Chapter 7

Placement

Placement is a key step in physical design cycle. A poor placement consumes larger areas, and results in performance degradation. It generally leads to a difficult or sometimes impossible routing task. The input to the placement phase is a set of blocks, the number of terminals for each block and the netlist. If the layout of the circuit within a block has been completed then the dimensions of the block are also known. Placement phase is very crucial in overall physical design cycle. It is due to the fact, that an ill-placed layout cannot be improved by high quality routing. In other words, the overall quality of the layout, in terms of area and performance is mainly determined in the placement phase.

The placement of block occurs at three different levels.

1. **System level placement:** At system level, the placement problem is to place all the PCBs together so that the area occupied is minimum. At the same time, the heat generated by each of the PCBs should be dissipated properly so that the system does not malfunction due to overheating of some component.

2. **Board level placement:** At board level, all the chips on a board along with other solid state devices have to be placed within a fixed area of the PCB. All blocks are fixed and rectangular in shape. In addition, some blocks may be pre-placed. The PCB technology allows mounting of components on both sides. There is essentially no restriction on the number of routing layers in PCB. Therefore in general, the nets can always be routed irrespective of the quality of components placement. The objective of the board-level placement algorithms is twofold: minimization of the number of routing layers; and satisfaction of the system performance requirements. For high performance circuits, the critical nets should have lengths which are less than a specified value and hence the placement algorithms should place the critical components closer together. Another key placement problem is the temperature profile of the board. The heat dissipation on a PCB should be uniform, i.e., the chips which generate maximum heat should not be placed closer to each other. If MCMs are

used instead of PCBs, then the heat dissipation problem is even more critical, since chips are placed closer together on a MCM.

3. **Chip level placement:** At chip level, the problem can be either chip planning, placement or floorplanning along with pin assignment. The blocks are either flexible or fixed, and some of them may be pre-placed. The key difference between the board level placement problem and the chip level placement is the limited number of layers that can be used for routing in a chip. In addition, the circuit is fabricated only on one side of the substrate. This implies that some 'bad' placements maybe unroutable. However, the fact that a given placement is unroutable will not be discovered until routing is attempted. This leads to very costly delays in completion of the design. Therefore, it is very important to accurately determine the routing areas in the chip-level placement problems. Usually, two to four layers are used for routing, however, chips with four or more layers routings are more expensive to fabricate. The objective of a chip-level placement or floorplanning algorithm is to find a minimum area routable placement of the blocks. In some cases, a mixture of macro blocks and standard cells may have to be placed together. These problems are referred to as *Mixed block and cell* placement and floorplanning problems. At chip level, if the design is hierarchical then the placement and floorplanning is also carried out in a hierarchical manner. The hierarchical approach can greatly simplify the overall placement process.

In the following sections, we will discuss the chip-level placement. The placement problem for MCMs, which is essentially a performance driven Board level placement problem, will be discussed in Chapter 14.

In this chapter, we will discuss placement problems in different design styles. Section 7.1 discusses the problem formulation. Section 7.2 presents the classification of placement algorithms. Remaining sections present various algorithms for the placement problem.

7.1 Problem Formulation

The placement problem can be stated as follows: Given an electrical circuit consisting of fixed blocks, and a netlist interconnecting terminals on the periphery of these blocks and on the periphery of the circuit itself, construct a layout indicating the positions of each block such that all the nets can be routed and the total layout area is minimized. The objective for high performance systems is to minimize the total delay of the system, by minimizing the lengths of the critical paths. It is usually approximated by minimization of the length of the longest net. This problem is known as the *performance (timing) driven placement problem.* The associated algorithms are called *performance (timing) driven placement algorithms.*

The quality of a placement is based on several factors:

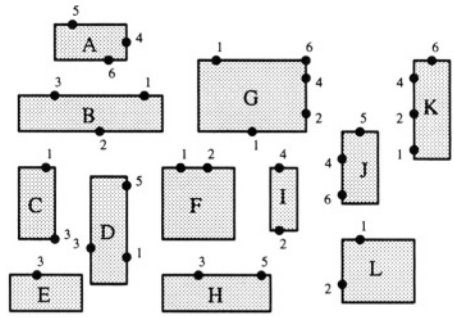

Figure 7.1: Blocks and set of interconnecting nets.

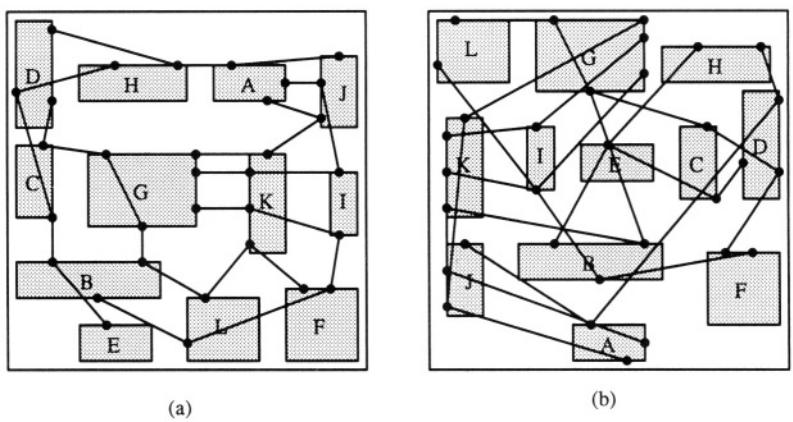

Figure 7.2: Two different placements of the same problem.

1. layout area.
2. completion of routing, and
3. circuit performance.

The layout area and the routability of the layout are usually approximated by the topological congestion, known as *rat's nest,* of interconnecting wires. Consider the simple example in Figure 7.1. Two different placements for this example are shown in Figure 7.2. The topological congestion in Figure 7.2(a) is much less than that of in Figure 7.2(b). Thus, the placement given in Figure 7.2(a) can be considered more easily routable than the placement given in Figure 7.2(b). In many cases, several objectives may contradict each other. For example, minimizing layout area may lead to increased maximum wire length and vice versa.

Let us formally state the placement problem. Let $B_1, B_2, \ldots, B_n$ be the blocks to be placed on the chip. Each $B_i, 1 \leq i \leq n$, has associated with it a height h_i and a width w_i. Let $\mathcal{N} = \{N_1, N_2, N_3, \ldots, N_m\}$ be the set of nets representing the interconnection between different blocks. Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_k\}$ represent rectangular empty areas allocated for routing between blocks. Let L_i denote the estimated length of net N_i, $1 \leq i \leq m$. The placement problem is to find iso-oriented rectangles for each of these blocks on the plane denoted by $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ such that

1. Each block can be placed in its corresponding rectangle, that is, R_i has width w_i and height h_i,
2. No two rectangles overlap, that is, $R_i \cap R_j = \phi$, $1 \leq i, j \leq n$,
3. Placement is routable, that is, $Q_j, 1 \leq j \leq k$, is sufficient to route all the nets.
4. The total area of the rectangle bounding $\mathcal{R}$ and $\mathcal{Q}$ is minimized.
5. The total wirelength is minimized, that is, $\sum_{i=1}^{m} L_i$ is minimized. In the case of high performance circuits, the length of longest net $\max\{L_i \mid i = 1, \ldots, m\}$ is minimized.

The general placement problem is NP-complete and hence, the algorithms used are generally heuristic in nature.

Although the actual wiring paths are not known at the time of placement, however, a placement algorithm needs to model the topology of the interconnection nets. An interconnection graph structure which interconnects each net is used for this purpose. The interconnection structure for two terminal trees is simply an edge between the two vertices corresponding to the terminals. In order to model a net with more than two terminals, rectilinear steiner trees are used as shown in Figure 7.3(a) to estimate optimal wiring paths for a net. This method is usually not used by routers, because of the NP-completeness of steiner tree problem. As a result, minimum spanning tree representations are the most commonly used structures to connect a net in the placement phase. Minimum spanning tree connections (shown in Figure 7.3(b)) allow branching only at the pin locations. Hence, the pins are connected in the form of minimum spanning tree of a graph. *Complete graph* interconnection is shown in (Figure 7.3(c)). It is easy to implement such structures. However, this method causes many redundant interconnections, and results in longer wire length.

The large number of objective functions can be classified into two categories, net metrics and congestion metric. The net metrics deal with the assumption that all the nets can be routed without interfering with other nets or with the components. Usually the length of a net is important as the interconnection delays depend on the length of the wire. The net metrics only quantify the amount of wiring and do not account for the actual location of these wires. The examples of this kind of objective functions are the total length of all nets and the length of the longest net. The congestion metric is used to avoid the

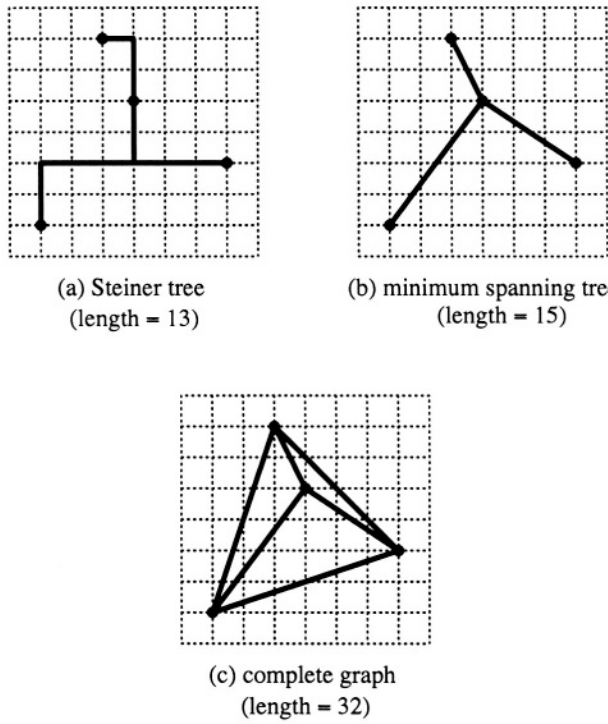

Figure 7.3: Interconnection topologies.

buildup of many nets in a particular area leading to congestion. Example of congestion metric is the number of nets that intersect with a routing channel.

The layout surface on which the circuit is to be placed is modeled into either geometric or topological models. For the geometric model, the placement algorithms tend to accept the layout area as a fixed constraint and tend to optimize the interconnections. The geometric models are appropriate for design styles where placement aspects such as size, shape and public pin positions do not change during the layout process such as PCB design. On the other hand, the placement systems which model the layout surface as a topological model assume the constraint to be the completion of interconnections and optimize the layout area. Topological models are appropriate for more flexible design styles such as full custom designs.

7.1.1 Design Style Specific Placement Problems

Different design styles impose different restrictions on the layout and have different objectives in placement problems.

1. **Full custom:** In full custom design style, the placement problem is the packing problem concerned with placing a number of blocks of different sizes and shapes tightly within a rectangular area. There is no restriction on how the blocks can be placed within the rectangle except that no two blocks may overlap. The primary objective is to minimize the total layout area. The irregularity of the block shapes is usually the main cause of unused areas. Since unused area increases the total area, the blocks must be placed so as to minimize unused areas. The objective of minimizing the layout area sometimes conflicts with the objective of minimizing the maximum length of a net. Therefore, in high performance circuit design, additional constraints on net lengths must also be considered.

2. **Standard cells:** The standard cell placement problem is somewhat simpler than the full custom placement problem, as all the cells have the same height. Cells are placed in rows and minimizing layout area is equivalent to minimizing the summation of channel heights and minimizing the width of the widest row. In order to reduce overall area, all rows should have equal widths. The total area consists of area required for the cell rows and the area required for routing or the channel area. The routing area estimates, which determine the channel height, play a key role in determining the overall area of the design. With the advent of over-the-cell routing, in which the empty spaces over the standard cell rows are used for routing, the channels in standard cells have almost disappeared giving rise to *channelless* standard cell designs. Standard cells are designed so the power and ground nets run horizontally through the top and bottom of cells.

3. **Gate arrays:** As mentioned in the previous chapter, in case of gate arrays, the partitioning of a circuit maps the circuit onto the gates of the gate array. Hence the problem of partitioning and placement is essentially the same in this design style. If partitioning does not actually assign gate locations, then a placement algorithm has to be used to assign subcircuits or gates to the slots on the gate array. Given a set of blocks $B_1, B_2, \ldots, B_n$, and set of slots $S_1, S_2, \ldots, S_r$, $r \geq n$, assign each block B_i to a slot S_j such that no two blocks are assigned to the same slot and the placement is routable. For high performance designs, additional constraints on net lengths have to be added.

   Another situation, where gate array partitioning and placement may be different, is when each 'gate' in the gate array is a complex circuit. In this case, the circuit is partitioned such that each subcircuit is equivalent to a 'gate'. The placement algorithm is then used to find the actual assignment. This happens to be the case in FPGAs and is discussed in Chapter 13.

7.2 Classification of Placement Algorithms

The placement algorithms can be classified on the basis of :

1. the input to the algorithms,
2. the nature of output generated by the algorithms, and
3. the process used by the algorithms.

Depending on the input, the placement algorithms can be classified into two major groups: *constructive placement* and *iterative improvement* methods. The input to the constructive placement algorithms consists of a set of blocks along with the netlist. The algorithm finds the locations of blocks. On the other hand iterative improvement algorithms start with an initial placement. These algorithms modify the initial placement in search of a better placement. These algorithms are typically used in an iterative manner until no improvement is possible.

The nature of output produced by an algorithm is another way of classifying the placement algorithms. Some algorithms generate the same solution when presented with the same problem, i.e., the solution produced is repeatable. These algorithms are called *deterministic* placement algorithms. Algorithms that function on the basis of fixed connectivity rules (or formulae) or determine the placement by solving simultaneous equations are deterministic and always produce the same result for a particular placement problem. Some algorithms, on the other hand, work by randomly examining configurations and may produce a different result each time they are presented with the same problem. Such algorithms are called as *probabilistic* placement algorithms.

The classification based on the process used by the placement algorithms is perhaps the best way of classifying these algorithms. There are two important class of algorithms under this classification: simulation based algorithms and partitioning based algorithms. Simulation based algorithms simulate some natural phenomenon while partitioning based algorithms use partitioning for generating the placement. The algorithms which use clustering and other approaches are classified under 'other' placement algorithms.

7.3 Simulation Based Placement Algorithms

There are many problems in the natural world which resemble placement and packaging problems. Molecules and atoms arrange themselves in crystals, such that these crystals have minimum size and no residual strain. Herds of animals move around, until each herd has enough space and it can maintain its predator-prey relationships with other animals in other herds. The simulation based placement algorithms simulate some of such natural processes or phenomena. There are three major algorithms in this class: simulated annealing, simulated evolution and force directed placement. The simulated annealing algorithm simulates the annealing process which is used to temper metals. Simulated

evolution simulates the biological process of evolution while the force directed placement simulates a system of bodies attached by springs. These algorithms are described in the following subsections.

7.3.1 Simulated Annealing

Simulated annealing is one of the most well developed placement methods available [BJ86, GS84, Gro87, Haj88, HRSV86, LD88, Oht86, RSV85, RVS84, SL87, SSV85]. The simulated annealing technique has been successfully used in many phases of VLSI physical design, e.g., circuit partitioning. The detailed description of the application of simulated annealing method to partitioning may be found in Chapter 5. Simulated annealing is used in placement as an iterative improvement algorithm. Given a placement configuration, a change to that configuration is made by moving a component or interchanging locations of two components. In case of the simple pairwise interchange algorithm, it is possible that a configuration achieved has a cost higher than that of the optimum but no interchange can cause a further cost reduction. In such a situation the algorithm is trapped at a local optimum and cannot proceed further. Actually this happens quite often when this algorithm is used on real life examples. Simulated annealing avoids getting stuck at a local optimum by occasionally accepting moves that result in a cost increase.

In simulated annealing, all moves that result in a decrease in cost are accepted. Moves that result in an increase in cost are accepted with a probability that decreases over the iterations. The analogy to the actual annealing process is heightened with the use of a parameter called *temperature T*. This parameter controls the probability of accepting moves which result in an increased cost. More of such moves are accepted at higher values of temperature than at lower values. The acceptance probability can be given by $e^{\frac{-\Delta C}{T}}$, where ΔC is the increase in cost. The algorithm starts with a very high value of temperature which gradually decreases so that moves that increase cost have lower probability of being accepted. Finally, the temperature reduces to a very low value which causes only moves that reduce cost to be accepted. In this way, the algorithm converges to a optimal or near optimal configuration.

In each stage, the configuration is shuffled randomly to get a new configuration. This random shuffling could be achieved by displacing a block to a random location, an interchange of two blocks, or any other move which can change the wire length. After the shuffle, the change in cost is evaluated. If there is a decrease in cost, the configuration is accepted, otherwise, the new configuration is accepted with a probability that depends on the temperature. The temperature is then lowered using some function which, for example, could be exponential in nature. The process is stopped when the temperature has dropped to a certain level. The outline of the simulated annealing algorithm is shown in Figure 7.4.

The parameters and functions used in a simulated annealing algorithm determine the quality of the placement produced. These parameters and functions include the cooling schedule consisting of initial temperature (*init-temp*),

```
Algorithm SIMULATED-ANNEALING
begin
    temp = INIT-TEMP;
    place = INIT-PLACEMENT;
    while (temp > FINAL-TEMP) do
        while (inner_loop_criterion = FALSE) do
            new_place = PERTURB(place);
            ΔC = COST(new_place) - COST(place);
            if (ΔC < 0) then
                place = new_place;
            else if (RANDOM(0, 1) > e^(ΔC/temp)) then
                place = new_place;
        temp = SCHEDULE(temp);
end.
```

Figure 7.4: The Simulated Annealing algorithm.

final temperature (*final_temp*) and the function used for changing the temperature (SCHEDULE), *inner_loop_criterion* which is the number of trials at each temperature, the process used for shuffling a configuration (PERTURB), acceptance probability (F), and the cost function (COST). A good choice of these parameters and functions can result in a good placement in a relatively short time.

Sechen and Sangiovanni-Vincentelli developed TimberWolf 3.2, which is a standard cell placement algorithm based on Simulated Annealing [SSV85]. TimberWolf is one of the most successful placement algorithms. In this algorithm, the parameters and functions are taken as follows. For the cooling schedule, *init_temp* = 4000000, *final-temp* = 0.1, and SCHEDULE $(T) = \alpha(T) \times T$ where $\alpha(T)$ is a cooling rate depending on the current temperature T. $\alpha(T)$ is taken relatively low when T is high, e.g. $\alpha(T) = 0.8$ when the cooling process just starts, which means the temperature is decremented rapidly. Then, in the medium range of temperature, $\alpha(T)$ is taken 0.95, which means that the temperature changes more slowly. When the temperature is in low range, $\alpha(T)$ is again taken 0.8, the cooling procedure go fast again. In this way, there are a total of 117 temperature steps. The graph for the cooling schedule is shown in Figure 7.5. The value of *inner_loop_criterion* is taken according to the size of the circuit, e.g., 100 moves per cell for a 200-cell circuit and 700 moves per cell for a 3000-cell circuit are recommended in [SSV85]. The new configuration is generated by making a weighted random selection from one of the following:

1. the displacement of a block to a new location,
2. the interchange of locations between two blocks,
3. an orientation change for a block.

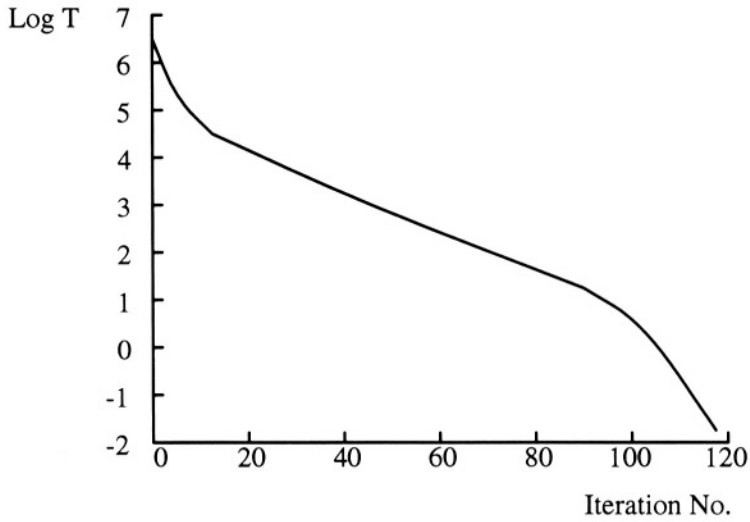

Figure 7.5: Cooling schedule in TimberWolf.

The alternative 3 is used only when the new configuration generated by using alternative 1 is rejected. The ratio r of single block displacement to pairwise interchange should be carefully chosen to give a best overall result. An orientation change of a block is simply a mirror image of that block's x-coordinate. The cost function is taken as:

$$COST = cost1 + cost2 + cost3$$

where $cost1$ is the weighted sum of estimate length of all nets, $cost2$ is the penalty cost for overlapping, and $cost3$ is the penalty cost for uneven length among standard cell rows.

$$cost1 = \sum_{i \in nets} [xspan(i) \times HWeight(i) + yspan(i) \times VWeight(i)]$$
$$cost2 = \sum_{i \neq j \in blocks} [overlap(i,j)]^2$$
$$cost3 = \sum_{i \in rows} |ActRowLength(i) - DesRowLength(i)| \times factor$$

Where, $xspan(i)$ and $yspan(i)$ are the horizontal and vertical spans of the minimum bounding rectangle of net i. Horizontal and vertical weights (*HW eight* and *VW eight*) are introduced so that each net can have different priority to be optimized, e.g. critical nets can have higher priority, and one direction can be favored over another direction. The quadratic function in $cost2$ is used to penalize more heavily on large overlaps than small ones. Actually overlap is not allowed in the placement. However, it takes large amount of computer time to remove all overlapping. So, it is more efficient to allow overlapping during intermediate placement and use a cost function to penalize the overlapping.

$ActRowLength(i)$ and $DesRowLength(i)$ are the actual row length and desired row length for the ith row, respectively. The *factor* is used so that the minimum penalty for the difference in length of rows is *factor*.

The simulated annealing is one of the most established algorithms for placement problems. It produces good quality placement. However, Simulated Annealing is computationally expensive and can lead to longer run times. Therefore, it is only suitable for small to medium sized circuits.

7.3.2 Simulated Evolution

Simulated evolution (genetic algorithm) is analogous to the natural process of mutation of species as they evolve to better adapt to their environment. It has been recently applied to various fields. Readers are referred to the chapter 5 for the description of simulated evolution algorithm used in partitioning.

We use the example of gate array placement problem to explain the simulated evolution algorithm. In gate array placement problem, the layout plane is divided into $S = \{S_1, S_2, \ldots, S_r\}$ slots. The problem of placing cells $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$, where $n \leq r$, is to assign some S_j to each B_i, such that no two cells are assigned to the same slot. The algorithm starts with an initial set of placement configurations, which is called the *population*. This initial placement can be generated randomly. The individuals in this population represent a feasible placement to the optimization problem and are actually represented by a string of symbols. The symbols used in the solution string are called *genes*. A solution string made up of genes is called a *chromosome*. A *schema* is a set of genes that make up a partial solution. Simulated evolution algorithm is iterative, and each iteration is called a *generation*. During each iteration the individuals of the population are evaluated on the basis of certain *fitness* tests which can determine the quality of each placement. Two individuals (corresponding to two possible placement configurations) among the population are selected as *parents* with probabilities based on their fitness. The better fitness an individual has, the higher the probability that it will be chosen. The operators called crossover, mutation and inversion, which are analogous to the counterparts in the evolution process, are then applied on the parents to combine 'genes' from each parent to generate a new individual called the *offspring*. The offsprings are then evaluated and a new generation is then formed by including some of the parents and the offsprings on the basis of their fitness in a manner that the size of population remains the same. As the tendency is to select high fitness individuals to generate offsprings and the weak individuals are deleted, the next generation tends to have individuals that have good fitness. The fitness of the entire population improves over the generations. That means the overall placement quality improves over iterations. At the same time, some 'bad' genes are inherited from previous generation even though the probability of doing so is quite low. In this way, it is assured that the algorithm does not get stuck at some local optimum. This is the basic mechanism of the algorithm which results in a good placement. The three genetic operators that are used for creating offsprings are discussed below.

1. **Crossover :** Crossover generates offsprings by combining schemata of two individuals at a time. This could be achieved by choosing a random cut point and generating the offspring by combining the left segment of one parent with the right segment of the other. However, after doing so, some blocks may be repeated while some other blocks may get deleted. This problem has been dealt with in many different ways. The amount of crossover is controlled by the crossover rate which is defined as the ratio of the number of offspring produced in each generation to the population size. The crossover rate determines the ratio of the number of searches in regions of high average fitness to the number of searches in other regions.

2. **Mutation:** This operator is not directly responsible for producing new offsprings but it causes incremental random changes in the offspring produced by crossover. The most commonly used mutation is pair-wise interchange. This is the process by which new genes which did not exist in the original generation can be generated. The mutation rate is defined as the percentage of the total number of genes in the population, which are mutated in each generation. It should be carefully chosen so that it can introduce more useful genes, and at the same time do not destroy the resemblance of *offsprings* to their *parents.*

3. **Selection:** After the offspring is generated, individuals for the next generation are chosen based on some criteria. There are many such selection functions used by various researchers. In *competitive selection* all the parents and offsprings compete with each other and the fittest individuals are selected so that the population remains constant. In *random selection* the individuals for the next generation are randomly selected so that the population remains constant. This could be advantageous considering the fact that by selecting the fittest individuals the population converges to individuals that share the same genes and the search might not converge to a optimum. However, if the individuals are chosen randomly, there is no way to gain improvements from older generation to new generation. By compromising both methods, *stochastic selection* makes selections with probabilities based on the fitness of each individual.

An algorithm developed by Cohoon and Paris [CP86] is shown in Figure 7.6. The scoring function is chosen to account for total net lengths and to penalize the placement with high wiring density in the routing channels. The score is given by:

$$\sigma = \frac{1}{2} \sum_{i \in nets} length(i) + \sum_{i \in HChannels} h_i'^2 + \sum_{i \in VChannels} v_i'^2$$

where

$$h_i' = \begin{cases} h_i - h_{avg} - h_{sd} & \text{if } h_i > h_{avg} - h_{sd} \\ 0 & \text{otherwise} \end{cases}$$

```
Algorithm GENIE
begin
    no_pop = SIZE-POP;
    no_offspring = no_pop × P_ψ;
    (* P_ψ stands for the crossover rate. *)
    pop = CONSTRUCT-POP(no_pop);
    for (i = 1 to no_pop) do
    (* Evaluate score of each individual in
       the population on the basis of its fitness. *)
        SCORE(pop(i));
    for (i = 1 to no_generation) do
    (* Generate no_generation generations *)
        for (j = 1 to no_offspring) do
            (x, y) = CHOOSE-PARENT(pop)
            offspring(j) = GENERATE(x, y);
            SCORE(offspring(j));
        pop = SELECT(pop, offspring, no_pop);
        for (j = 1 to no_pop) do
            MUTATE(pop(j));
    return highest scoring configuration in population;
end.
```

Figure 7.6: The Simulated Evolution algorithm.

$$v_i' = \begin{cases} v_i - v_{avg} - v_{sd} & \text{if } v_i > v_{avg} - v_{sd} \\ 0 & \text{otherwise} \end{cases}$$

where, h_i (v_i) is the number of nets intersecting horizontal (vertical) channel i. h_{avg} (v_{avg}) is the mean of h_i (v_i). h_{sd} (v_{sd}) is the standard deviation of h_i (v_i).

The parent choosing function is performed alternatively as either selecting parents with probabilities proportional to their fitness or selecting parents with probabilities proportional to their fitness and an additional constraint such that they have above average fitness. Two crossover operators can be used. One selects a random cell C_s and brings the four closest neighbors in parent 1 into neighboring slots in parent 2. At the same time, the cells in these slots in parent 2 are pushed outward until vacant slots are found. The other one selects a square of $k \times k$ cells from parent 1 where k is a random number with mean of 3 and variance of 1, and copy the square into parent 2. The result of this copying would result in the loss of some cells. So, before copying, the cells in parent 2 that are not part of square are being pushed outward into some vacant slots.

One possible mutation function is to use a greedy technique to improve the

placement cost. It selects a cell C_i on a net N_j and searches the cell C_k on the same net that is farthest from cell C_i. C_k is then brought close to the cell C_i. The cell which needs to be removed from that slot is pushed outward until a vacancy is found.

Besides the implementation described above, there are other implementations, e.g. the genetic approach developed by Chan, Mazumder and Shahookar, which uses a two-dimensional bitmap chromosome to handle the placement of macro cells and gate arrays [CMS91]. In addition, the simulated evolution was investigated in [CM89, KB87, Kli87, SM90a, SM90b].

7.3.3 Force Directed Placement

Force directed placement explores the similarity between placement problem and classical mechanics problem of a system of bodies attached to springs. In this method, the blocks connected to each other by nets are supposed to exert attractive forces on each other. The magnitude of this force is directly proportional to the distance between the blocks. According to Hooke's law, the force exerted due to stretching of the springs is proportional to the distance between the bodies connected to the spring. If the bodies were allowed to move freely, they would move in the direction of the force until the system achieved equilibrium. The same idea is used for placing the blocks. The final configuration of the placement of blocks is the one in which the system achieves equilibrium.

In [Qui75], Quinn developed a placement algorithm using force directed method. In this algorithm, all the blocks to be placed are considered to be rectangles. These blocks are classified as movable or fixed. Let $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ be the blocks to be placed, and (x_i, y_i) be the Cartesian coordinates for B_i. Let $\Delta x_{ij} = |x_i - x_j|$, $\Delta y_{ij} = |y_i - y_j|$. $\Delta d_{ij} = \sqrt{(\Delta x_{ij})^2 + (\Delta y_{ij})^2}$. Let F_x^i (F_y^i) be the total force enacted upon B_i by all the other blocks in the x-direction (y-direction). Then, the force equations can be expressed as:

$$F_x^i = \sum_{j=1}^{n} [-k_{ij} \times \Delta x_{ij}]$$
$$F_y^i = \sum_{j=1}^{n} [-k_{ij} \times \Delta y_{ij}]$$

where $i = 1, 2, \ldots, n$, k_{ij} = attractive constant between blocks B_i and B_j and $k_{ij} = 0$ if $i = j$. The blocks connected by nets tend to move toward each other, and the force between them is directly proportional to the distance between them. On the other hand, the force model does not reflect the relationship between unconnected blocks. In fact, the unconnected blocks tend to repel each other. So, the above model should be modified to include these repulsion effects. Since the formulation of the force equation in x-direction is the same as in y-direction, in the following, only the formulation in x-direction will be discussed.

$$F_x^i = \sum_{j=1}^{n} [-k_{ij} \times \Delta x_{ij} + \delta k_{ij} \times R \times \Delta x_{ij} / \Delta d_{ij}], \quad i = 1, 2, \ldots, n$$

where $\delta k_{ij} = 1$ when $k_{ij} = 0$, and $\delta k_{ij} = 0$ when $k_{ij} = 1$. R is the repulsion constant directly proportional to the maximum of k_{ij} and inversely proportional to n.

In addition, it is also desirable to locate the center of all movable blocks in some predetermined physical location (usually the geometric center of the layout plane) so that the placement of blocks is balanced. Physically, it is equivalent to have the forces acted upon the set of all movable blocks being removed. Suppose there are m movable blocks. Then, the force equations become:

$$F_x^i = \sum_{j=1}^{n} [-k_{ij} \times \Delta x_{ij} + \delta k_{ij} \times R \times \Delta x_{ij}/\Delta d_{ij}] - F_{ext}, \quad i = 1, 2, \ldots, n$$

where F_{ext} is the total external force acted upon the set of all movable blocks by the fixed blocks and $F_{ext} = \{\sum_{i=1}^{m} \sum_{j=m+1}^{n} [-k_{ij} \times \Delta x_{ij} + \delta k_{ij} \times R \times \Delta x_{ij}/\Delta d_{ij}]\}/m$.

The placement problem now becomes a problem in classical mechanics and the variety of methods used in classical mechanics can be applied. To solve for the set of force equations, one of the methods is to set the potential energy equal to $\sum_{i=1}^{n} [F_x^{i\,2} + F_y^{i\,2}]$, and apply the unconstrained minimization method, i.e., Fletcher-Reeves method [FR64], since the solution of the force equations correspond to the state of zero potential energy of the system.

Besides the implementation presented by Quinn [Qui75], there are various implementations [AJK82, Got81, HK72, HWA78, Oht86, QB79].

7.3.4 Sequence-Pair Technique

A packing of set of rectangles is nothing but non-overlapping placement of rectangles. Sequence-pair is a representation of such a packing in terms of a pair of module name sequences. In algorithms like simulated annealing solution space is infinite and thus the algorithms stops the search for optimal solution half-way and outputs the result. A finite solution space which includes an optimal solution is the key for successful optimization. Sequence-pair technique generates such a finite solution space. Murata et. al. proved in [MFNK96] that searching the solution space generated by sequence-pair technique using simulated annealing placement algorithm where move is change of the sequence-pair, gives efficient rectangular packing.

A procedure called **Gridding** is used to encode a placement on a chip to a sequence-pair. Let **P** be a packing of m modules on chip **C**. In *gridding* procedure m non-intersecting,non-overlapping lines (lines doesn't cross boundaries of modules also) are drawn from south-west corner to north-east corner of the chip and each line passes through one module diagonally. These lines can be linearly ordered and this is S_1 of sequence-pair (S_1, S_2). Second order S_2 of sequence-pair can be obtained by drawing similar kind of lines from south-east corner of the chip to north-west corner of the chip.

Given a sequence-pair (S_1, S_2) one of the optimal solution under the constraint can be obtained in $O(m^2)$ time by applying the longet path algorithm

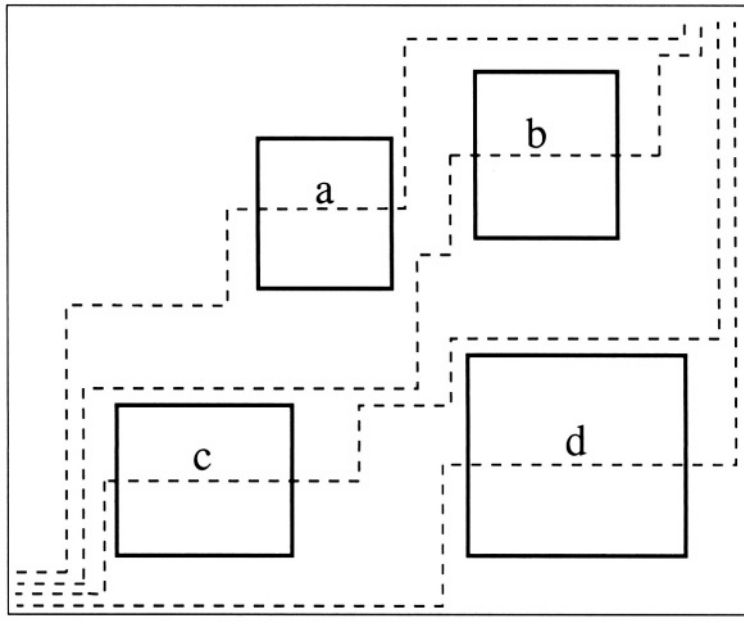

Figure 7.7: Sequence pair for the given placement is a, b, c, d.

for vertex weighted directed acyclic graphs. A relation (*LeftOf, RightOf, BelowOf, AboveOf*) between a pair of modules can be determined based on the location of modules relative to each other. Prom the given sequence-pair (S_1, S_2) it is easy to generate such relations between modules of the chip. If *leftof* $(m_1) = (m_2)$ means m_2 is left side of m_1. m_2 will be left side of m_1 if m_2 is before m_1 in both the orders of sequence-pair.

In Figure 7.7 sequence-pair for the given placement is *abcd, cdab.*
LeftOf(a) = () , Modules that are after *a* in both the orders of sequence-pair.
RightOf(a) = (b),Modules that are before *a* in both the orders of sequence-pair.
AboveOf (a) = (), Modules that are before *a* in first order and after 'a' in second order of sequence-pair.
BelowOf(a) = (c,d), Modules that are after *a* in first order and before 'a' in second order of sequence-pair.

In Figure 7.8 horizontal and vertical constraint graphs are generated for the sequence-pair *abcd, cdab.*

A directed and vertex-weighted graph called "horizontal-constraint graph" G_h, can be constructed using modules as verticies, module widths as weight of vetices and *leftof* relation as edges of graph. Similarly using "below" relation and height of the block vertical-constraint graph G_v, can be generated. For both the graphs source and sink vertices are out side the chip boundary with

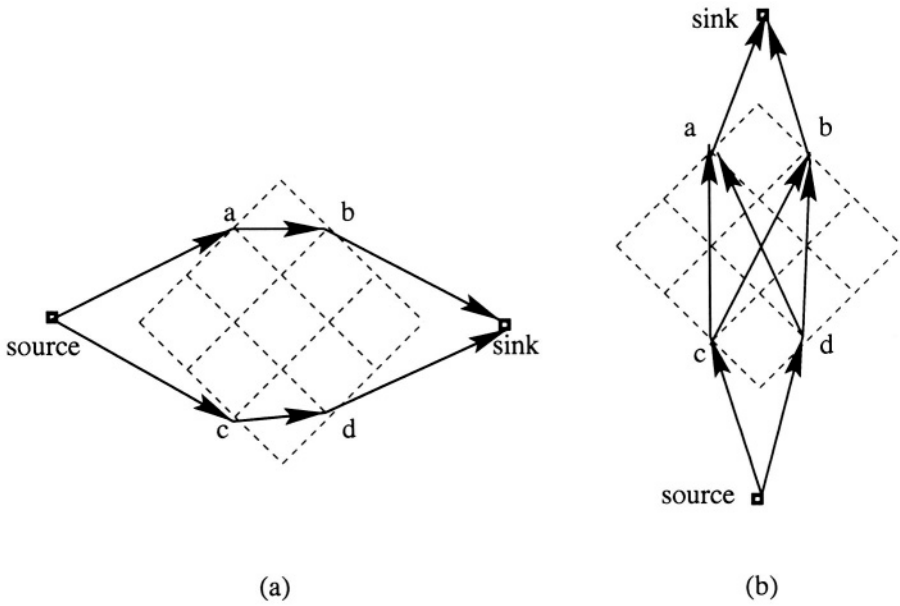

Figure 7.8: (a) Horizontal and (b) Vertical Constraint Graphs from a given sequence-pair.

weight of zero. Neither of these graphs contains any directed cycle. Module pairs that have horizontal edges in G_h do not overlap horizontally and similarly module pairs that have vertical edges in G_v do not overlap vertically. Thus no two modules overlap each other in the resultant placement because any pair of modules are either in horizontal or vertical relation. The width and height of the chip is determined by the longest path length between the source and the sink in G_h and G_v. Since the width and height of the chip is independently minimum, the resultant packing is the best of all the packings under the constraint. The longest path length calculation on each graph can be done in O(m2) time, proportional to the number of edges in the graph.

For a given chip C of m modules $O(m!)^2$ sequence-pairs are possible and each sequence-pair can be mapped to a packing in $O(m^2)$ time, and atleast one of the sequence-pair corresponds to the optimal packaging solution. When the orientation of the block is not fixed then the size of the solution space increases to $(m!)^2 2^m$.

Authors of [MFNK96] applied this technique in a simulated annealing algorithm where move is a change of the sequence-pair. They have used three kinds of pair-interchanges.

1. Two module names in S_1, for placement optimization.
2. Two module names in both S_1 and S_2 for placement optimization.

3. Width and Height of a module for orientation optimization.

The initial sequence-pair is made as $S_1 = S_2$, which corresponds to a linear horizontal arrangement of modules. The temparature was decreased exponentially. Operation 1 was performed with higher probability in higher temperatures and operation 3 was performed with higher probability in lower temperatures to achieve better results.

The above technique can be extended to consider wire lengths also.

7.3.5 Comparison of Simulation Based Algorithms

Both the simulated annealing and simulated evolution are iterative and probabilistic methods. They can both produce optimal or near-optimal placements, and they are both computation intensive. However, the simulated evolution has an advantage over the simulated annealing by using the history of previous trial placements. The simulated annealing can only deal with one placement configuration at a time. In simulated annealing it is possible that a good configuration maybe obtained and then lost when a bad configuration is introduced later. On the other hand, the good configuration has much better chance to survive during each iteration in simulated evolution since there are more than one configurations being kept during each iteration. Any new configuration is generated by using several configurations in simulated evolution. Thus, history of previous placements can be used. However, the genetic method has to use much more storage space than the simulated annealing since it has to memorize all individual configurations in the population. Unlike simulated annealing and simulated evolution, force directed placement is applicable to general designs, such as full custom designs. The force-directed methods are relatively faster compared to the simulated annealing and genetic approaches, and can produce good placement.

7.4 Partitioning Based Placement Algorithms

This is an important class of algorithms in which the given circuit is repeatedly partitioned into two subcircuits. At the same time, at each level of partitioning, the available layout area is partitioned into horizontal and vertical subsections alternately. Each of the subcircuits so partitioned is assigned to a subsection. This process is carried out till each subcircuit consists of a single gate and has a unique place on the layout area. During partitioning, the number of nets that are cut by the partition is usually minimized. In this case, the group migration method can be used.

7.4.1 Breuer's Algorithm

The main idea for Breuer's algorithm [Bre77a, Bre77b] is to reduce the number of nets being cut when the circuit is partitioned. Various objective functions

have been developed for this method. These objective functions are as given below.

1. **Total net-cut objective function:** All the nets that are cut by the partitioning are taken into account. This sum includes all nets cut by both horizontal and vertical partitioning cut lines. Minimizing this value is shown to be equivalent to minimizing the semi-perimeter wire-length [Bre77a, Bre77b].

2. **Min-max cut value objective function:** In the case of standard cells and gate arrays, the channel width depends on the number of nets that are routed through the channel. The more the number of nets the larger is the channel width and therefore the chip size. In this case the objective function is to reduce the number of nets cut by the cut line across the channel. This will reduce the congestion in channels having a large number of nets but at the expense of routing them through other channels that have a fewer number of nets or through the sparser areas of the channel.

3. **Sequential cut line objective function:** A third objective function is introduced to ease the computation of net cuts. Even though the above two objective functions represent a placement problem more accurately, it is very difficult to compute the minimum net cuts. This objective function reduces the number of nets cut in a sequential manner. After each partition, the number of nets cut is minimized. This greedy approach is easier to implement, however, it may not minimize the total number of nets cut.

In addition to the different objective functions, Breuer also presented several placement procedures in which different sequence of cut lines are used.

1. **Cut Oriented Min-Cut Placement:** Starting with the entire chip, the chip is first cut by a partition into two blocks. The circuit is also partitioned into two subcircuits so that the net cut is minimized. All the blocks formed by the partition are further partitioned by the second cut line and this process is carried out for all the cut lines. This partitioning procedure is sequential and easy to implement but it does not always yield good results because of the following two reasons. Firstly, while processing a cut line, the blocks created by the previous cut lines have to be partitioned simultaneously. Secondly, when a cut line partitions a block into two, the blocks to be placed in one of the partition might not fit in the partition created by the cut line as it might require more space than the block to be placed in the other partition (see Figure 7.9(a)).

2. **Quadrature Placement Procedure:** In this procedure, each region is partitioned into four regions of equal sizes by using horizontal and vertical cut lines alternatively. During each partitioning, the cutsize of the partition is minimized. As it cuts through the center and reduces the

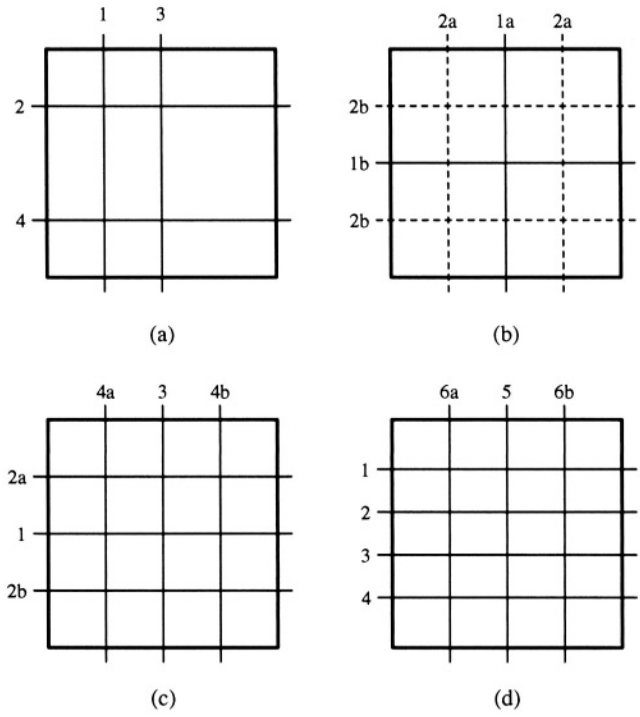

Figure 7.9: Different sequences of cut lines.

cutsize, this process reduces the routing density in the center. Currently, this is the most popular sequence of cut lines for min-cut algorithms (see Figure 7.9(b)).

3. **Bisection Placement Procedure:** The layout area is repeatedly bisected (partitioned into two equal parts) by horizontal cut lines until each subregion consists of one row. Each of these rows is then repeatedly bisected by vertical cut lines till each resulting subregion contains only one slot thus fixing the positions of all blocks. This method is usually used for standard cell placement and does not guarantee the minimization of the maximum net cut per channel (see Figure 7.9(c)).

4. **Slice Bisection Placement Procedure:** In this method, a suitable number of blocks are partitioned from the rest of the circuit and assigned to a row, which is called a slicing, by horizontal cut lines. This process is repeated till each block is assigned to a row. The blocks in each row are then assigned to columns by bisecting using vertical cut lines. This technique is most suitable for circuits which have a high degree of interconnection at the periphery since this procedure tends to reduce the wire

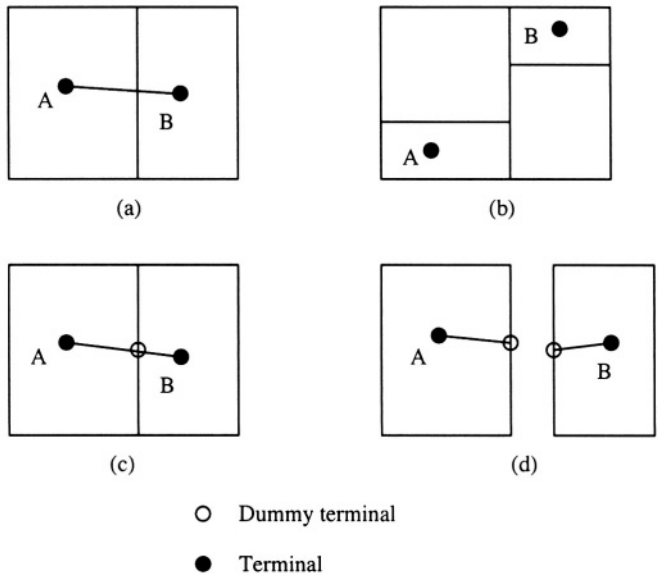

Figure 7.10: Terminal propagation.

congestion at the periphery (see Figure 7.9(d)).

In any procedure described above, if the partitioning is to minimize the number of nets cut by the partition, a group migration method can be used in the partitioning process.

7.4.2 Terminal Propagation Algorithm

The partitioning algorithms partitioned the circuit merely to reduce the net cut. Therefore, the partitioning algorithms cannot be directly used for placement. This is illustrated in Figure 7.10. If the partitioning algorithm were to be used directly, terminals A and B may move away from each other as a result of partitioning, as shown in Figure 7.10(b). This not only increases the net length but increases the congestion in the channels as well. Hence unlike partitioning algorithms, placement algorithms which are based on partitioning need to preserve the information regarding the terminals which are connected and fall into two different partitions because of the cut. This can be done by propagating a dummy terminal to the nearest point on the boundary, when a net connecting two terminals is cut, as shown in Figure 7.10(c). When this dummy terminal is generated, the partitioning algorithm will not assign the two terminals in each partition, as shown in Figure 7.10(b), into different partitions as this would not result in a minimum cut. This method called the *terminal propagation* method was developed by Dunlop and Kernighan [DK85].

```
Algorithm CLUSTER-GROWTH
begin
    let B be the set of blocks to be placed;
    select a seed block B block from B;
    place B in the layout;
    B = B − B;
    while (B ≠ φ) do
        select a block B from B;
        place B in the layout;
        B = B − B;
end.
```

Figure 7.11: The Cluster growth algorithm.

7.5 Other Placement Algorithms

In this section, different kind of placement algorithms are described, which are neither simulation based nor partition based. These include cluster growth, quadratic assignment, resistive network optimization, and branch-and-bound algorithms.

7.5.1 Cluster Growth

In this constructive placement algorithm, the bottom-up approach is used. Blocks are placed sequentially in a partially completed layout. The seed or the first block is usually selected and placed by the user. After the seed block is placed, other blocks are selected and placed one by one to complete the layout. The selection and placement techniques differentiate in various cluster growth techniques.

In cluster growth algorithm, the block that is highly connected (have the most connections) to the already placed blocks is selected to be placed. Then, this block is placed either close to the block that it is highly connected to or a exhaustive search is carried out for the best possible location for the block. The outline of the cluster growth algorithm is shown in Figure 7.11.

The random constructive placement is a degenerate form of cluster growth. In this case, the selection of blocks is made randomly and its position is also fixed randomly. As this method does not take into account the interconnections and other circuit features, in most of the cases, it does not produce a good layout. This method is sometimes utilized to generate a basic layout for an iterative placement algorithm.

7.5.2 Quadratic Assignment

This method solves an abstract version of the gate array placement problem. It assumes that the blocks are points and have zero area. The cost of connecting two blocks B_i and B_j given by c_{ij} is stored in a connection matrix. The distance between slot k and slot l, given by d_{kl} is stored in a distance matrix. The objective is to map the blocks onto slots such that the product of connectivity and distance between the slots to which the blocks have been mapped (which gives the net length), for all the blocks, is minimized. This objective is equivalent to minimizing the total wire length for the circuit. This placement problem has been formulated as a quadratic assignment problem by Hall [Hal70]. If C is the connection matrix and c_i is the sum of all elements in the ith row of C, then a diagonal matrix D can be defined as,

$$d_{ij} = \begin{cases} 0, & \text{if i} \neq \text{j}, \\ c_i, & \text{if } i = j \end{cases}$$

Let a matrix E be defined as $E = D - C$ and $X^T = [x_1, x_2, \ldots, x_n]$ and $Y^T = [y_1, y_2, \ldots, y_n]$ be row vectors representing the x– and y– coordinates of the desired solution. Hall proved that a nontrivial solution is obtained and the objective function is minimized if the smallest eigenvalues of the matrix E are chosen. The corresponding eigenvectors X and Y then give the coordinates of all the blocks.

7.5.3 Resistive Network Optimization

The placement problem has been transformed into the problem of minimizing the power dissipation in a resistive network by Cheng and Kuh [CK84]. The objective function, which is the squared Euclidean wire length, is written in a matrix form. This representation is similar to the matrix representation of resistive networks. This method can include fixed blocks in the formulation. Also, blocks with irregular sizes are allowed within cell rows. The algorithm comprises of subprograms which are used for optimization, scaling, relaxation, partitioning and assignment. The efficiency of the method comes from the fact that it takes advantage of the sparsity of the netlist. Slot constraints are used which guarantee the placement of blocks to be legal and each block is allocated to one slot. There are upto n constraints, where n is the number of blocks. The slot constraints are given by the equation

$$\sum_{i=1}^{n} x_i^j = \sum_{i=1}^{n} p_i^j, \qquad 1 \leq j \leq n$$

The algorithm maps the given circuit to a resistive network in which the pads and fixed blocks are represented as fixed voltage sources. Using the slot constraints the power dissipation in the circuit is minimized which causes the blocks to cluster around the center of the chip. The higher order slot constraints when applied cause the blocks to spread out. This step is called the *scaling step.* A repeated partitioning and relaxation then aligns the blocks with the slot locations.

7.5.4 Branch-and-Bound Technique

The general branch-and-bound algorithm can be applied to the placement problem. This method can be used for small circuits as it is a computationally intensive method. The method assumes that all the feasible solutions and the scores of these solutions are known. All these solutions make up a set called the *solution set.* The solution can be systematically searched. The search can be actually represented by a tree structure. The leaves of the tree are all the solutions. The selection of a solution is equivalent to traversing a branch of the tree and this step is called the *branch* step. If at any node in this tree a solution yields a score which is greater than the currently known lowest, then the search continues in another part of the decision tree. This step is the *bound* step. Hence the algorithm actually prunes the decision tree which results in reduced computation.

Consider a gate array with three slots S_1, S_2, S_3 and three blocks B_1, B_2, B_3. At the first level of the tree, the root has three branches, each corresponding to a different placement of B_1 in three different slots. All the child nodes of the root will have two branches, each specifying two positions of B_2 in the remaining two slots. Finally, all grand children of root will have exactly one branch, specifying the slot for B_3.

The branch-and-bound algorithm traverses the tree and computes the cost of the solution at any given node. The cost can simply be the total wire length due to the placement of blocks upto that node. If this cost is higher than another known placement, this subtree need not be explored.

7.6 Performance Driven Placement

The delay at chip level, which depends on interconnecting wires plays a major role in determining the performance of the chip. As the blocks in a circuit become smaller and smaller, the size of the chip decreases. As a result, the delay due to the connecting wire becomes a major factor for high performance chips. The placement algorithms for high performance chips have to generate placements which will allow routers to route nets within the timing requirements. Such problems are called performance driven placement and the algorithms are called performance driven algorithms. The performance driven placement algorithms can be classified into two major categories, one which use the net-based approach and the other which use the path-based approach. In path-based approach [Don90, JK89], the critical paths in the circuit are considered and the placement algorithms try to place the blocks in a manner that the path length is within its timing constraint. On the other hand, the net-based approach [DEKP89, Dun84, Oga86, HNY87, MSL89], tries to route the nets to meet the timing constraints on the individual nets instead of considering the paths. In this case, the timing requirement for each net has to be decided by the algorithm. Usually a pre-timing analysis generates the bounds on the netlengths which the placement algorithms have to satisfy while placing the blocks. Gao, Vaidya and Liu [GVL91] presented a algorithm for high

performance placement. The algorithm consists of the following steps:

1. Upper bounds for the netlengths are deduced from the timing requirements which is a part of the input to the algorithm. Each net has a set of such upper bounds. This provides the algorithm with maximum flexibility. The timing requirements are expressed by a set of linear constraints which are solved using convex programming techniques. A new convex programming algorithm is used for which the computational complexity depends only on the number of variables rather than the number of linear constraints.

2. A modified min-cut placement algorithm is used to obtain the placement of the blocks. The upper bounds calculated in the previous step guide the min-cut algorithm in placing the blocks. The min-cut algorithm, is a modified version of the Fiduccia's min-cut algorithm which tries to minimize the number of nets whose lengths exceed their corresponding upper bounds in addition to minimizing the size of the cutset.

3. The next step is to check whether all timing requirements are satisfied in the placement generated by the modified min-cut placement algorithm.

4. In case all the timing requirements are met, the placement is valid and is accepted. Otherwise the set of upper-bounds obtained in step 1 is modified and the steps 2 and 3 are repeated. Most other algorithms could not handle situations where the placement generated did not meet the timing specifications.

7.7 Recent Trends

In Very Deep Sub-Micron(VDSM) designs, Placement problem is considered much more than simply achieving the routability of the design and minimizing the chip die area . Several other critical issues such as timing, zero clock-skew, even power distribution are increasing the complexity of the placement problem exponentially. Since placement phase is one of the early phases of the IC physical design, lot of attention is paid to placement phase in IC design cycle.

Timing driven placement is very critical to IC design and some of the techniques to perform timing driven placement are discussed in [DNA+90], [RMNP97], [SS95], and [SKT97].

Algorithms to estimate the wire lengths are becoming part of placement algorithms because accurate estimation of wire lengths help to fix the problems in placement phase itself rather than in routing phase. Estimation of wirelength during the placement stage helps to understand the routability of the design. One of the techniques to estimate wire length is discussed in [CKM+98].

1. Early Placement to obtain better Wire Load Models(WLM) for synthesis. WLM is a parameter for the delay estimate for logic synthesis algorithm.

2. Placement for Cross talk avoidance.

3. Placement for Minimizing clock skew.

4. Placement for even power distribution.

7.8 Summary

Placement is a key step in physical design cycle. Several placement algorithms have been presented. Simulated annealing and simulated evolution are two most successful placement algorithm. Although these algorithms are computationally intensive, they do produce good placements. Integer programming based algorithms for floorplanning have been also been successful. Several algorithms have been presented for pin assignment, including optimal pin assignment for channel pin assignment problems. The output of the placement phase must be routable, otherwise placement has to be repeated.

7.9 Exercises

1. Consider the following blocks. $R_1 = 15 \times 15, R_2 = 25 \times 15, R_3 = 10 \times 30, R_4 = 30 \times 20, R_5 = 10 \times 15, R_6 = 20 \times 5, R_7 = 10 \times 25, R_8 = 30 \times 15, R_9 = 10 \times 65, R_{10} = 10 \times 25$. If D_{ij} represents the center to center distance between blocks i and j then determine if these blocks can be placed together so that the distances between the blocks are within their specified values. The distances that are to be maintained are $D_{12} = 30, D_{23} = 35, D_{14} = 18, D_{34} = 40, D_{24} = 30, D_{13} = 45$.

† 2. Implement the Simulated Annealing algorithm. Consider the graph shown in Figure 7.12. Each vertex represents rectangle R_i whose dimensions are specified in problem 1, the edges of the graph represent the connectivity of the blocks. Use the Simulated Annealing algorithm to generate a placement.

† 3. For the placement obtained in problem 5, implement a pin assignment algorithm which will reduce the total net length and minimize the maximum length of a net. Generate a complete routing for this placement. The routing can be generated on an uniform grid and two nets can intersect only when they are perpendicular to each other.

‡ 4. Implement the Simulated annealing algorithm for the general cell placement problem.
Hint: Instead of exchanging a big block with a small block, a big block can be exchanged with a cluster of small blocks.

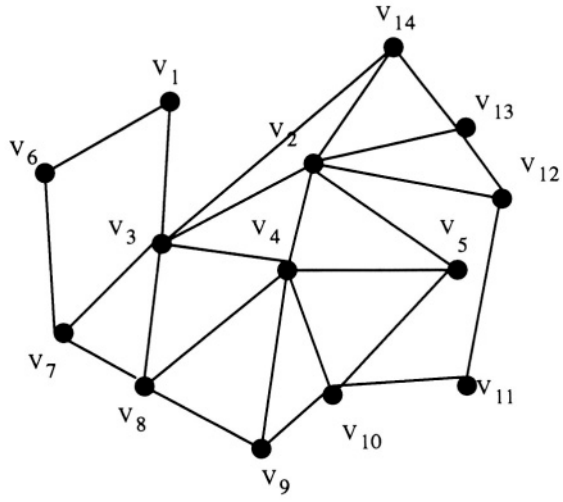

Figure 7.12: Example of a connectivity graph.

‡ 5. For a given placement, implement a pin assignment algorithm which will rotate the pins on the blocks either in clockwise or anticlockwise direction till the total net length is reduced. While rotating the pins on the blocks, the order of these pins must be maintained.

6. For the blocks specified in problem 1 generate the integer program constraints to solve the placement problem. Consider some of the blocks as flexible and solve the floorplanning problem using the integer program.

† 7. Implement a placement algorithm for high performance circuits which takes into account path delays instead of net delays.

‡ 8. Modify the min-cut algorithm to incorporate the terminal propagation scheme.

‡ 9. Develop Simulated Evolution algorithm for standard cells design.

‡ 10. Develop Simulated Evolution algorithm for full custom designs.

‡ 11. Several industrial libraries allow cells with different cell heights. This leads to irregular shape channels. Suggest modifications required for applying the Simulated annealing algorithm to standard cells of uneven heights.

‡ 12. Implement force directed placement algorithm for gate array design style.

Bibliographic Notes

A linear assignment algorithm for the placement problem has been discussed in [Ake81]. Two partition/interchange processes are described in [Pat81] for

solving the placement problem. The graph is partitioned into several smaller graphs for initial placement in both the methods and finally interchange optimization is carried out. The simulated annealing optimization method has been adapted to the placement of macros on chips for full custom design in [JJ83].

A hierarchical placement procedure incorporating detailed routing and timing information has been discussed in [GKP+90]. The procedure is based on the min-cut method. Global routing and timing analysis is carried out after every cut which guides the subsequent cell partitioning. [Leb83] discusses an interactive program to get a good floorplan. It includes graphical output, block and pad manipulation and a cost function for estimation of total wire length. In [MM93] S. Mohan and P. Mazumder present a placement algorithm in the distributed computing environment. In [SSL93] S. Sutanthavibul, E. Shragowitz, and R. Lin present a timing-driven placement algorithms for high performance VLSI chips. In [SDS94] Shanbhag, Danda, and Sherwani presented an algorithm for mixed macro block and standard cell designs. Algorithm for mixed macro-cell and standard-cell placement to minimize the chip size and interconnection wire length is presented in [XGC97]. Quadratic placement technique is revisited in [ACHY97].