**EE244: Design Technology for Integrated Circuits and Systems**

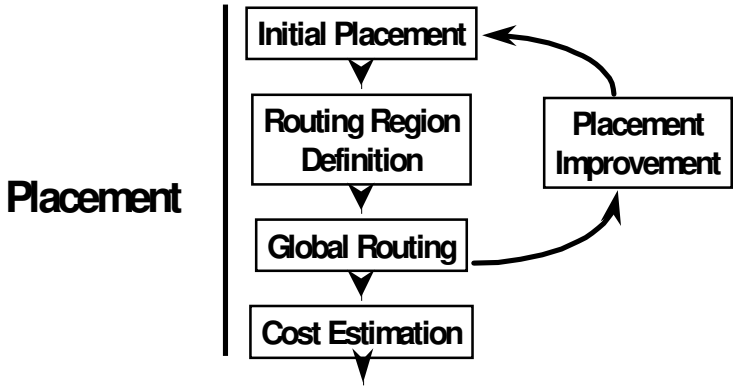
# Outline

**Lecture 3.1**

- **Upper bounds and Rent's Rule**
- **Wire length estimation**
- **Introduction to Placement**



# Physical Design: Overall Flow






Copyright © 1997, A. Richard Newton

# Rent's Rule

- In 1960, E.F.Rent published two IBM internal memoranda containing log plots of the "number of pins" vs the "number of circuits" in a logic design, which yielded:

$$N_p = K_p N_g^{\beta}$$

- Where $N_p$ =no. of external signal connections on a block, $N_g$= no. of logic gates in a block, $K_p$ is a proportionality constant, and $\beta$ is Rent's Constant.
- For IBM computers at that time, $K_p$ = 2.5 and $\beta$ = 0.6
- Later used to determine the average interconnect length in a gate-array as a function of the number of gates and the row spacing.



# Rent's Rule

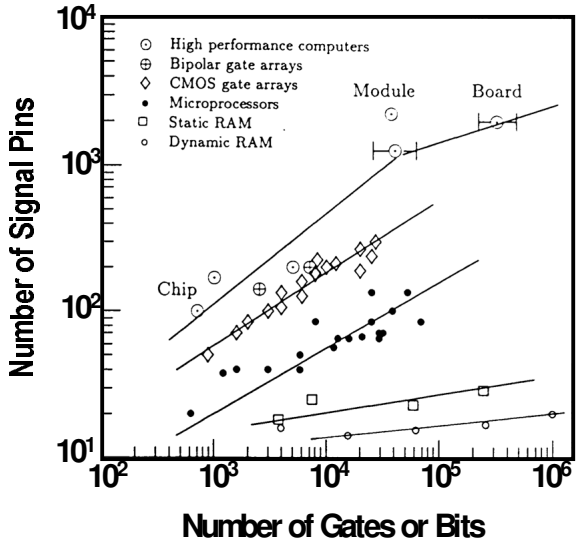
- An empirical result, very *circuit-fabric* (architecture) specific
- Multiplexed I/O, serial ports, bidirectional I/O
- Examples (from Bakoglu):

| System or Chip Type | Exponent $\beta$ | Multiplier $K_P$ |
|---|---|---|
| Static Memory | 0.12 | 6 |
| Microprocessor | 0.45 | 0.82 |
| Gate-Array | 0.50 | 1.9 |



Copyright © 1997, A. Richard Newton

# Rent's Curves for Chips & Systems





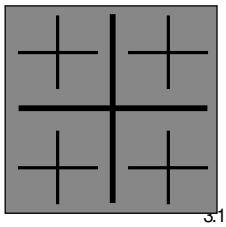

# Average Interconnection Length

- Can be approximated by partitioning the chip into hierarchical divisions and estimating the connections between partitions by Rent's Rule.
- Max. interconnect length between two divisions is the size of the division that is one hierarchy level above them.

$L_{ave} \propto N^{r-0.5}$ $r>0.5$

$L_{ave} \propto log(N)$ $r=0.5$

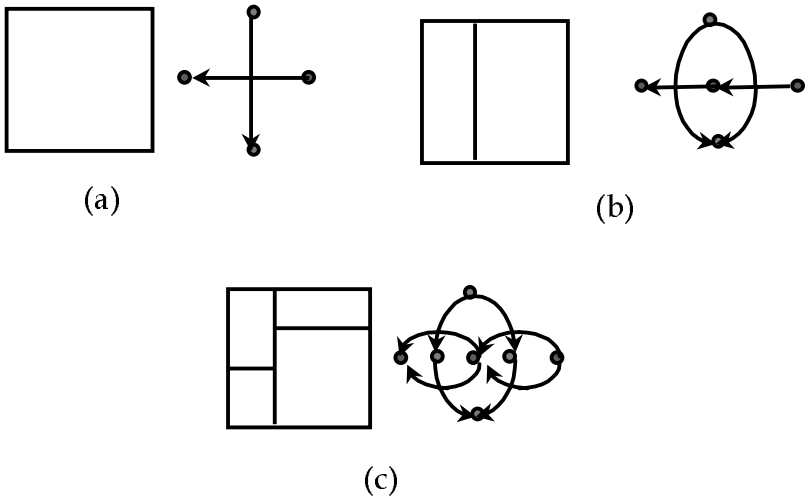
$L_{ave} \propto f(N)$ $r<0.5, f \text{ independent of } N$





Copyright © 1997, A. Richard Newton

# Initial Placement: Min-Cut






# Mincut Initial Placement

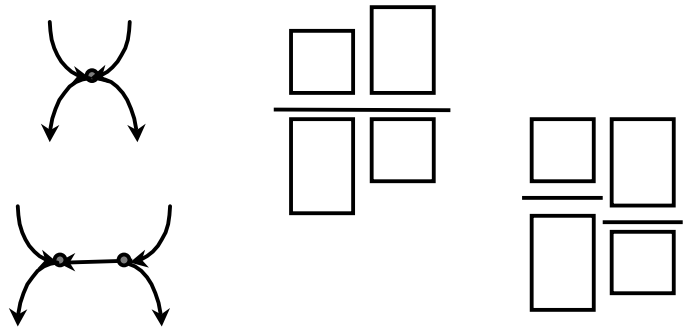
- Represent the collection of cells by an arrangement on non-overlapping rectangles.
- Represent the collection of rectangles by a pair of mutually-dual, polar (directed, acyclic, one source one sink) graphs
- $G_x = (V_x, E_x)$ and $G_y = (V_y, E_y)$
- $e_x^i$ represents a rectangle with $L(e_x^i) = L(e_y^i) = A$
- To start $L(e_x^i) \bullet L(e_y^i) = \sqrt{\sum a_x^i a_y^i}$
- Repeat the process recursively, using K&L to determine elements of each partition
- Replace $L(e_x^i), L(e_y^i)$ by actual $a_x^i, a_y^i$ (true length)
- Compute longest path from source to sink (both graphs)
- Allocate location (float non-critical cells)



Copyright © 1997, A. Richard Newton

# Mincut Initial Placement

- Apply rotation, mirror, and "squeeze"
- Iterate blocks to improve wiring congestion





# Placement Improvement

- Problem: Given an initial placement, is it possible to improve the placement in some sense by interchanging components?
  - How do we measure improvement?
  - Average, worst-case, weighted (e.g. critical paths), interconnect density

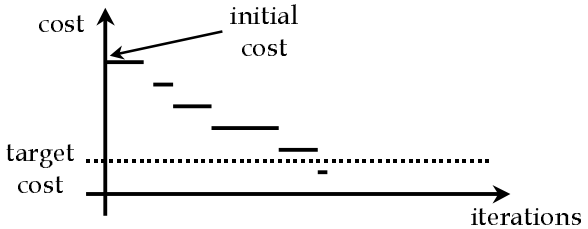


Copyright © 1997, A. Richard Newton

# 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

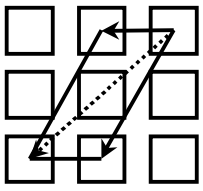





# Force-Directed Relaxation

- 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; }}}
```



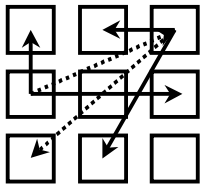


Copyright © 1997, A. Richard Newton

# Other Force-Directed Methods

- **Force-Directed Pairwise Relaxation**

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

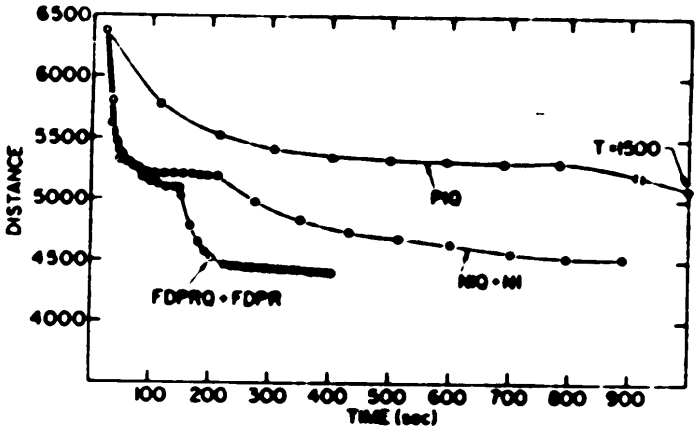


- **Force-Directed Pairwise Relaxation**

```
compute forces on all modules;
foreach( primary module, m) {
    compute target location;
    in an ε-neighborhood {
        compute module which if interchanged with m
          reduces overall force;
        swap modules;
    }
```



# Comparison of Interchange Strategies






Copyright © 1997, A. Richard Newton