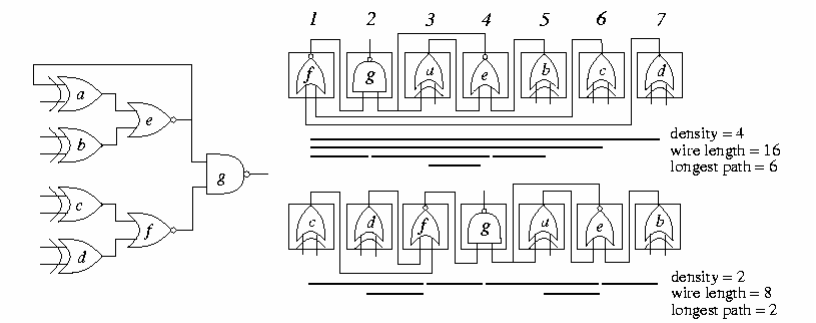
Placement by Cluster Growth

- 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.



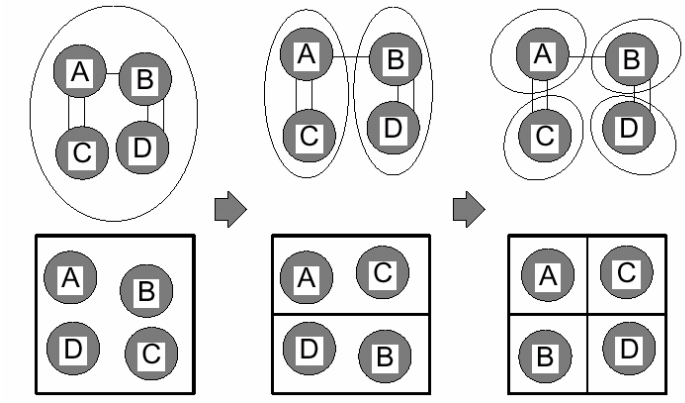
Cluster Growth Example

- # 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 ⇒ *e* has the most connectivity.
- Place *e* in the center, slot 4. *a, b, g* are connected to *e*, and $\hat{c}_{ae} = 2, \hat{c}_{be} = \hat{c}_{eg} = 1$ ⇒ Place *a* next to *e* (say, slot 3). Continue until all cells are placed.
- Further improve the placement by swapping the gates.



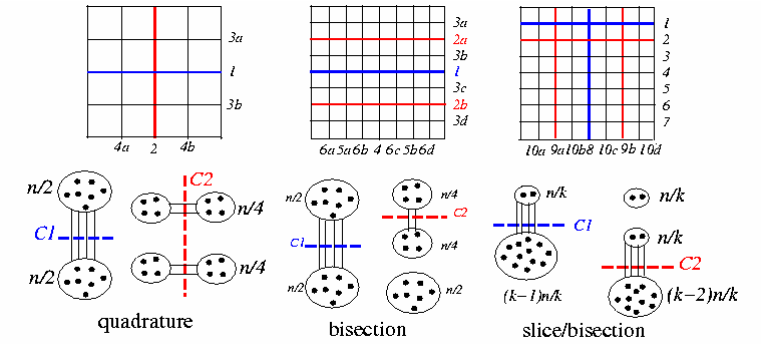
Top-down Placement: Min Cut

- Starts with the whole circuit and ends with small circuits.
- Recursive bipartitioning of a circuit (e.g., K&L) leads to a min-cut placement.



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.
- **Slice/Bisection:** good for cells with high interconnection on the periphery.



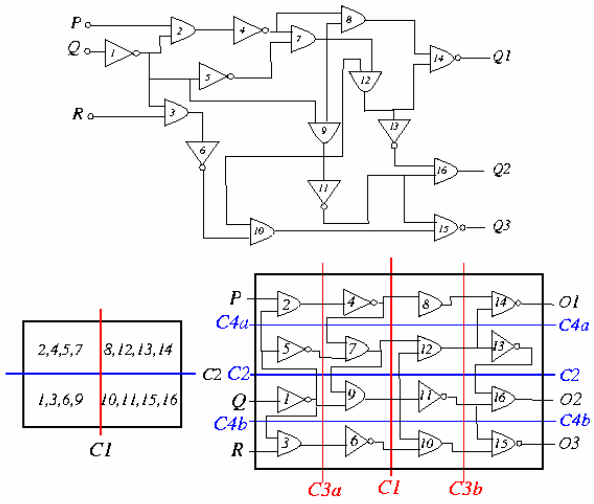
Algorithm for Min-Cut Placement

Algorithm: Min_Cut_Placement(*N*, *n*, *C*)

```
/* N: the layout surface */
/* n : # of cells to be placed */
/* n0: # of cells in a slot */
/* C: the connectivity matrix */

1 begin
2 if (n ≤ n0) then PlaceCells(N, n, C)
3 else
4     (N1, N2) ← CutSurface(N);
5     (n1, C1), (n2, C2) ← Partition(n, C);
6  Call Min_Cut_Placement(N1, n1, C1);
7  Call Min_Cut_Placement(N2, n2, C2);
8 end
```

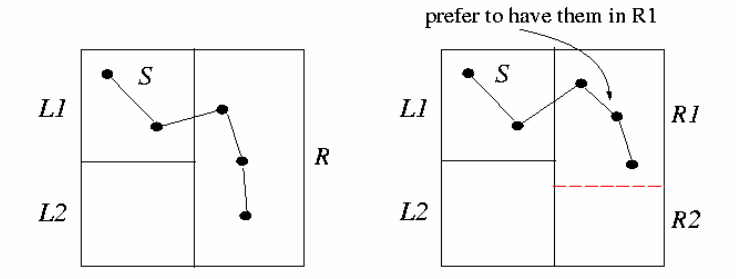
Quadrature Placement Example

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



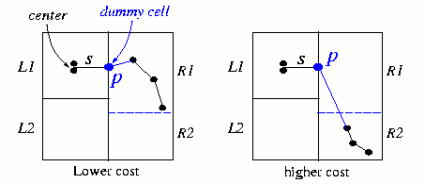
Min-Cut Placement with Terminal Propagation

- 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?

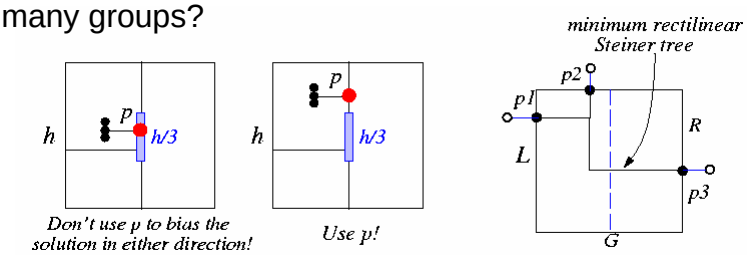


Terminal Propagation

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

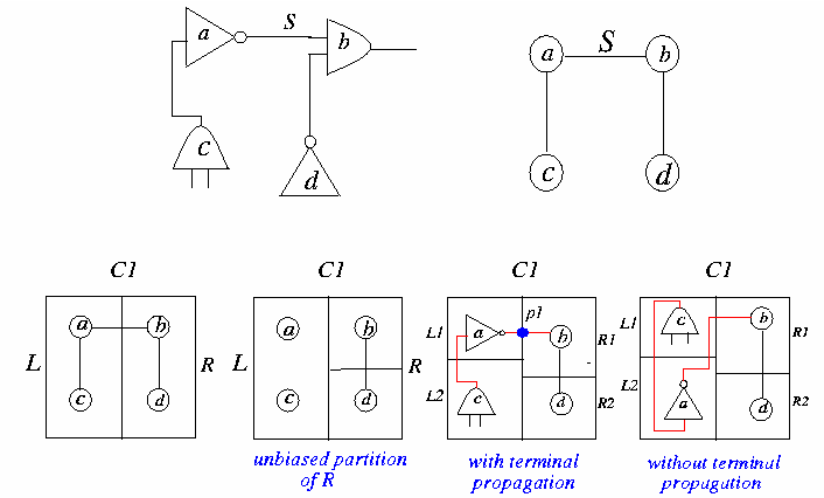


- When not to use *p* to bias partitioning? Net *s* has cells in many groups?



Terminal Propagation Example

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

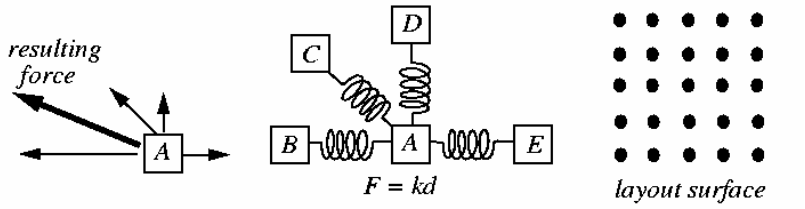


General Procedure for Iterative Improvement

```
Algorithm: Iterative_Improvement()
1  begin
2  s ← initial_configuration();
3  c ← cost(s);
4  while (not stop()) do
5      s' ← perturb(s);
6      c' ← cost(s');
7      if (accept(c, c'))
8      then   s ← s';
9  end
```


Placement by the Force-Directed Method

- Quinn, Jr. & Breuer, "A force directed component placement procedure for printed circuit boards," *IEEE Trans. Circuits and Systems*, June 1979.
- Reduce 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.



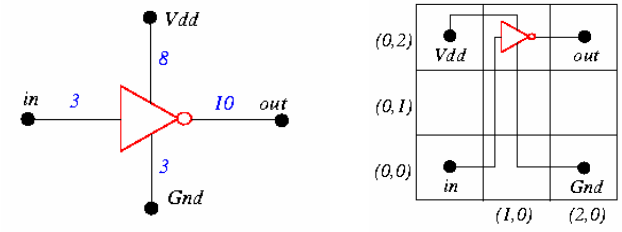
Finding the Zero-Force Target Location

- 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, $\hat{x}_i = \frac{8\times0+10\times2+3\times0+3\times2}{8+10+3+3} = 1.083$ and $\hat{y}_i = 1.50$.



Force-Directed Placement

- 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-force 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*.

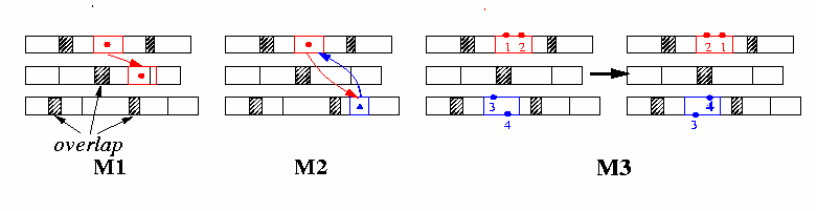
```
Algorithm: Force-Directed_Placement

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 ← 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 ← TRUE; AbortCount ← 0;
13            SAME AS PRESENT LOCATION:
14                EndRipple ← TRUE; AbortCount ← 0;
15            LOCKED:
16                Move selected cell to the nearest vacant location;
17                EndRipple ← TRUE; AbortCount ← AbortCount + 1;
18                if (AbortCount > AbortLimit) then
19                    Unlock all cell locations;
19                    IterationCount ← IterationCount + 1;
20            OCCUPIED AND NOT LOCKED:
21                Select cell as the target location for next move;
22                Move seed cell to target location and lock the target location;
23                EndRipple ← FALSE; AbortCount ← 0;
26 end
```


Placement by Simulated Annealing

- Sechen and Sangiovanni-Vincentelli, “The TimberWolf placement and routing package,” *IEEE J. Solid-State Circuits*, Feb. 1985.
- 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.

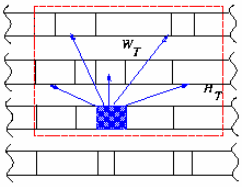
Solution Space & Neighborhood Structure

- **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.



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 big enough to contain the whole chip.
  - Window size shrinks as temperature decreases. Height & width $\propto$ $log(T)$.
  - Stage 2 begins when window size is so small that no inter-row module interchanges are possible.



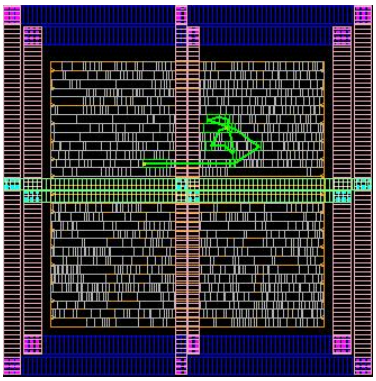
Cost Function

- 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. (α_i=1, β_i =1 $\Rightarrow$ half 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.
  - $C_2 = \gamma \sum_{i \neq j} O^2_{ij}$, 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.
  - $C_2 = \delta \sum_{r \in Rows} |L_r - D_r|$, δ : penalty weight.
  - D_r: desired row length.
  - L_r: sum of the widths of the modules in row *r*.

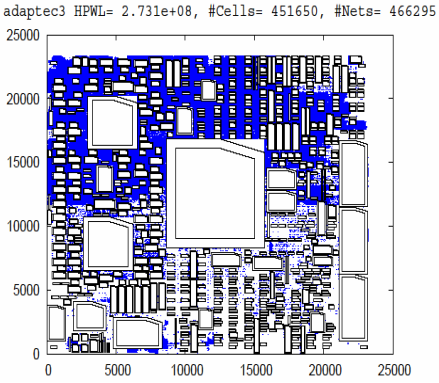
Annealing Schedule

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

Placement Examples



Cell placement with critical path



Mixed cell/macro placement (2 million cells + 500 macros)