

Chapter 7

Placement and Partitioning

VLSI Placement

Placement is the problem of automatically assigning correct positions on the chip to predesigned cells, such that some cost function is optimized.

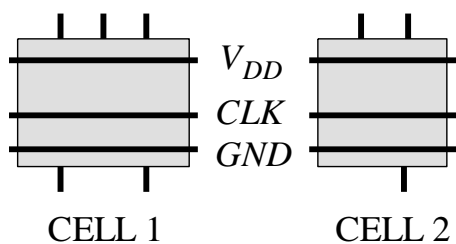
Different ways of designing create different placement problems. The most important of these are:

- Standard-cell placement;
- Building-block placement;
- A combination of the above.



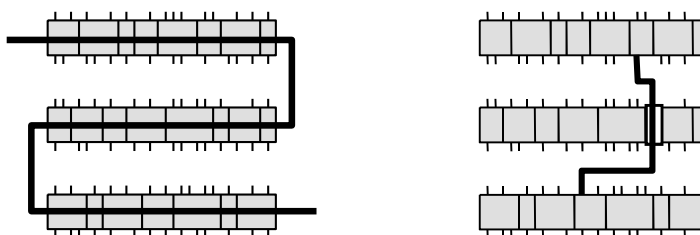
Standard-Cell Placement

- Standard cells have been designed in such a way that power and clock connections run horizontally through the cell and other I/O leaves the cell from top or bottom sides.



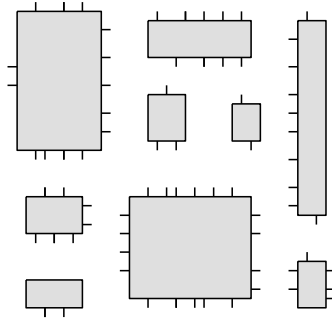
Standard-Cell Placement (Ctd.)

- The cells are placed in rows.
- Sometimes **feedthrough cells** are added to ease wiring.



Building-Block Placement

- The cells to be placed have arbitrary shapes.



Consequences of Fabrication Method

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

Cost Function

What does a placement algorithm try to optimize? E.g.

- the total area;
- the total wire length;
- the number of horizontal/vertical wire segments crossing a line.

Constraint:

- the placement should be routable (no cell overlaps; obeying wiring capacities).
- In some designs: timing constraints (some wires should always be shorter than a given length).

Relation with Routing

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

Wire Length Estimation

Example of **wire length metrics**:

- half of the perimeter of the rectangle enclosing all terminals in a net,
- minimum rectilinear spanning, } Using euclidean or
- minimum rectilinear Steiner tree, } manhattan distance
- squares of all pairwise terminal distances in a net using a **quadratic cost function**:

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



- typically used with clique model representation
- γ_{ij} is edge weight, can account for multiplicity of nets

Placement Algorithms

The placement problem is an NP-complete problem \Rightarrow heuristics are used in placement algorithms.

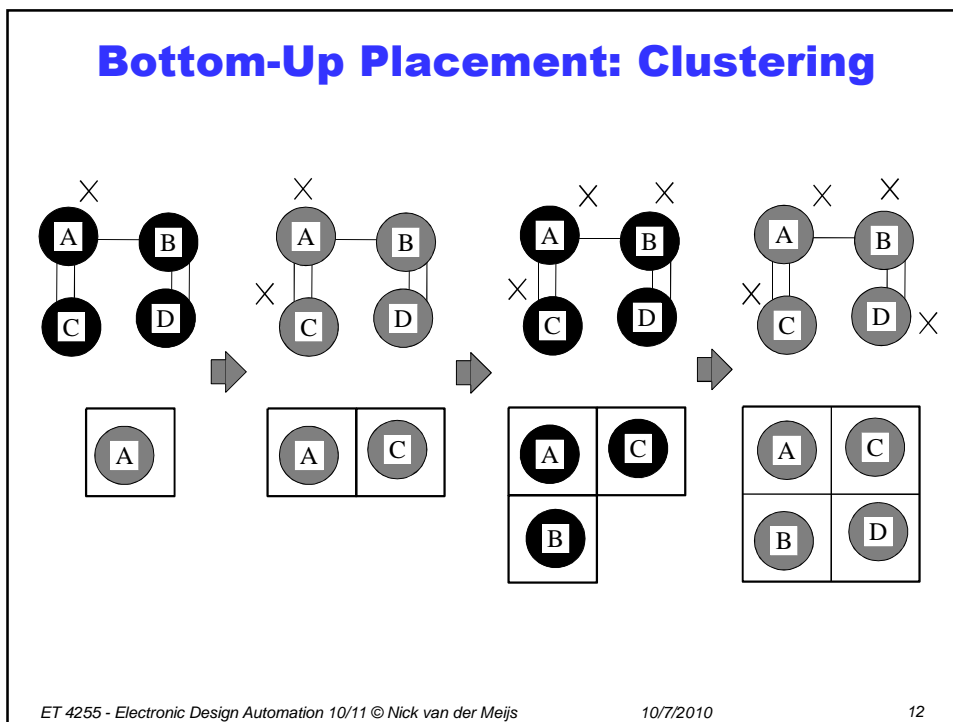
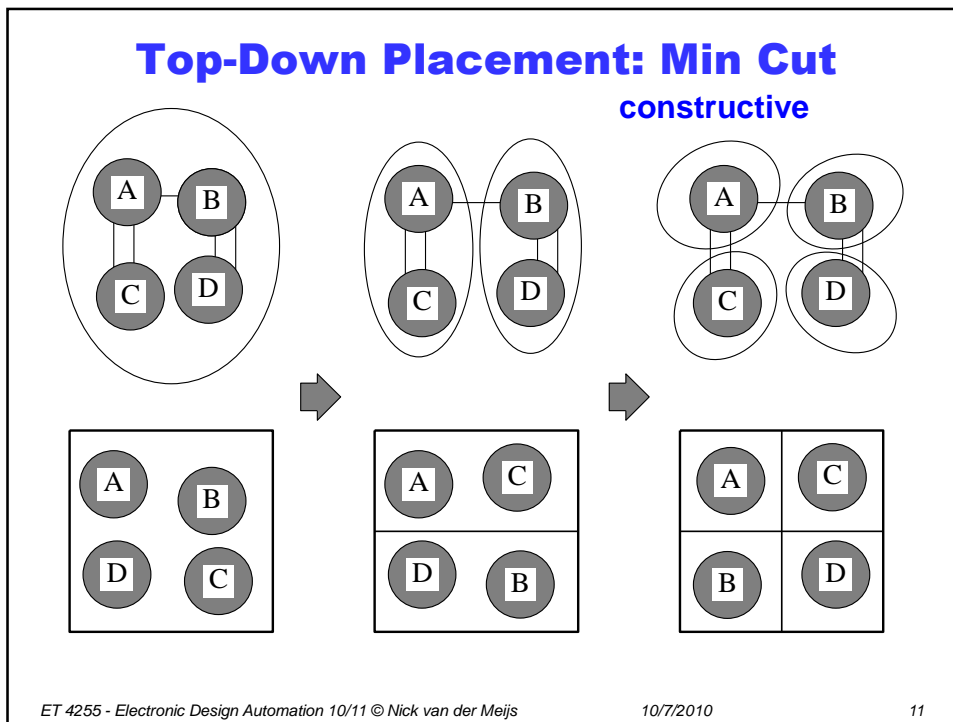
Classification:

- **Constructive** algorithms
once position of a cell is fixed, it is not modified anymore.
- **Iterative** algorithms
intermediate placements are modified in an attempt to improve the cost function.

Most approaches combine both elements:

- Constructive algorithms are used to obtain an **initial placement**.
- Initial placement followed by **iterative improvement**





General Procedure for Iterative Improvement

```

iterative_improvement()
{
  s ← initial_configuration();
  c ← cost(s);
  while (!stop()) {
    s' ← perturb(s);
    c' ← cost(s');
    if (accept(c, c'))
      s ← s';
  }
}

```

Force-Directed Iterative Improvement

- The “clique representation” of the netlist is used.
- For each cell i its **center of gravity** (x_{ig}, y_{ig}) , is computed using:

$$x_{ig} = \frac{\sum_{j=1}^n \gamma_{ij} x_j}{\sum_{j=1}^n \gamma_{ij}} \quad \text{and} \quad y_{ig} = \frac{\sum_{j=1}^n \gamma_{ij} y_j}{\sum_{j=1}^n \gamma_{ij}}.$$

- Cell i is exchanged with the cell located at its center of gravity, if this decreases the cost.

Placement Outlook

- Considerable academic activity on placement recently
- ISPD placement contests
- Appears large gap between typical placement solutions and optimal placements

Gap to known (upper) Bound

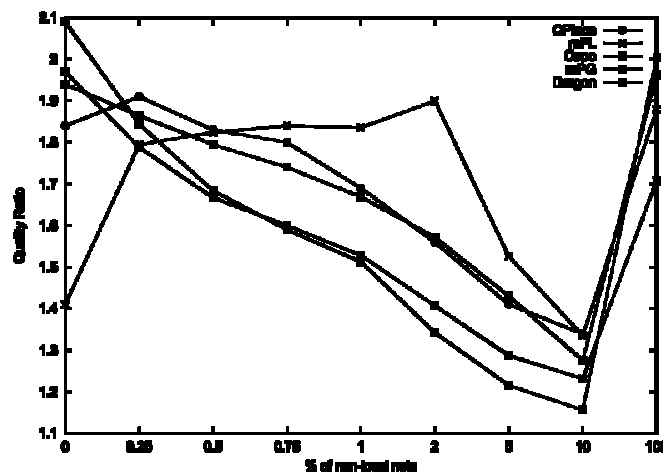


Fig. 1. Average solution quality vs percentage of non-local nets, from PEKO (0% non-local nets) through PEKU (0.25% to 10% of nonlocal nets) to G-PEKU (100% non-local nets). Each data point is an average quality ratio for a given placer over all circuits in the given suite.

[Cong, J. et al., ACM TOEAS, 2005, 389-430]

Chapter 7 (ctd)

Placement and Partitioning

ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

17

Partitioning

Decomposition of design, guided by cost function

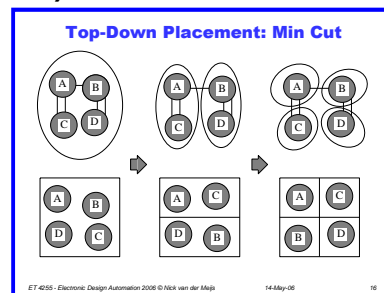


Applications

- mapping on available components
e.g. separate IC's or printboards
- dividing a synthesis problem into subproblems
Important if $C \gg O(N)$
- As a step in placement (min-cut)
- ...

Cost Function

- Min # of interconnections
between subcircuits
- size of subcircuits
- dissipation of subcircuits
- ...



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs

10/7/2010

18

Graph Formalism

- Circuit is represented by a graph

modules	↔	nodes
connections	↔	edges
connection count	↔	edge weight

- goal is to determine subgraphs, in such a way that:
 - \cup subgraphs = original graph
 - \cap subgraphs = \emptyset
 - cost function minimal

Set Formalism

modules	$M = \{m_1, m_2, \dots, m_n\}$
nets	$N = \{n_1, n_2, \dots, n_k\}$
every net	$n_j = \{m_{j1}, m_{j2}, \dots, m_{ji}\}$
cost matrix	$C_{ij} = \# \text{ of nets between } m_i \text{ and } m_j$
partitioning	$\Pi = \{\Pi_1, \Pi_2, \dots, \Pi_t\}$ $\Pi_i \subset M$ $\cup \Pi_i = M$ $\Pi_i \cap \Pi_j = \emptyset \quad \forall i \neq j$ $ \Pi_i \leq K_i \text{ (capacity)}$
Bi-Partition	$ \Pi = 2$

Optimization Problem

Minimize

$$C(\Pi) = \sum_{m_i \in \Pi_k, m_j \in \Pi_h, k \neq h} C_{ij}$$

Bi-Partition

$$C(\Pi) = \sum_{m_i \in \Pi_1, m_j \in \Pi_2} C_{ij}$$

These problems are NP-hard

So: heuristics

Constructive (Bi-)Partitioning

- Internal cost of module $m_i \in \Pi_1$

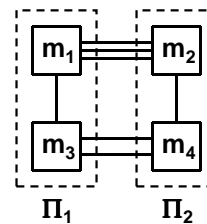
$$I(m_i) = \sum_{m_j \in \Pi_1} C_{ij}$$

- External cost of module $m_i \in \Pi_1$

$$E(m_i) = \sum_{m_j \notin \Pi_1} C_{ij}$$

- Cost function

$$C(m_i) = E(m_i) - I(m_i)$$



$$I(m_1)=1, E(m_1)=3, C(m_1)=2$$

Constructive (Bi-)Partitioning (2)

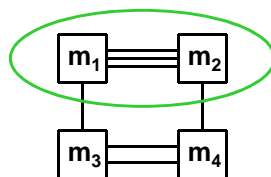
Algorithm

1. place first module (seed) in Π_1 , rest in Π_2 .
2. determine $C(m_i)$ for all modules in Π_2 .
3. add module with lowest C to Π_1 .
4. repeat step 2 and 3 until Π_1 is full.

Greedy

Constructive (Bi-)Partitioning (3)

Example



Initialization

1. place first module (seed) in Π_1 .
 $\Pi_1^{(1)} = \{m_1\}$

Iteration 1

2. determine $C(m_i)$ for all other modules.

module	I	E	C
m_2	1	3	-2
m_3	2	1	1
m_4	3	0	3

3. add module with the lowest C to Π_1 .
 $\Pi_1^{(2)} = \{m_1, m_2\}$
4. repeat step 2 and 3 until Π_1 is full.

Iterative Improvement

Repeatedly interchange modules between 2 partitions

Random Interchange

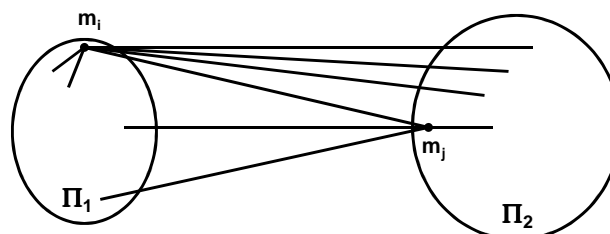
- choose arbitrary $m_i \in \Pi_1$ and $m_j \in \Pi_2$.
- accept interchange if cost function decreases

Kernighan-Lin Heuristic



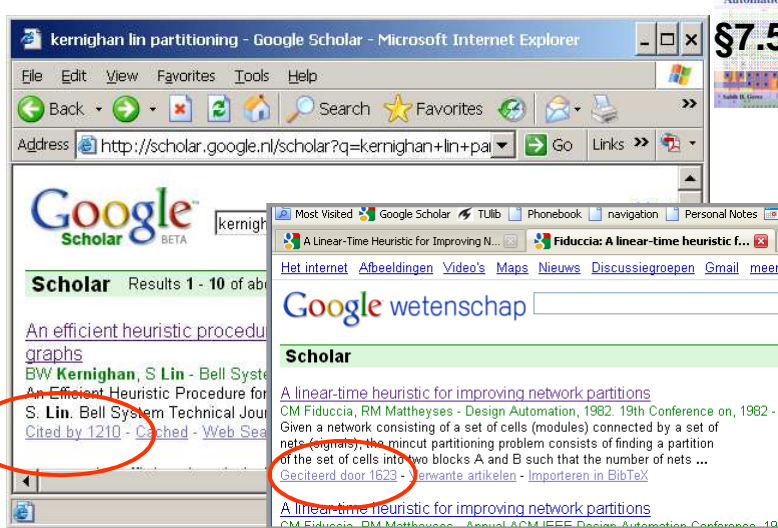
Given

- $m_i \in \Pi_1, m_j \in \Pi_2$
- score function $D(m_i, m_j)$
improvement in cost by interchanging m_i and m_j
 $D(m_i, m_j) = E(m_i) + E(m_j) - I(m_i) - I(m_j) - 2c_{ij}$



$$E(m_i)=4; E(m_j)=3; I(m_i)=2; I(m_j)=1; D(m_i, m_j)= 4+3-2-1-2 = 2$$

Kernighan-Lin Heuristic



Seems to be a pretty important paper

Algorithms for
**VLSI Design
Automation**

\$7.5.1

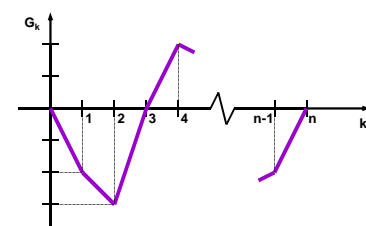
ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 27

Kernighan-Lin Heuristic (2)

Algorithm

1. Initially all modules are interchangeable
2. Determine for every pair of interchangeable modules (m_1, m_2) , $m_1 \in \Pi_1$, $m_2 \in \Pi_2$, the score $D(m_1, m_2)$.
3. Interchange the pair of modules that has max D in step 2. These modules are not interchangeable anymore. Calculate and retain the total cost of this partition.
4. Repeat steps 2 and 3 until all modules have been interchanged once.
5. The final result is the partition that has the lowest cost in step 3.

Repeat algorithm as long as there is an improvement.



ET 4255 - Electronic Design Automation 10/11 © Nick van der Meijs 10/7/2010 28

Evaluation

- The algorithm can be repeated until no more improvement is possible. The number of repetitions turns out not to depend on the problem size (3 or 4 times in general).
- Time complexity is $O(n^2 \log n)$
- None of these heuristics guarantees an optimal solution.
- Random interchange as well as Kernighan-Lin can get stuck in a local minimum.
- Needed: an algorithm with **uphill moves**.

Partitioning by means of Simulated Annealing

- Same moves and cost function as random interchange
- Can be successful
- Relatively long calculations
- Be careful with detailed parameters of algorithm