CHAPTER

11 Placement

Chris Chu
Iowa State University, Ames, Iowa

ABOUT THIS CHAPTER

Placement is the process of determining the locations of circuit devices on a die surface. It is an important stage in the VLSI design flow, because it affects routability, performance, heat distribution, and to a less extent, power consumption of a design. Traditionally, it is applied after the logic synthesis stage and before the routing stage. Since the advent of deep submicron process technology around the mid-1990s, interconnect delay, which is largely determined by placement, has become the dominating component of circuit delay. As a result, placement information is essential, even in early design stages, to achieve better circuit performance. In recent years, placement techniques have been integrated into the logic synthesis stage to perform physical synthesis and into the architecture design stage to perform physical-aware architecture design.

This chapter begins with an introduction to the placement stage. Next, various placement problem formulations are discussed. Then, partitioning-based approach, simulated annealing approach, and analytical approach for global placement are presented. After that, legalization and detail placement algorithms are described. The chapter concludes with a discussion of other placement approaches and useful resources to placement research.

11.1 INTRODUCTION

Traditionally, placement is the design stage after logic synthesis and before routing in the VLSI design flow. In logic synthesis, a netlist is generated. Then in placement, the locations of the circuit modules in the netlist are determined. After placement, routing is performed to lay out the nets in the netlist.

Placement is a critical step in the VLSI design flow mainly for the following four reasons. First, placement is a key factor in determining the performance of a circuit. Placement largely determines the length and, hence, the delay of interconnect wires. As feature size in advanced VLSI technology continues to reduce, interconnect delay has become the determining factor of circuit performance.

Interconnect delay can consume as much as 75% of clock cycle in advanced design. Therefore, a good placement solution can substantially improve the performance of a circuit. Second, placement determines the routability of a design. A well-constructed placement solution will have less routing demand (*i.e.*, shorter total wirelength) and will distribute the routing demand more evenly to avoid routing hot spots. Third, placement decides the distribution of heat on a die surface. An uneven temperature profile can lead to reliability and timing problems. Fourth, power consumption is also affected by placement. A good placement solution can reduce the capacitive load because of the wires (by having shorter wires and larger separation between adjacent wires). Hence the switching power consumption can be reduced.

In recent years, it has become essential for the logic synthesis stage to incorporate placement techniques to perform physical design aware logic synthesis (*i.e.*, physical synthesis). The reason is that without some placement information, it is impossible to estimate the delay of interconnect wires. Hence, given the significance of interconnect delay, logic synthesis will not have any meaningful timing information to guide the synthesis process. As a result, the synthesized netlists will have poor performance after placement. For the same reason, consideration of placement information during architecture design is also increasingly common.

Placement is a computationally difficult problem. Even the simple case of placing a circuit with only unit-size modules and 2-pin nets along a straight line to minimize total wirelength is NP-complete [Garey 1974]. The VLSI placement problem is much more complicated. The circuit may contain modules of different sizes and may have multi-pin nets. The placement region is two-dimensional. Other cost functions may be used rather than total wirelength. There may also be different constraints for different design styles. (Details of problem formulations can be found in Section 11.2.) As designs with millions of modules are now common, it is a major challenge to design efficient placement algorithms to produce high-quality placement solutions.

One way to overcome the complexity issue is to perform placement in several manageable steps. One common flow is as follows.

1. **Global placement.** Global placement aims at generating a rough placement solution that may violate some placement constraints (*e.g.*, there may be overlaps among modules) while maintaining a global view of the whole netlist.
2. **Legalization.** Legalization makes the rough solution from global placement legal (*i.e.*, no placement constraint violation) by moving modules around locally.
3. **Detailed placement.** Detailed placement further improves the legalized placement solution in an iterative manner by rearranging a small group of modules in a local region while keeping all other modules fixed.

The global placement step is the most important one of the three. It has the most impact on placement solution quality and runtime, and has been the focus

of most prior research works. After global placement, the placement solution is almost completely determined. In legalization and detailed placement, only local changes in module locations will be made. Therefore, the main emphasis of this chapter is the global placement step. The most commonly used global placement approaches are **partitioning-based approach, simulated annealing approach**, and **analytical approach**. The analytical approach will be presented with the most details, because it is currently the best approach in both quality and runtime.

11.2 PROBLEM FORMULATIONS

The input to the placement problem is a placement region, a set of modules, and a set of nets. The widths and heights of the placement region and all modules are given. The locations of I/O pins on the placement region and on all modules are fixed. Sometimes, some input modules (*e.g.*, buffer bays, I/O modules, IP blocks) are preplaced by designers, and, hence, their locations are also fixed before placement. Each net specifies a collection of pins in the placement region and/or in some modules that are connected. Basically, placement is to find a position for each module within the placement region so that there is no overlap among the modules and some objective is optimized. Many variations in the placement problem formulation exist, because different designs may require different objectives and different design styles may introduce different constraints. The placement problems for common design styles and objectives are presented in the following.

11.2.1 Placement for different design styles

11.2.1.1 *Standard-cell placement*

In a standard-cell design, all modules have the same height. The placement of standard cells has to be aligned with some prespecified standard-cell rows in the placement region. Because of the popularity of standard-cell design, most placement algorithms assume a standard-cell design style.

11.2.1.2 *Gate array/FPGA placement*

In gate array or FPGA design, the modules can only be placed at some predefined sites that are arranged in a regular array.

11.2.1.3 *Macro block placement*

In macro block placement, each module is a macro block of fixed shape and orientation. The macro blocks have to be placed within the placement region without overlap among them. The macro block placement problem is similar to the fixed-outline floorplanning problem. However, in floorplanning, the shape and the orientation of the macro blocks are usually assumed to be

changeable. Macro block placement can be considered to be a special case of fixed-outline floorplanning. If the number of modules is small, it can be solved by floorplanning techniques (please refer to Chapter 10).

11.2.1.4 *Mixed-size placement*

Mixed-size placement places both macro blocks and standard cells in a circuit. Modern designs often contain a large number of macro blocks together with a huge number of standard cells. As a result, mixed-size placement is a common formulation in recent years. Because macro blocks are typically orders of magnitude larger than standard cells, the handling of the nonoverlapping constraints among the modules presents a unique challenge.

11.2.2 Placement objectives

11.2.2.1 *Total wirelength*

Total wirelength is the most commonly used objective in placement formulations. Minimization of total wirelength indirectly optimizes several other objectives. First, routability can be improved by less routing demand. Second, timing can be better because shorter wires have less delay. Third, power consumption can be reduced because shorter wires also introduce less capacitive load. Notice that total wirelength minimization is only a heuristic in optimizing these other objectives. Even if the *total* wirelength is reduced, nets in the most congested region, along the timing critical paths, or with the highest switching activities may not be shorter. To specify the importance of different nets in optimizing another objective, a weight can be assigned to each net. Then the total weighted wirelength will have a much better correlation to the other objective.

It is difficult to predict during placement the wirelength of a net after routing, because it is router-dependent. Several approaches estimate the routed wirelength for a given placement. The most widely used approach is ***half-perimeter wirelength*** (HPWL). The HPWL of a net is equal to half of the perimeter of the smallest bounding rectangle that encloses all the pins of the net. For example, for the 5-pin net in Figure 11.1a, the HPWL is $W + H$ as shown in Figure 11.1b. HPWL is popular because it can be computed in linear time and it can be written as a simple closed-form function of the coordinates of the pins (see Section 11.5.1). It also provides exact wirelength for optimally routed nets with two and three pins. However, HPWL can significantly underestimate the wirelength for nets with four or more pins.

Another approach of wirelength estimation is based on ***rectilinear minimum spanning tree*** (RMST). A RMST is a tree with minimum total wirelength in Manhattan distance to connect a given set of nodes. For example, the RMST of the net in Figure 11.1a is given in Figure 11.1c. The best time complexity for RMST construction is $O(n \log n)$ [Guibas 1983]. RMST wirelength is exact for nets with two pins, but it can overestimate the wirelength of optimally routed nets with three or more pins by up to 50% [Hwang 1976]. In practice, RMST

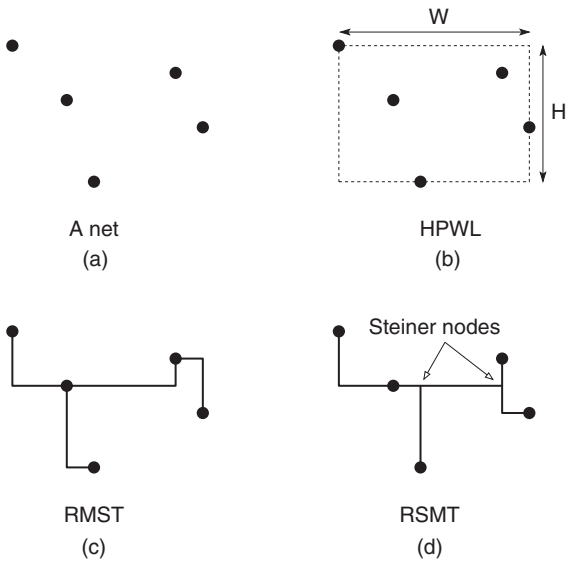

FIGURE 11.1

Wirelength estimation techniques.

can produce more accurate wirelength estimation than HPWL (especially for nets with high pin count) in a runtime several times more than that of HPWL.

A highly accurate approach is based on ***rectilinear Steiner minimal tree*** (RSMT). A RSMT is a tree with minimum total edge length in Manhattan distance to connect a given set of nodes possibly through some extra (*i.e.*, Steiner) nodes. For example, the RSMT of the net in Figure 11.1a is given in Figure 11.1d. If there is no routing congestion, RSMT is the preferred way to route a net, because it gives the minimum wirelength. Thus, RSMT wirelength has a very high correlation with routed wirelength unless the design is heavily congested and requires a lot of detour in routing. RSMT construction is NP-complete [Garey 1979]. In the past, all exact and heuristic RSMT algorithms were very time-consuming. Hence, traditionally, RSMT was rarely used for wirelength estimation during placement. Recently, a lookup–table based RSMT heuristic algorithm called FLUTE [Chu 2008] was introduced. It is more accurate than all previous RSMT heuristics, yet it is as fast as RMST algorithms. With FLUTE, it is feasible for placers to use the highly accurate RSMT wirelength in guiding the placement process.

11.2.2.2 ***Routability***

Routability is the most basic requirement of a placement solution. Any placement solution is useless if the routing cannot be completed. However, the routability of a placement solution is very hard to evaluate. Routability is router-dependent. There is no objective measure of routability. Even for a specific router, the routability is still very hard to estimate because of the complicated behavior of a

router. One way of routability estimation is to call the router to perform a rough routing (*e.g.*, global routing), but this way is computationally very expensive. A more popular way is to assume the routing of each net follows some probability distribution and then estimate the routing congestion of each edge in the routing grid by the expected number of crossing nets. However, this way is not accurate and is still quite expensive computationally.

Because of the high computational cost, routability estimation is rarely incorporated into the placement objective function in guiding the placement process in practice. Instead, routability is often indirectly optimized during placement by a **white space allocation approach**. Regions that are predicted to be congested are given more white space (*i.e.*, placed with a lower module density) to provide more routing tracks.

11.2.2.3 *Performance*

Placement has significant impact on the delay of interconnects, and hence the performance of circuits. Because interconnect delay becomes a more dominating component of circuit delay as feature size continues to decrease, performance-driven placement is increasingly important. However, the delay of a net heavily depends on other factors like routing, buffering, driver size, wire width, and wire spacing. It is computationally too expensive to perform those tasks during the placement process. In other words, it is basically impossible to obtain accurate timing information during placement. In practice, the delay of a net is heuristically controlled during placement by controlling the length of the net. The net delay can be either reduced by assigning a larger net weight or bounded by constraining the net length.

11.2.2.4 *Power*

For most circuits, the major component of power consumption is switching power, which is consumed whenever a gate switches (*i.e.*, the capacitive load driven by the gate is charged/discharged). The capacitance of a net is proportional to its wirelength. So the power minimization problem can be formulated as a wirelength minimization problem. As power consumption is also proportional to the amount of switching, the nets should be weighted by their switching activity factors. Note that the clock distribution network is a global net driving a huge load and switches every clock cycle. It consumes a significant portion of switching power. Therefore, special attention should be paid to the placement of clocked elements (including latches, flip-flops, memories, and dynamic gates) so that the capacitance of clock wires can be minimized without increasing clock skew or degrading timing.

11.2.2.5 *Heat distribution*

An uneven temperature profile on a chip may adversely affect the characteristics of temperature-sensitive circuits. It may also lead to reliability problems. Therefore, it is desirable to properly distribute the heat-generating elements of a circuit to achieve an even temperature profile.

Thermal-driven placement is difficult for several reasons. First, the heat generation of each element changes over time, because it depends on the operations performed by the circuit. Second, heat is generated by both transistors and wires, and the heat generation by wires is hard to predict during placement. Third, the temperature profile is determined by both generation and transfer of heat and is difficult to approximate without the use of time-consuming simulation.

A practical solution to the thermal-driven placement problem is to distribute the average heat generation of modules evenly in the placement region (by assuming the dynamic nature of heat generation, wire heat generation, and heat transfer are secondary effects). This module heat distribution problem is similar to the module area distribution problem in placement and can be solved by similar techniques.

11.2.3 **A common placement formulation**

Although there are many variations in placement problem formulations for different design styles and with different objectives, the underlying issues for the various formulations are the same. For placement of different design styles and for thermal-driven placement, the modules have to be properly distributed. For optimization of different objectives, the lengths of wires have to be reduced.

In this chapter, the focus is on the very popular problem of minimum-wirelength placement for standard-cell design. In particular, total HPWL is considered to be the objective function, because RSMT wirelength is traditionally expensive to compute and difficult to optimize. An algorithm for this formulation can usually be extended to handle other design styles and objectives. For other design styles (even with preplaced modules), as long as the module density can be properly controlled during global placement, minor placement constraint violations can be easily resolved during legalization. The thermal-driven and routability-driven placement problems can also be handled by controlling heat density and white space density, respectively, in addition to area density during global placement. The performance and power optimization problems can be formulated as a weighted wirelength minimization problem.

11.3 **GLOBAL PLACEMENT: PARTITIONING-BASED APPROACH**

Roughly speaking, the **partitioning problem** is to divide a circuit into several subcircuits of similar sizes such that the number of connections among subcircuits is minimized. A circuit placement can be generated by recursively applying a partitioning procedure. Such an approach is called **partitioning-based placement** or **min-cut placement**. In the following sections, the basics for partitioning will first be introduced. The application of partitioning to placement will then be presented.

11.3.1 Basics for partitioning

Most partitioning algorithms solve the **bipartitioning** (or **2-way partitioning**) problem, which is to divide a circuit into two subcircuits. The bipartitioning problem will be discussed in the following.

11.3.1.1 *Problem formulation*

The circuit bipartitioning problem can be formulated as a hypergraph bipartitioning problem by modeling a circuit as a hypergraph. A circuit module is represented by a vertex, and the area of the module is modeled as the vertex size. A net is represented by a hyperedge, and the criticality of the net is modeled as the hyperedge weight. For example, the circuit in Figure 11.2a can be modeled by the hypergraph in Figure 11.2b in which the set of vertices is {A, B, C, D, E} and the set of hyperedges is {{A, B}, {A, C}, {A, D}, {B, D}, {D, E}, {B, C, E}}.

Given a hypergraph $G(V, E)$, where each vertex $v \in V$ has a size $s(v)$ and each hyperedge $e \in E$ has a weight $w(e)$, the hypergraph bipartitioning problem is to divide the set V into two subsets V_1 and V_2 such that the total weight of hyperedges being cut (*i.e.*, spanning both subsets) is minimized and the total sizes of vertices in V_1 and in V_2 are close to some user-defined values. Formally, the cost function can be written as:

$$CutCost(V_1, V_2) = \sum_{e \in E \text{ s.t. } e \cap V_1 \neq \emptyset \wedge e \cap V_2 \neq \emptyset} \omega(e)$$

The size constraint on V_1 can be specified by use of a ratio parameter γ and a tolerance parameter ε as follows:

$$\gamma - \epsilon \leq \sum_{v \in V_1} s(v) / \sum_{v \in V} s(v) \leq \gamma + \epsilon$$

The size constraint on V_2 is indirectly specified as

$$\sum_{v \in V_2} s(v) = \sum_{v \in V} s(v) - \sum_{v \in V_1} s(v)$$

Hypergraph bipartitioning is NP-complete [Garey 1979].

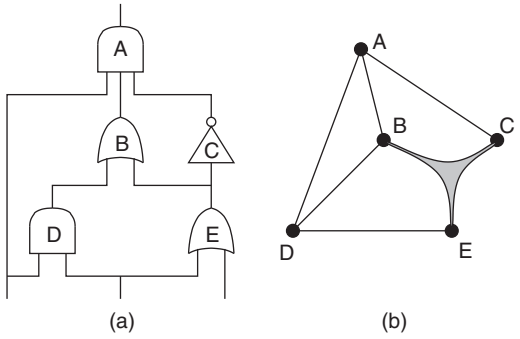

FIGURE 11.2

The hypergraph model of a circuit.

11.3.1.2 *The Fiduccia-Mattheyses algorithm*

A classical approach to solve the hypergraph bipartitioning problem is an iterative heuristic by Fiduccia and Mattheyses [Fiduccia 1982]. This heuristic is commonly called the FM algorithm. To explain the FM algorithm, the concept of the **gain** of a vertex has to be first introduced. Given a bipartitioning solution, the gain $g(v)$ of a vertex v is the reduction in the cut cost if vertex v is moved from its current partition to the other partition. For example, for the hypergraph in Figure 11.2b, assume all hyperedges have a weight of 1. In the initial bipartition as shown in Figure 11.3a, three hyperedges {A, C}, {B, C, E}, and {D, E} are cut. Hence the cut cost is 3. If vertex E is moved to the other partition as shown in Figure 11.3b, the cut cost becomes 2. Therefore, $g(\text{E}) = 3 - 2 = 1$. If vertex A is moved to the other partition as shown in Figure 11.3c, the cut cost becomes 4. Therefore, $g(\text{A}) = 3 - 4 = -1$.

The FM algorithm is described in Algorithm 11.1. The basic idea of the FM algorithm is to iteratively refine the current bipartition by greedily moving a vertex with maximum gain to the other side (step 6). Steps 3 to 12 are called a **pass**. In a pass, each vertex is tentatively moved to the other side one time (step 8). Once a vertex is moved, it will be locked (step 9) and will not be considered again until next pass. At the end of a pass, the sequence of first k moves that provides the best total gain G is made permanent (step 12). Note that g_j for some $j \in \{1, \ldots, k\}$ may be negative. That means moving v_j to the other side will make the cut worse. For a simple greedy algorithm that looks at the gain value of one vertex to decide a move, it will not proceed further. In other words, it gets trapped at a local minimum. For the FM algorithm, however, the move may still be taken, because moves subsequent to that may eventually improve the cut. Therefore, it can get out of the local minimum and generate bipartitions with better cut cost. The algorithm is repeated until there is no improvement in a pass (*i.e.,* $G = 0$).

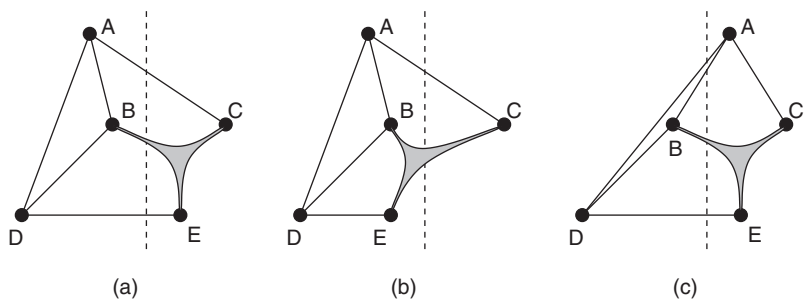

FIGURE 11.3

Illustration of the gain of a vertex.

Algorithm 11.1 The *Fiduccia and Mattheyses* (FM) Algorithm

1. Get an initial bipartition;
2. Repeat
3. Unlock all vertices;
4. For $i = 1$ to n where $n = |V|$
5. Begin
6. Select a vertex v_i with maximum gain among all unlocked vertices that will not violate the size constraints if moved to the other side;
7. Set $g_i = g(v_i)$;
8. Tentatively move vertex v_i to the other side;
9. Lock vertex v_i;
10. End
11. Find k such that $G = \sum_{i=1}^{k} g_i$ is maximized;
12. Make the moves for $v_1, \ldots, v_k$ permanent and discard the moves for $v_{k+1}, \ldots, v_n$;
13. Until $G = 0$;

Let n be the number of vertices of the hypergraph and p be the total number of pins in all hyperedges. For a given bipartition, the gain of all vertices can be computed in a straightforward manner described in the following. Let $g_e(v)$ be the contribution of hyperedge e to $g(v)$ for any $v \in V$ and $e \in E$. In other words, $g(v) = \sum_{e \in E} g_e(v)$. If $v \notin e$, then obviously $g_e(v) = 0$, because moving vertex v to the other partition does not affect whether or not hyperedge e is cut. Consider $v \in e$. Suppose the vertices in e are divided into two subsets e_1 and e_2 according to the bipartition. Assume without loss of generality that $v \in e_1$. If $e_1 = \{v\}$, then $g_e(v) = w(e)$, as e is cut in the given bipartition but will not be cut if v is moved to the other partition. If $e_2 = \emptyset$, then $g_e(v) = -w(e)$ as e is not cut in the given bipartition but will be cut if v is moved to the other partition. Otherwise, $g_e(v) = 0$ as e remains cut regardless of the partition v is in. For any $e \in E$, $g_e(v)$ for all $v \in e$ can be computed in $O(|e|)$time. Therefore, $g(v)$ for all $v \in V$ can be computed in $O(p)$ time. In other words, step 6 takes $O(p)$ time. Hence, each pass takes $O(np)$ time.

A technique based on incremental gain computation is presented in [Fiduccia 1982]. This technique reduces the runtime of each pass to $O(p)$ if all hyperedges have unit weight.[1] Instead of computing the gains from scratch in each pass, a **gain bucket** data structure is used to store the gain values. The gain bucket data structure is illustrated in Figure 11.4. It consists of a pointer array with index

[1]The technique can be easily extended for hyperedges with bounded integer weight.

ranging from $-p_{max}$ to p_{max}, where p_{max} is the maximum number of hyperedges connecting to a vertex. Indices of the array correspond to possible gain values. The entry with index g in the array points to a list of vertices with gain g. A MAXGAIN index is maintained to keep track of the bucket for the vertices with maximum gain. Besides, a VERTEX array is kept to allow direct access to all vertices in the vertex lists.

In the FM algorithm, one gain bucket data structure is used to record the gains of the unlocked vertices for each partition. At the beginning of each pass, the gains for the current bipartition are computed, and the gain bucket data structures are constructed in $O(p)$ time. Then the vertices with the maximum gain can be obtained immediately. After a vertex is moved, the gains of the vertices connecting to it may be changed. With the help of the VERTEX array, the update of the gain bucket data structure for each vertex with changed gain takes only constant time. The number of vertices connecting to each vertex is equal to the number of pins in it. Therefore, the total update time for one pass is proportional to the total number of pins, *i.e.,* $O(p)$.

11.3.1.3 *A multilevel scheme*

The FM algorithm works well in practice for hypergraphs with up to hundreds of vertices. For larger hypergraphs, a bottom-up hierarchical scheme on the basis of recursive clustering, which is often called a **multilevel scheme**, can produce higher-quality bipartitions in a relatively small amount of runtime. The multilevel scheme consists of three phases, namely, **coarsening phase, initial partitioning phase**, and **uncoarsening and refinement phase**. The three phases are illustrated in Figure 11.5. During the coarsening phase, a sequence of successively smaller (coarser) hypergraphs is constructed by clustering heavily connected vertices together. During the initial partitioning phase, a bipartition of the coarsest hypergraph is computed by any bipartitioning algorithm. During the uncoarsening and refinement phase, the bipartition is

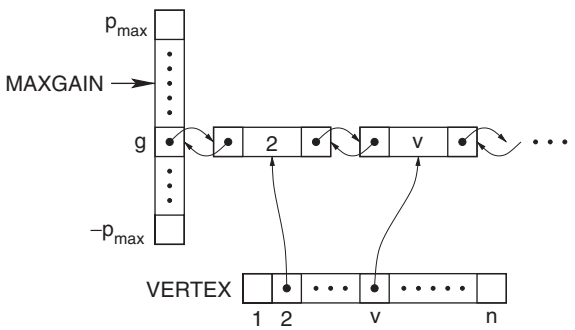

FIGURE 11.4

Gain bucket data structure of the FM algorithm.

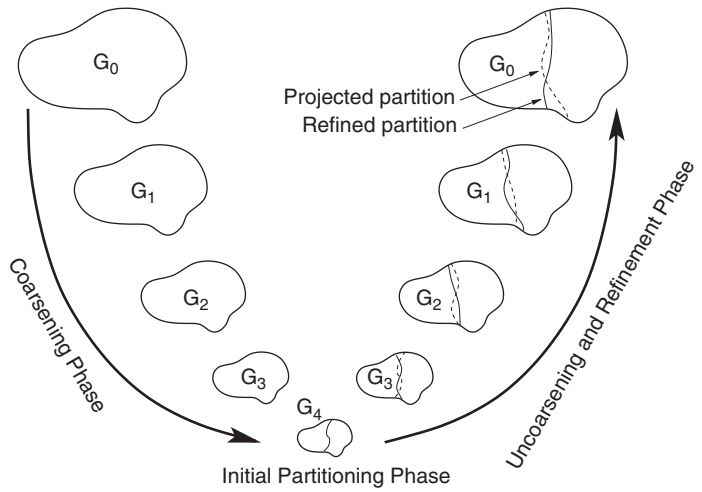

FIGURE 11.5

The three phases of multilevel bipartitioning [Karypis 1997].

successively projected to the next level finer hypergraph, and at each level an iterative refinement algorithm such as FM is used to further improve the bipartition.

hMetis [Karypis 1997] is one of the earliest and best multilevel hypergraph partitioning algorithms. In hMetis, three different coarsening techniques are applied. The initial bipartition is simply a random bipartition. The refinement of the uncoarsening hypergraphs is done by FM. Interested readers may refer to [Karypis 1997] for more details of hMetis and [Alpert 1997] for another example of high-quality multilevel partitioning algorithm. Note that the multilevel scheme can also be applied to placement as pointed out in Section 11.4.2 and discussed in Section 11.5.4.

11.3.2 Placement by partitioning

11.3.2.1 *The basic idea*

Partitioning algorithms can be used to perform placement. The partitioning-based placement approach is illustrated in Figure 11.6. Given a circuit and a placement region, the placement problem is to assign each circuit module to some specific location in the placement region. The approach starts by partitioning the circuit into two subcircuits A and B, and correspondingly, dividing the placement region by a **cutline** into two subregions A and B (see Figure 11.6b). The areas of subregions A and B should be bigger than the total module areas of subcircuits A and B, respectively. Then subcircuits A and B are assigned to subregions A and B, respectively. In other words, the placement of

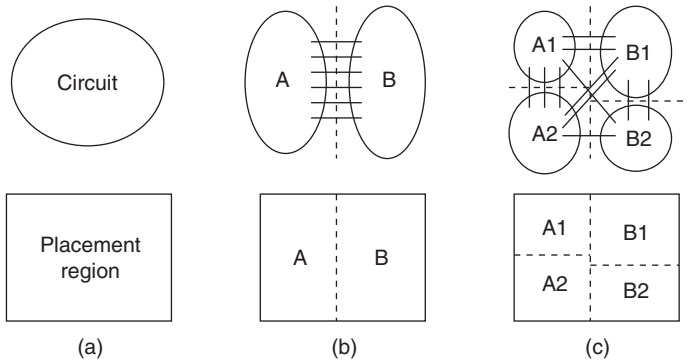

FIGURE 11.6

Partitioning-based placement approach.

each module is restricted to a smaller region. The locations of modules can be further restricted by recursively partitioning the subcircuits and dividing the subregions (as shown in Figure 11.6c). The process continues until each subcircuit contains a few modules that are assigned to a small subregion. After that, legalization is performed to pack all modules in each subcircuit into the corresponding subregion, and detailed placement is applied to further reduce the wirelength.

Note that cutlines of different directions (*i.e.*, horizontal or vertical) and locations may be used for each subregion. For example, in the division of subregion *A* in Figure 11.6c, the cutline could be vertical instead of horizontal. It could also be near the top instead of close to the middle. Different schemes of cutline selection are discussed in [Breuer 1977a, 1977b].

In partitioning-based placement, the minimization of the cut cost during partitioning can be considered to be an indirect way to minimize wirelength. The intuition is that to minimize wirelength, heavily connected modules should be placed close to one another. This can be achieved by minimizing the cut cost, because it will force heavily connected modules to be on the same side of a cut. Then, they will be placed in the same subregion. Alternately, cut cost minimization can be viewed as a way to minimize the routing congestion across the cutline.

11.3.2.2 *Terminal propagation technique*

One issue with recursive partitioning is that each subcircuit has not only nets internal to it but also nets connected to external modules. The effect of external nets should also be considered during the partitioning of a subcircuit. Dunlop and Kernighan [Dunlop 1985] developed a technique called **terminal propagation** to handle external nets. The technique is illustrated in Figure 11.7 and explained in the following.

Consider an example in Figure 11.7a in which subcircuit *A* has a net connecting module *p* to an external module *q*. To take into account the effect of

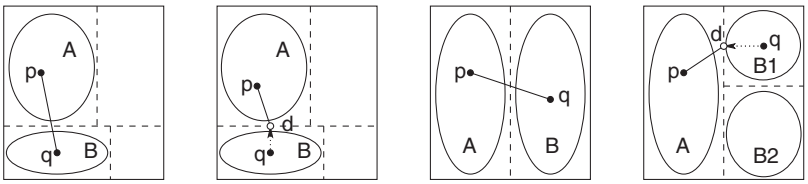

FIGURE 11.7

Terminal propagation.

this external net, assume module q is placed at the center of subregion B. The terminal in q is "propagated" to the closest point on the boundary of subregion A and is replaced by a **dummy terminal** d (as shown in Figure 11.7b). During the partitioning of A, the net between p and d is treated as an internal net and d is treated as a fixed terminal of subcircuit A. For this example, if subregion A is divided by a horizontal cutline, module p will be biased toward the lower partition because of the net between p and d. However, if subregion A is divided by a vertical cutline, d will be very near to the cutline. In this case, it is not clear whether the external net should bias p to the left or to the right partition. It is suggested in [Dunlop 1985] that dummy terminals that are within the middle third of the side should be ignored.

Another example is shown in Figure 11.7c. Assume both subregions A and B are divided by horizontal cutlines. Suppose subcircuit B is partitioned first. During the partitioning of B, the dummy terminal caused by p is too close to the cutline and hence is ignored. Suppose module q is assigned to the top subregion $B1$. Then during the partitioning of A, a dummy terminal d is introduced near to the top of subregion A (as shown in Figure 11.7d). Thus, it will bias p to the top.

11.3.3 Practical implementations

In the past, partitioning-based placement was generally perceived to be simple and efficient but not as good as simulated annealing or analytical approaches in terms of solution quality. In late-1990s, partitioning algorithms dramatically improved because of the multilevel scheme. Partitioning-based placement should also improve as a result. Capo [Caldwell 2000] and Fengshui [Yildiz 2001a, 2001b; Agnihotri 2003] are two placement algorithms that leverage this breakthrough in partitioning. They demonstrated that a careful implementation of the partitioning-based approach could produce very competitive wirelength with a relatively short runtime.

11.3.3.1 *The Capo algorithm*

In Capo, different bipartitioning techniques are applied to subcircuits of different sizes during recursive bipartitioning. For instances with more than 200

modules, a multilevel FM algorithm is used. For instances with 35 to 200 modules, the flat FM algorithm is used. Smaller instances are solved optimally with branch-and-bound.

In addition, much attention has been paid in Capo to the handling of partitioning tolerance. First, an "uncorking" technique is proposed to prevent large modules from being the first modules in a bucket of the FM algorithm. In an ordinary FM implementation, a large module at the head of a bucket may fail to move to the other partition without violating constraints on partition sizes. Smaller modules further in the same bucket will not be considered for moves, because the bucket is temporarily invalidated. This "corking" effect may degrade solution quality of FM. Second, repartitioning, which refers to a chained FM calls on the same partitioning instance, is presented. The first call is performed with a much larger tolerance than requested to ensure mobility of all modules. In subsequent calls, the tolerance gradually decreases to the original value. Third, a high tolerance of $\varepsilon = 20\%$ is used for vertical cutlines, because the cutline locations can be adjusted after partitioning according to sizes of partitions. However, this technique cannot be applied to horizontal cutlines, because their locations are more discrete (*i.e.*, aligned to standard-cell rows) and cannot be easily adjusted. Fourth, a formula is derived to determine the tolerance on the basis of the amount of white space in an instance. An instance with more white space will have a larger partitioning tolerance.

11.3.3.2 *The Fengshui algorithm*

The overall scheme of Fengshui is very similar to that of Capo. Several major differences are outlined in the following. In [Yildiz 2001a], instead of locally optimizing each subcircuit by bipartitioning, all subcircuits are partitioned simultaneously. The problem is formulated as multiway partitioning. Moreover, the partitioning cost function is HPWL rather than min-cut. In [Yildiz 2001b], a dynamic programming approach is presented to select a good sequence of cutlines. The sequence is optimal under certain assumptions. In [Agnihotri 2003], a fractional cut approach is introduced. In this approach, horizontal cutlines are not required to be aligned with standard-cell row boundaries. To handle the assignment of cells to rows that may be partially covered by a region, a dynamic programming–based legalization algorithm is developed. As fewer constraints are imposed on partitioning for horizontal cutlines, the wirelength can be reduced.

11.4 GLOBAL PLACEMENT: SIMULATED ANNEALING APPROACH

Simulated annealing is introduced in Section 4.4.4. It is an iterative heuristic for solving combinatorial optimization problems. The basic idea of simulated annealing is to search for a **configuration** with low cost by iteratively moving from the current configuration to a neighbor configuration. If the cost of the

neighbor configuration is lower than that of the current configuration, the **move** will be taken. Otherwise (*i.e.*, the move causes an increase in the cost), the move may still be taken with a probability that is decreasing over time according to a **cooling schedule**. This probabilistic move helps the search procedure to get out of a local minimum.

Simulated annealing is very popular, because it is a very robust technique that can be easily applied to virtually any optimization problem. To design a simulated annealing based algorithm for a given problem, one simply needs to define the configuration space, several types of moves, the cooling schedule, and the cost function. However, simulated annealing based algorithms are usually comparatively slow, especially for large problem instances. The main challenge of designing a simulated annealing based algorithm is to make it efficient without compromising the solution quality. Two simulated annealing based placement algorithms will be described in the following. Readers may also find the discussions on simulated annealing based floorplanning algorithms in Chapter 10 useful.

11.4.1 The placement algorithm in TimberWolf

Simulated annealing based placement algorithm was popularized in the mid-1980s by the TimberWolf place-and-route package [Sechen 1986]. The TimberWolf standard-cell placement algorithm consists of two stages. Stage 1 allows overlaps among cells and movement of cells between rows. Stage 2 eliminates all overlaps and only performs interchange of adjacent cells. The details are described in the following.

11.4.1.1 *Stage 1*

In stage 1, a configuration is an arrangement of the cells into the standard-cell rows possibly with cell overlaps. Three moves are defined:

M1: Move a cell to a new location, which can be in a different row.
M2: Swap two cells, which can be in different rows.
M3: Mirror a cell's x-coordinates.

The three moves are selected randomly with unequal probability. In each step, a selection between M1 and M2 is first made, with M1 four times more likely than M2. If M1 is selected but the new configuration is rejected, then M3 will be attempted for the same cell with a probability of 1/10. The applicable range for M1 and M2 is specified by a rectangular window called range limiter. For M1, the window is centered at the center of the randomly selected cell. A random location within the window will be chosen as the destination of the cell. For M2, a swap will be attempted only if the window can be positioned such that it contains both centers of the two randomly selected cells. At the beginning of stage 1, the horizontal span and vertical span of the window are equal to twice the horizontal span and vertical span of the chip, respectively. (Therefore, if the center of the window is

positioned at a corner of the chip, the window will still cover the entire chip.) During the annealing process, the horizontal span and vertical span of the window decrease slowly in proportion to the logarithm of the temperature.

The cost function has three components:

$$C = C_1 + C_2 + C_3$$

The first component, C_1, is an estimation of the total interconnect cost. For a net e, let w_e and h_e be the width and height of its bounding box, and β_e and γ_e be user-specified horizontal and vertical weights. Then,

$$C_1 = \sum_e (\beta_e \omega_e + \gamma_e h_e)$$

The second component, C_2, is an overlap penalty function. Let LinearOverlap(i, j) be the amount of overlap of cells i and j in the x-direction. Then,

$$C_2 = \sum_{i \neq j} \text{LinearOverlap}(i, j)^2$$

The third component, C_3, is a penalty function that serves to control the row lengths. For each row r, let $d(r)$ be the desired row length and $l(r)$ be the sum of the widths of the cells in row r. Then,

$$C_3 = \theta \times \sum_r |l(r) - d(r)|$$

where θ is a user-specified parameter.

11.4.1.2 *Stage 2*

When the vertical span of the range limiter window has been reduced to less than the center-to-center spacing between the rows, TimberWolf enters stage 2. At the beginning of stage 2, feed-through cells are inserted as required, and cell overlaps are eliminated by the following procedure. First, the cells in each row are sorted according to the x-coordinate of their centers. Then, they are re-placed side-by-side starting from the left edge of the row. After that, the simulated annealing continues.

In stage 2, the moves are more restrictive. M1 is not allowed. M2 considers swapping two adjacent cells only if they are in the same row. M3 is attempted only when M2 is attempted and rejected. In addition, the cost function is effectively just C_1. As there is no cell overlap, $C_2 = 0$. Because cells are not allowed to change rows, C_3 remains constant.

11.4.1.3 *Annealing schedule*

In the annealing schedule of TimberWolf, the initial temperature is 4,000,000. Then the temperature is decreased according to the following function:

$$T_{new} = \alpha(T_{old}) \times T_{old}$$

where α at different temperature is set according to a very specific table presented in [Sechen 1986]. Roughly, α starts at 0.8 when the temperature is high. Then it gradually increases as temperature decreases. It peaks at 0.94 when temperature is between 200 and 5000. After that, it steadily decreases to 0.7 as temperature drops. Finally, α is set to 0.1 when the temperature is below 1.5. The annealing process terminates when the temperature is less than 0.1. There are 117 temperature levels in this annealing schedule. At each temperature, a fixed number of moves are attempted. The number of moves per temperature is a function of the circuit size. For a 200-cell circuit, 100 moves per cell are recommended. Thus, 2.3×10^6 configurations have to be evaluated in total. For a 3000-cell circuit, 700 moves per cell are recommended. The number of configurations to be evaluated will increase dramatically to 245.7×10^6.

11.4.2 The Dragon placement algorithm

Simulated annealing based algorithms like TimberWolf can produce placement solutions of excellent quality for small circuits (with up to a few thousand cells). However, they tend to be increasingly inefficient for larger circuits. One way to improve their scalability is to perform placement in a hierarchical manner. A multilevel scheme (*i.e.*, bottom-up hierarchical scheme based on recursive clustering) is used in an improved version of TimberWolf [Sun 1995]. A top-down hierarchical scheme based on recursive partitioning is applied in the Dragon placement algorithm [Wang 2000]. The Dragon algorithm will be discussed in the following.

In fact, Dragon takes a hybrid approach that combines simulated annealing and partitioning. A hierarchy as shown in Figure 11.8 is formed by recursive quadrisectioning (*i.e.*, 4-way partitioning) by use of hMetis. At level b, the original circuit is partitioned into 4^b subcircuits. Correspondingly, the placement region is divided into a regular array of 4^b bins. Each subcircuit is assigned to one bin. Then low-temperature simulated annealing is applied to minimize the wirelength by swapping subcircuits among the bins. The recursive quadrisectioning terminates when a bin contains less than approximately 7 cells. Then low-temperature simulated annealing is again used to further reduce

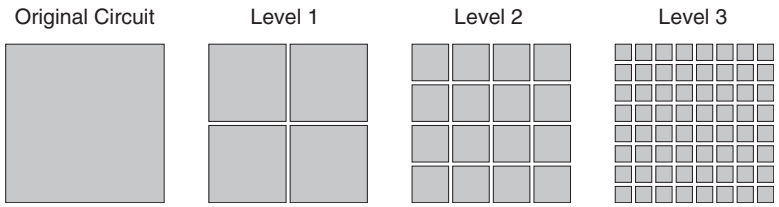

FIGURE 11.8

The top-down hierarchy used in Dragon.

wirelength by relocating a single cell to a different bin in each move. Finally, detailed placement is done by a greedy algorithm.

Compared with a flat annealing–based placement approach, annealing at high levels is swapping subcircuits among the bins. In other words, it is moving a large number of cells by a long distance in each move. It can be more efficient than swapping individual cells locally. Besides, quadrisectioning together with refinement at higher levels provides a good starting solution to simulated annealing and hence can shorten the annealing process significantly. Compared with a pure partitioning-based placement approach, the annealing-based swapping can correct wrong decisions made by quadrisectioning at higher levels. In addition, the swapping is based directly on wirelength rather than cut cost. Hence, it can generate higher-quality solutions than the partitioning-based approach alone.

11.5 GLOBAL PLACEMENT: ANALYTICAL APPROACH

The basic idea of the analytical approach is to express the cost function and the constraints as analytical functions of the coordinates of the modules. Then the placement problem is transformed into a mathematical program. To illustrate this approach, an exact, but impractical, formulation is first presented in Section 11.5.1. Practical analytical placement techniques can be classified as **quadratic** and **nonquadratic**, which are presented in Sections 11.5.2 and 11.5.3, respectively. Note that analytical placement techniques generally treat both standard cells and macros in the same manner. (Sometimes, special techniques are used to handle large macros to improve wirelength or to make legalization easier, but in theory, they are not essential.) Thus, the techniques presented in this section can be considered to be for mixed-size designs.

Some notations used in this section are introduced in the following. Consider a circuit with a set of modules V and a set of nets E is to be placed in a region of width W and height H. Assume the modules in V are indexed from 1 to n. For module $i \in V$, let w_i and h_i be its width and height, and let x_i and y_i be the x- and y-coordinates of its center. For each net $e \in E$, let c_e be its weight. Note that if a module i is a fixed module, then x_i and y_i are constants. Otherwise, they are variables. Assume for simplicity that all pins are located at the center of a module.

11.5.1 An exact formulation

It is instructive to first look at an exact formulation to the wirelength-minimized placement problem for mixed-size design. Then the rationales, pros, and cons of practical analytical placement techniques presented in Sections 11.5.2 and 11.5.3 can be better understood.

The HPWL of net $e \in E$ can be written as:

$$\text{HPWL}_e(x_1, \ldots, x_n, y_1, \ldots, y_n) = \left(\max_{i \in e}\{x_i\} - \min_{i \in e}\{x_i\}\right) + \left(\max_{i \in e}\{y_i\} - \min_{i \in e}\{y_i\}\right)$$

To express the nonoverlapping constraints among modules, it is necessary to introduce the following function:

$$\Theta([L1, R1], [L2, R2]) = [\min(R1, R2) - \max(L1, L2)]^+$$

where

$$[z]^+ = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

$\Theta([L1, R1], [L2, R2])$ gives the length of the overlapping region of the intervals $[L1, R1]$ and $[L2, R2]$ as illustrated in Figure 11.9. Then the overlapping area between module i and module j is given by:

$$\text{Overlap}_{ij}(x_i, y_i, x_j, y_j) = \Theta\left(\left[x_i - \frac{\omega_i}{2}, x_i + \frac{\omega_i}{2}\right], \left[x_j - \frac{\omega_j}{2}, x_j + \frac{\omega_j}{2}\right]\right) \Theta\left(\left[y_i - \frac{h_i}{2}, y_i + \frac{h_i}{2}\right], \left[y_j - \frac{h_j}{2}, y_j + \frac{h_j}{2}\right]\right)$$

The placement problem can be written as the following mathematical program:

$$\begin{aligned} \text{Minimize} \quad & \textstyle\sum_{e \in E} c_e \times \text{HPWL}_e(x_1, \ldots, x_n, y_1, \ldots, y_n) \\ \text{Subject to} \quad & \text{Overlap}_{ij}(x_i, y_i, x_j, y_j) = 0 \text{ for all } i, j \in V \text{ s.t. } i \neq j \\ & 0 \leq x_i - \frac{\omega_i}{2}, x_i + \frac{\omega_i}{2} \leq W \text{ for all } i \in V \\ & 0 \leq y_i - \frac{h_i}{2}, y_i + \frac{h_i}{2} \leq H \text{ for all } i \in V \end{aligned}$$

This mathematical program is extremely difficult to handle. The functions $\text{Overlap}_{ij}(x_i, y_i, x_j, y_j)$ for $i, j \in V$ are highly nonconvex and not differentiable. The functions $\text{HPWL}_e(x_1, \ldots, x_n, y_1, \ldots, y_n)$ for $e \in E$, although convex, are

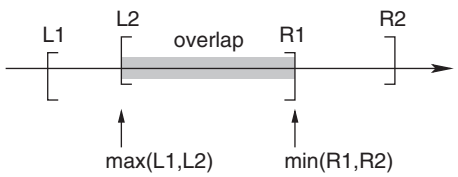

FIGURE 11.9

The overlapping region of the intervals $[L1, R1]$ and $[L2, R2]$.

not differentiable. Moreover, there are $O(n^2)$ constraints with the number of modules n being up to millions in modern designs. Therefore, this is not a practical formulation. In practice, the wirelength is approximated by some differentiable and convex functions. The nonoverlapping constraints are usually replaced by some simpler constraints to make the module distribution roughly even. Then legalization is performed to eliminate the module overlaps, and detailed placement is applied to refine the solution with a more accurate wirelength metric. Various techniques are presented in the remainder of this section.

11.5.2 Quadratic techniques

In quadratic placement techniques, the placement problem is transformed into a sequence of **convex quadratic programs**. Convex quadratic program is a mathematical program with a convex and quadratic objective function and linear constraints.

11.5.2.1 *Quadratic wirelength*

First, the way to express the placement cost function as a quadratic function is presented. Suppose for the time being that all nets in the circuit are 2-pin nets. (Multi-pin nets will be discussed later.) Consider a net $\{i, j\}$ (*i.e.*, connecting module i and module j). Its wirelength is given by the Manhattan distance between the modules:

$$L_{\{i,\,j\}} = |x_i - x_j| + |y_i - y_j|$$

This is usually referred to as linear wirelength. However, this function is not differentiable. So a common idea is to consider the squared Euclidean distance between the modules instead:

$$\tilde{L}_{\{i,\,j\}} = (x_i - x_j)^2 + (y_i - y_j)^2$$

This is usually called **quadratic wirelength**. To help visualize the functions, the x-components of $L_{\{i,j\}}$ and $\tilde{L}_{\{i,j\}}$ with a fixed value of $x_j = 2$ are plotted as functions of x_i in Figure 11.10.

In quadratic placement techniques, it is more convenient to set the cost function $\tilde{L}$ to be half[2] of the total weighted quadratic wirelength:

$$\tilde{L} = \frac{1}{2} \sum_{1 \le i < j \le n} c_{\{i,\,j\}} \times \left((x_i - x_j)^2 + (y_i - y_j)^2 \right)$$

[2]Half of the total weighted quadratic wirelength is used so that the derivatives will have simpler forms.

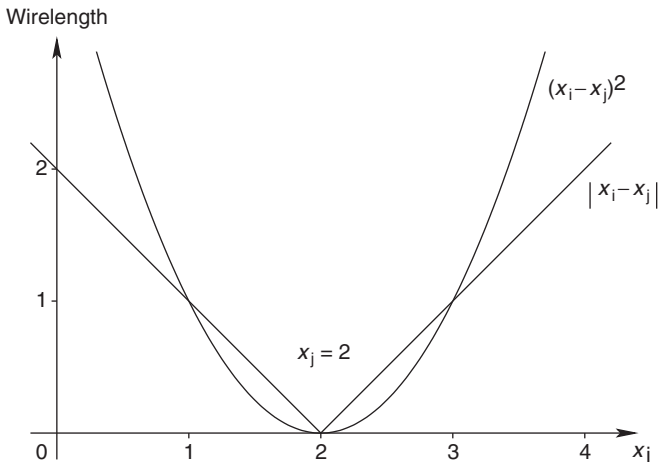

FIGURE 11.10

Comparison of quadratic wirelength and linear wirelength.

$\tilde{L}$ can be written succinctly in matrix form in terms of the coordinates of movable modules. In the following, assume modules 1 to r are movable and modules $r+1$ to n are fixed. Therefore, $x_1, \ldots, x_r$ and $y_1, \ldots, y_r$ are variables, and $x_{r+1}, \ldots, x_n$ and $y_{r+1}, \ldots, y_n$ are constants. Let $\boldsymbol{x} = (x_1 x_2 \cdots x_r)^T$ and $\boldsymbol{y} = (y_1 y_2 \cdots y_r)^T$ be the vectors of x-coordinate and y-coordinate of movable modules, respectively. Let $\boldsymbol{D} = (d_{ij})_{r\times r}$ be a diagonal matrix such that $d_{ii} = \sum_{j\in V} c_{\{i,j\}}$ for all $i \in \{1, \ldots, r\}$. Let $\boldsymbol{C} = (c_{ij})_{r\times r}$ be the connectivity matrix among movable modules, *i.e.,* $c_{ij} = c_{ji} = c_{\{i,j\}}$ for all $i, j \in \{1, \ldots, r\}$. Let $\boldsymbol{Q} = \boldsymbol{D} - \boldsymbol{C}$. Let $\boldsymbol{d}_x = (d_{x_1} \cdots d_{x_r})^T$ such that $d_{x_i} = -\sum_j \in \{r+1, \ldots, n\} c_{ij} x_j$ and $\boldsymbol{d}_y = (d_{y_1} \cdots d_{y_r})^T$ such that $d_{y_1} = -\sum_j \in \{r+1, \ldots, n\} c_{ij} y_j$. Then

$$\left[\tilde{L} = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x{}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{Q}\boldsymbol{y} + \boldsymbol{d}_y{}^T\boldsymbol{y} + \text{constant terms}\right]$$

For example, for the circuit with 3 movable modules and 3 fixed modules as represented by the graph in Figure 11.11,

$$\begin{aligned}
\tilde{L} = \; & \frac{1}{2}(c_{12}((x_1 - x_2)^2 + (y_1 - y_2)^2) \\
& + (c_{13}((x_1 - x_3)^2 + (y_1 - y_3)^2) \\
& + (c_{14}((x_1 - x_4)^2 + (y_1 - y_4)^2) \\
& + (c_{24}((x_2 - x_4)^2 + (y_2 - y_4)^2) \\
& + (c_{25}((x_2 - x_5)^2 + (y_2 - y_5)^2) \\
& + (c_{36}((x_3 - x_6)^2 + (y_3 - y_6)^2) \\
= \; & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{Q}\boldsymbol{y} + \boldsymbol{d}_y^T\boldsymbol{y} \\
& + \frac{1}{2}((c_{14} + c_{24})x_4^2 + c_{25}x_5^2 + c_{36}x_6^2 + (c_{14} + c_{24})y_4^2 + c_{25}y_5^2 + c_{36}y_6^2)
\end{aligned}$$

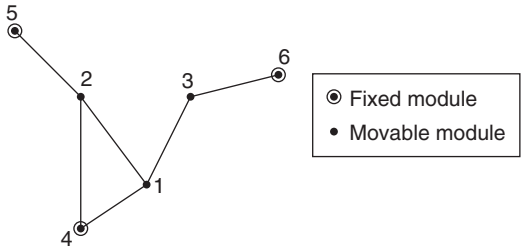

FIGURE 11.11

Connections of a circuit.

where

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \ \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix},$$

$$\boldsymbol{Q} = \begin{pmatrix} c_{12} + c_{13} + c_{14} & 0 & 0 \\ 0 & c_{21} + c_{24} + c_{25} & 0 \\ 0 & 0 & c_{31} + c_{36} \end{pmatrix} - \begin{pmatrix} 0 & c_{12} & c_{13} \\ c_{21} & 0 & c_{23} \\ c_{31} & c_{32} & 0 \end{pmatrix}$$

$$\boldsymbol{d}_x = \begin{pmatrix} -c_{14}x_4 \\ -c_{24}x_4 - c_{25}x_5 \\ -c_{36}x_6 \end{pmatrix} \text{and } \boldsymbol{d}_y = \begin{pmatrix} -c_{14}y_4 \\ -c_{24}y_4 - c_{25}y_5 \\ -c_{36}y_6 \end{pmatrix}$$

It is clear that $\tilde{L}_{\{i,j\}}$ for all $\{i, j\} \in E$ are convex and continuously differentiable functions. Hence, $\tilde{L}$, which is a weighted sum of $\tilde{L}_{\{i,j\}}$'s, is also convex and differentiable. So $\tilde{L}$ should be easy to minimize. In particular,

$$\frac{\partial \tilde{L}}{\partial \boldsymbol{x}} = \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x \text{ and } \frac{\partial \tilde{L}}{\partial \boldsymbol{y}} = \boldsymbol{Q}\boldsymbol{y} + \boldsymbol{d}_y$$

Therefore, the placement with minimum wirelength is given by

$$\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x = 0 \text{ and } \boldsymbol{Q}\boldsymbol{y} + \boldsymbol{d}_y = 0 \tag{11.1}$$

In other words, if nonoverlapping constraints are ignored, the quadratic placement problem is equivalent to solving a system of linear equations. If all movable modules are connected to fixed modules either directly or indirectly, $\boldsymbol{Q}$ is positive definite and thus invertible. This implies the existence of a unique global optimal solution. The simplicity of quadratic formulation is the main reason for its popularity. Note that $\boldsymbol{x}$ and $\boldsymbol{y}$ can be solved independently. For brevity's sake, sometimes only the x-component will be discussed from now on.

11.5.2.2 *Force interpretation of quadratic wirelength*

The problem of quadratic wirelength minimization can also be interpreted as a classical mechanics problem of finding the equilibrium configuration for a system of objects attached to zero-length springs. Consider each circuit module as an object and each 2-pin net $\{i, j\}$ as a stretched spring with spring constant $c_{\{i,j\}}$ connecting object i and object j. For the circuit represented by Figure 11.11, the corresponding spring system is shown in Figure 11.12.

The potential energy stored in spring $\{i, j\}$ is:

$$\varepsilon_{\{i,j\}} = \frac{1}{2} \times c_{\{i,j\}} \times (\text{Length of spring}\{i,\ j\})^2$$

$$= \frac{1}{2} \times c_{\{i,j\}} \times \left((x_i - x_j)^2 + (y_i - j_j)^2 \right)$$

Hence, the total potential energy of the spring system is equal to $\tilde{L}$. In other words, finding the minimum energy configuration for the spring system is equivalent to minimizing the quadratic wirelength in the quadratic placement formulation.

For a spring system, the minimum energy configuration is also the same as the force-equilibrium configuration. Note that the gradient of the total potential energy to the coordinates of an object gives the total force acting on the object. Therefore, the entries in the vectors $\boldsymbol{Qx} + \boldsymbol{d_x}$ and $\boldsymbol{Qy} + \boldsymbol{d_y}$ are the x-components and y-components of the total forces acting on the objects. In other words, another interpretation of the optimal placement conditions in Equation (11.1) is that all objects are in force equilibrium. For a nonequilibrium system (*i.e.*, a circuit placement with suboptimal quadratic wirelength), the total force on an object provides the best way of movement for the object to minimize the total energy (*i.e.*, the total weighted quadratic wirelength). Extra forces can also be added to influence the placement in a desirable manner (*e.g.*, to spread out the objects). Many quadratic placement algorithms use the guidance provided by springs/extra forces to optimize a placement solution (*e.g.*, [Quinn 1975]). Those algorithms are often called **force-directed placement algorithms**.

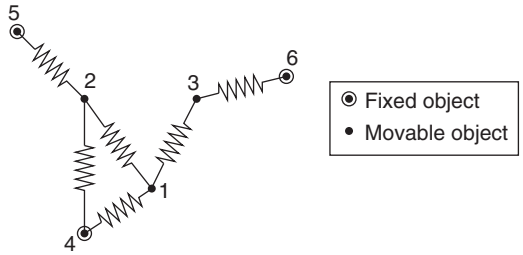

FIGURE 11.12

The spring system corresponding to the example in Figure 11.11.

The forces exerted by the springs are given by Hooke's law. The force exerted on object i by a spring connecting objects i and j (as illustrated in Figure 11.13) is:

$$\boldsymbol{F}_{ij} = c_{\{i,j\}} \times \text{Displacement from object } i \text{ to object } j$$

Its magnitude is:

$$\left|\boldsymbol{F}_{ij}\right| = c_{\{i,j\}} \times \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$$

To find the total force exerted by several springs on an object, it is often more convenient to decompose the force by each spring into x- and y-components:

$$\left|\boldsymbol{F}_{ij}^x\right| = c_{\{i,j\}} \times \left|x_j - x_i\right| \text{ and}$$
$$\left|\boldsymbol{F}_{ij}^y\right| = c_{\{i,j\}} \times \left|y_j - y_i\right|$$

Then the x-component and y-component of the total force are the sum of the x-component and y-component of the forces by all springs, respectively.

11.5.2.3 *Net models for multi-pin nets*

Circuits typically contain both 2-pin nets and multi-pin nets. To place circuits with multi-pin nets by quadratic techniques, various models have been proposed to replace each net by a group of 2-pin nets.

The traditional model is to replace each net by a **clique** (*i.e.*, complete graph). For example, for the 5-pin net in Figure 11.14a, the clique model is shown in Figure 11.14b. The net weights of the 2-pin nets in the clique model should be set properly to balance the minimization of 2-pin nets and multi-pin nets. For a k-pin net with net weight c, the weight of each 2-pin net in the clique is usually set to either $c/(k-1)$ [Vygen 1997] or $2c/k$ [Kleinhans 1991; Eisenmann 1998].

Another model is the **star model** [Vygen 1997; Mo 2000] as illustrated in Figure 11.14c. In the star model, one extra dimensionless module called the star module is introduced for each net. The star module is placed together with other movable modules during placement. Therefore, two extra variables corresponding to the x- and y-coordinates of the star module are added to the placement problem.

It is proved in [Viswanathan 2004] that the clique model and the star model are equivalent in quadratic placement if the net weights are set properly.

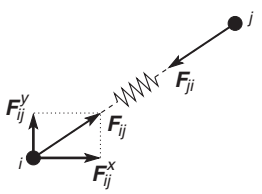

FIGURE 11.13

The forces by a stretched spring connecting objects i and j.

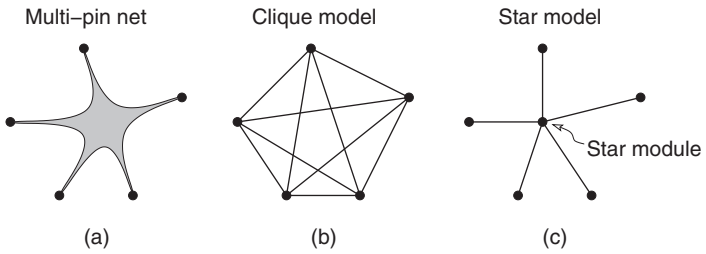

FIGURE 11.14

Clique and star models for multi-pin nets.

Specifically, consider a k-pin net connecting modules $1, \ldots, k$. In the clique model, if the weight of each 2-pin net is set to c, then the total force on module i by the 2-pin nets in the clique is given by:

$$F_i^{clique} = c \times \sum_{j=1}^{k} (x_j - x_i)$$

In the star model, let x_s be the x-coordinate of the star module. If the weight of each 2-pin net is set to $k \times c$, then the total force on the star node is given by:

$$F_s = \sum_{j=1}^{k} k \times c \times (x_j - x_s)$$

By setting $F_s = 0$, the force-equilibrium position for the star node is:

$$x_s = \frac{1}{k} \sum_{j=1}^{k} x_j$$

The force on module i by the 2-pin net connecting to the star is given by:

$$\begin{aligned} F_i^{star} &= k \times c \times (x_s - x_i) \\ &= k \times c \times \left(\frac{1}{k} \sum_{j=1}^{k} x_j - x_i \right) \\ &= c \times \left(\sum_{j=1}^{k} x_j - k \times x_i \right) \\ &= c \times \sum_{j=1}^{k} (x_j - x_i) \\ &= F_i^{clique} \end{aligned}$$

Because the forces exerted are the same, the clique and the star models are equivalent, and they can be used interchangeably in quadratic placement.

On the basis of the equivalence of clique and star models, the **hybrid net model** [Viswanathan 2004] is a natural choice. In the hybrid net model, the clique model is used for nets with 2 to 3 pins, and the star model is used for nets with 4 or more pins. It has been shown empirically that the hybrid net model reduces the number of 2-pin nets by more than 10× over the clique model for industrial circuits [Viswanathan 2007a]. It can also significantly reduce both the number of 2-pin nets and number of variables over the star model, because approximately 70% of nets in a typical circuit have 2 or 3 pins. Because the runtime to solve a quadratic placement problem is roughly proportional to the number of 2-pin nets and it increases slightly with the number of variables, the hybrid net model can speed up quadratic placement significantly.

11.5.2.4 *Linearization methods*

As shown in Figure 11.10, quadratic wirelength is a very rough approximation to linear wirelength. For small circuits, despite the inaccuracy of quadratic wirelength, quadratic placement techniques can still generate very competitive solutions. For larger circuits, however, this inaccuracy is a major bottleneck to the quality of quadratic placement solutions.

The authors in [Sigl 1991] presented a method to approximate the linear wirelength in a quadratic placement framework by iteratively adjusting the spring constant. Assume the star model is used to replace multi-pin nets. For a net e in the original circuit, let x_e be the coordinate of the associated star module. Then the total linear wirelength (for the circuit after applying the star model) can be written as:

$$L^{star} = \sum_{e \in E} \sum_{i \in e} |x_i - x_e|$$

Consider the function:

$$\tilde{L}^{star} = \sum_{e \in E} \sum_{i \in e} \frac{(x_i - x_e)^2}{g_{ie}}$$

If $g_{ie} = |x_i - x_e|$ then $\tilde{L}^{star} = L^{star}$. However, g_{ie}'s are set to constants so that $\tilde{L}^{star}$ would become a quadratic function. To approximate L^{star} with $\tilde{L}^{star}$, the function $\tilde{L}^{star}$ is optimized iteratively such that g_{ie} in current iteration is set according to the coordinates of previous iteration. Intuitively, $1/g_{ie}$ can be viewed as a variable spring constant that decreases with increasing spring length. The iterative process terminates when the g_{ie} factors no longer change significantly.

Notice that even L^{star} is just a rough approximation to the total HPWL objective function. Thus instead of setting g_{ie} to $|x_i - x_e|$, an experimentally-verified net specific factor is used:

$$g_{ie} = \sum_{i \in e} |x_i - x_e| \text{ for all } i \in e$$

This choice has two advantages. First, the summation reduces the influence of nets with many connected modules and emphasizes most nets connecting only two or three modules. Second, the summation also prevents increasing the force on modules close to the star node by too much. According to HPWL, as long as a module is inside the net bounding box, it is not helpful to increase the force to pull it farther inside.

Nets becoming very short (*i.e.,* g_{ie} becoming very small) may cause numerical problems during the minimization of $\tilde{L}^{star}$. Therefore, g_{ie} is lower bounded (by the average module width for example) to ensure that g_{ie} will never be zero.

Spindler and Johannes [Spindler 2006] introduced a **BoundingBox net model**, which, when combined with the preceding wirelength linearization idea, can accurately model HPWL in a quadratic placement framework. In the BoundingBox net model, a multi-pin net is transformed into only a few characteristic 2-pin nets as illustrated in Figure 11.15a. It is different from the clique model in which all possible 2-pin nets are included as shown in Figure 11.15b. Consider a k-pin net. For a given placement, suppose the modules are indexed in ascending order of their x-coordinates. Therefore, module 1 is at the left boundary and module k is at the right boundary of the net's bounding box. All connections are joined to the two boundary modules. One 2-pin net is connecting modules 1 and k. Two 2-pin nets are connecting each of the remaining $k-2$ inner modules to module 1 and module k, respectively. The total number of 2-pin nets is $1 + 2(k-2)$. Let $N = \{\{1, k\}, \{1, 2\}, \{2, k\}, \{1, 3\}, \{3, k\}, \ldots, \{1, k-1\}, \{k-1, k\}\}$ be the set of 2-pin nets.

According to the BoundingBox net model, the wirelength $\tilde{L}^{BB}$ of the k-pin net is defined as:

$$\tilde{L}^{BB} = \frac{1}{2} \sum_{\{i,j\} \in N} \omega_{\{i,j\}} \times \left(x_i - x_j\right)^2$$

where

$$\omega_{\{i,j\}} = \frac{2}{k-1} \times \frac{1}{l_{\{i,j\}}}$$

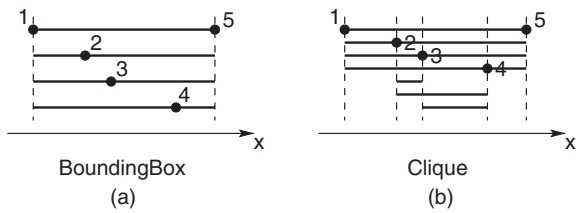

FIGURE 11.15

The BoundingBox and the clique net models for a 5-pin net.

If $l_{\{i,\,j\}}$ is set to $|x_i - x_j|$ for all $\{i, j\} \in N$, then:

$$\begin{aligned}
\tilde{L}^{BB} &= \frac{1}{2}\left(\sum_{\{i,j\}\in N} \frac{2}{k-1} \times \frac{1}{|x_i - x_j|} \times (x_i - x_j)^2\right) \\
&= \frac{1}{k-1}\left(\sum_{\{i,j\}\in N} |x_i - x_j|\right) \\
&= \frac{1}{k-1}\left(|x_1 - x_k| + \sum_{2\le i\le k-1} (|x_1 - x_i| + |x_i - x_k|)\right) \\
&= \frac{1}{k-1}(|x_1 - x_k| + (k-2) \times |x_1 - x_k|) \\
&= |x_1 - x_k|
\end{aligned}$$

Thus, the BoundingBox net model gives the exact HPWL if the net weights are set appropriately.

Like the linearization method in [Sigl 1991], the correct net weights $w_{\{i,\,j\}}$ are searched by iteratively optimizing $\tilde{L}^{BB}$ such that $l_{\{i,j\}}$ in current iteration is set to $|x_i - x_j|$ of previous iteration. In each iteration, the $l_{\{i,\,j\}}$ factors are constants and hence $\tilde{L}^{BB}$ becomes a quadratic function of module coordinates.

For simplicity, only the x-component of the model is described previously. The y-component can be constructed similarly. However, because the boundary modules and module distances may be different in x- and y-directions, the set of 2-pin nets introduced and the $l_{\{i,\,j\}}$ factors are most likely different. Moreover, even for a given direction, the boundary modules may change from iteration to iteration. Hence, the set of 2-pin nets and the net weights have to be updated continually. The overhead associated with maintaining two copies of connectivity matrices and the need to search for the net weights by an iterative process are the main disadvantages of the BoundingBox model over the clique, star, and hybrid models.

The BoundingBox net model has several advantages. First, it allows quadratic techniques to perform placement with the HPWL metric. Second, unlike the clique or star models, no connection is introduced among the inner modules to pull them together.[3] The inner modules are able to move more freely as in the HPWL model. Third, this model introduces much fewer 2-pin nets than the clique model. It does introduce more 2-pin nets than the star/hybrid model for nets with 4 or more pins, but the difference is not very significant.

Another way to mitigate the inaccuracy of quadratic wirelength is to correct the mistakes by refining the placement solution with some linear metrics. Detailed placement can be viewed as one example of this approach. In detailed placement, as a simple problem of locally rearranging a few modules is considered, an accurate wirelength model (*e.g.*, HPWL or even RSMT wirelength) can usually be applied. However, because corrections are restricted by the local

[3]In the star model, the inner modules are pulled together indirectly through the star module.

nature and the legality requirement of module movements, the effectiveness of detailed placement in optimizing the linear cost function is limited.

A better technique called ***iterative local refinement*** (ILR) is proposed in FastPlace [Viswanathan 2004]. ILR can be applied to any global placement solution before legalization. It works in iterations. In each iteration, the placement region is divided into bins by a regular grid structure. The bin size is large at the beginning iterations and is gradually reduced to consider progressively finer module movements. After binning, the modules are examined one by one. For each module, it is tentatively moved from its original bin to its eight adjacent bins as shown in Figure 11.16. For each tentative move, one score is computed. The score is a weighted sum of a wirelength component and a density component. The wirelength component is the total change in HPWL of all nets connected to the module. The density component is a function of the module densities of the original bin and the target bin. It rewards movements from a dense bin to a sparse bin. If all eight scores are negative, the module will remain unmoved. Otherwise, the move with the highest score will be taken. This iterative process is repeated until there is no significant improvement in wirelength.

Because ILR is not constrained by the nonoverlapping requirement and can move modules by a relatively long distance, it is much more effective than detailed placement in correcting major problems in the placement solution. It also helps the spreading of modules. Besides, it is an extremely fast technique because of its simplicity.

11.5.2.5 *Handling nonoverlapping constraints*

In placement, the two primary goals are to minimize the wirelength and to avoid module overlaps. These two goals are in conflict with each other. Wirelength minimization brings modules together. Overlap avoidance requires modules to spread out. Note that if the nonoverlapping constraints are ignored, for a circuit

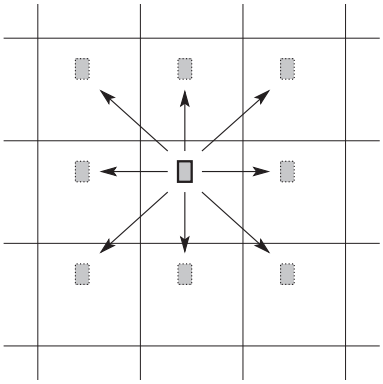

FIGURE 11.16

Eight tentative moves of iterative local refinement.

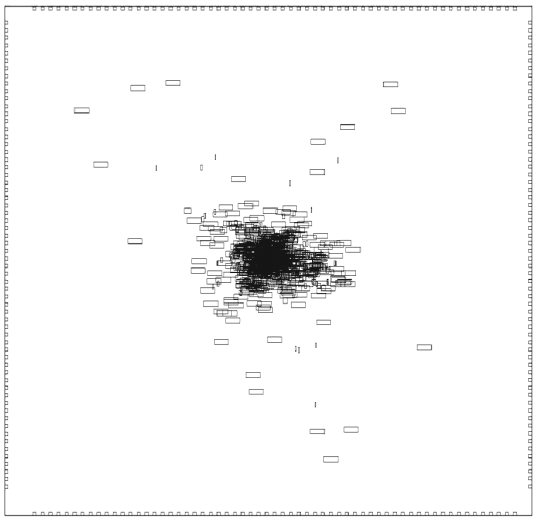

FIGURE 11.17

The placement solution for a circuit with fixed I/O pins at boundary when quadratic wirelength is minimized and nonoverlapping constraints are ignored.

without fixed modules, the optimal solution is to place all modules at the same location. This solution has zero wirelength but is meaningless because of the serious overlap issue. Even for a circuit with fixed modules (*e.g.*, I/O pins at boundary), which help pulling the movable modules away from each other, the wirelength-minimized placement without considering overlaps typically has a lot of overlaps at the center of the placement region as illustrated in Figure 11.17.

As pointed out in Section 11.5.1, instead of completely eliminating module overlaps in an analytical framework, an even distribution of modules is targeted in practice. In quadratic placement, there are basically two ways to make the module distribution more even. The first way is to add center-of-mass constraints to prevent modules from clustering together. The second way is to add forces to pull modules from dense regions to sparse regions. For both ways, the constraints/forces are added in an iterative manner to gradually spread out the modules. Note that even with the additional constraints/forces, the quadratic wirelength minimization problem can still be formulated as a convex quadratic program. Therefore, the circuit placement problem is transformed into a sequence of convex quadratic programs.

The technique of adding center-of-mass constraints is first introduced by GORDIAN [Kleinhans 1991]. Similar techniques are also used in BonnPlace [Brenner 2005] and hATP [Nam 2006]. The algorithm of GORDIAN is presented in the following. In GORDIAN, given an uneven quadratic placement solution, the module distribution can be improved by the following procedure. Assume the modules have to be spread horizontally. First, a vertical cutline is used to

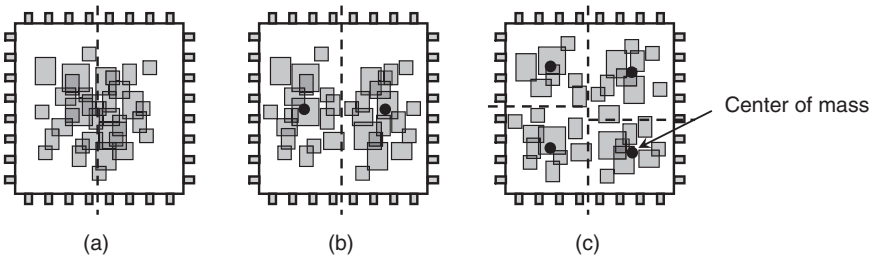

FIGURE 11.18

Module spreading by center-of-mass constraints in GORDIAN.

partition all modules into two subcircuits and the placement region into two subregions (see Figure 11.18a). Then, for each subcircuit, a constraint in the x-direction is added to force the center of mass of all its modules to be at the center of the corresponding region. Next, the placement problem with the two additional constraints is solved again. The center-of-mass constraints should pull the two subcircuits horizontally away from each other as shown in Figure 11.18b. This procedure is applied hierarchically to improve the distribution in each subregion (see Figure 11.18c) until each subcircuit contains less than a predefined number of modules. Note that at each hierarchical level, the placement of all subcircuits is considered together as a single global optimization problem.

The coordinates of the center of mass are the area weighted mean values (*i.e.*, linear functions) of the module coordinates. In other words, the center-of-mass constraints are linear equality constraints. Therefore, the global optimization problem at each hierarchical level is a convex quadratic program, which is equivalent to solving a system of linear equations.

Although the center-of-mass constraints help spreading, they hurt wirelength. For any two subcircuits belonging to the same parent in the hierarchy, the center-of-mass constraints draw them apart by a long distance. Hence the connections between them will become much longer. The wirelength impact can be minimized if the cut cost between the two subcircuits can be reduced. To avoid a large cut cost, both direction and position of the cutline are chosen carefully. To determine the cut cost of every possible partition by a vertical cutline, the cutline is scanning from left to right and the cut cost is updated whenever the cutline passes over a module. Only the partitions in which both subcircuits are at least 35% of the area of their parent are considered. The cut costs for horizontal cutline can be found similarly. The cutline with the smallest cut cost among all directions and positions is chosen. After that, the FM bipartitioning algorithm is optionally applied to refine the partition by moving modules that are close to the cutline. Moreover, after global optimization, if there are a lot of overlaps between two subcircuits, it indicates a bad bipartition, because many modules tend to migrate to the other region. In that case, they are repartitioned.

The technique of adding forces to spread modules in a quadratic placement framework was first introduced in Kraftwerk [Eisenmann 1998]. In Kraftwerk, density-based forces are derived to pull modules from high-density regions to low-density regions. However, constant forces are used, and the magnitude of the forces is set heuristically. As a result, the convergence is hard to control and the algorithm is not as fast as it should be. In the following, the improved technique in the new version of Kraftwerk [Spindler 2006] is presented. Note that the x-coordinates will be focused in the discussion following.

For a given placement, let $\boldsymbol{x}$ be the vector of current module positions and $\boldsymbol{x}$ be the vector of variables representing module positions in a new placement to be determined. The additional force for each module is separated into two components: **hold force** and **move force**. The hold force vector $\boldsymbol{F}_x^{hold}$ is defined as:

$$\boldsymbol{F}_x^{hold} = -(\boldsymbol{Q}\boldsymbol{x}' + \boldsymbol{d}_x)$$

It is used to counterbalance the total forces by the nets of the circuit in the current placement $\boldsymbol{x}'$. It makes sure that if the placement problem is solved again, all modules will be held in their current positions. Hold forces do not depend on $\boldsymbol{x}$ and hence are constant forces.

The move force is used to move a module toward less dense regions. Let $D(x, y)$ be the module density at location (x, y). It is defined as the number of modules that cover (x, y) minus the average density $(\sum_i w_i \times h_i)/(W \times H)$. The distribution $D(x, y)$ can be viewed as a charge distribution, which creates an electrostatic potential ϕ based on the Poisson equation:

$$\Delta\phi = -D(x, y)$$

The Poisson equation can be solved efficiently by geometric multigrid solvers (*e.g.*, [Kowarschik 2001]).

In the electrostatic formulation, the potential ϕ is high in regions where the distribution $D(x, y)$ is high, and *vice versa*. Hence the gradient of the potential $(\partial\phi/\partial x, \partial\phi/\partial y)^T$ can be used to move the modules away from high-density regions toward low-density regions and thereby reduce the overlaps among the modules. For each module i, its target position $\hat{x}_i$ is:

$$\hat{x}_i = x_i' - \left|\frac{\partial\Phi}{\partial x}\right|_{(x_i', y_i')}$$

Move forces are added to guide modules toward their target positions. They are generated by use of the **fixed-point** idea proposed in FAR [Hu 2002]. Each module i is connected to its target position (*i.e.*, a fixed point) by a spring with spring constant $\hat{c}_i$. So the move force vector $\boldsymbol{F}_x^{move}$ is given by:

$$\boldsymbol{F}_x^{move} = \hat{\boldsymbol{Q}}(\boldsymbol{x} - \hat{\boldsymbol{x}})$$

where $\hat{\boldsymbol{Q}} = \text{diag}(\hat{c}_i)$. Note that spring forces rather than constant forces are used for move forces so that module movements are limited. Each module can be

moved at most up to its target position. This helps the convergence of Kraftwerk significantly. The spring constants $\hat{c}_i$ control the tradeoff between rate of convergence and wirelength. If large $\hat{c}_i$ values are used, modules will be moved close to their target positions. Hence the placer will converge faster to an even density distribution. On the other hand, small $\hat{c}_i$ values allow module positions to be determined mostly by wirelength minimization.

In the new placement solution, the sum of net force, hold force, and move force on each module should be zero:

$$(\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{d}_x) - (\boldsymbol{Q}\boldsymbol{x}' + \boldsymbol{d}_x) + \hat{\boldsymbol{Q}}(\boldsymbol{x} - \hat{\boldsymbol{x}}) = 0$$

Therefore, the new module positions $\boldsymbol{x}$ can be found by solving the following system of linear equations:

$$(\boldsymbol{Q} + \boldsymbol{Q})(\boldsymbol{x} - \boldsymbol{x}') = -\hat{\boldsymbol{Q}}(\boldsymbol{x}' - \hat{\boldsymbol{x}})$$

This spreading procedure is repeated until the placement density distribution is even enough. It can be proved that this procedure always converges to an overlap-free placement.

Note that besides this potential-based method, the target positions can also be computed by simpler heuristics like **cell shifting** [Viswanathan 2004] or **grid warping** [Xiu 2004]. All these methods try to equalize the placement density of nearby regions by locally moving modules. There is no proof of convergence for these methods, but they work well in practice.

To mitigate the negative effect of the additional forces on wirelength in force-directed spreading algorithms, a **force-vector modulation** technique is proposed in RQL [Viswanathan 2007b]. The technique is based on the observation that the spreading force is small for most modules but very huge for a few percent of all modules. A huge force implies that the corresponding module is pulled away from its natural position (*i.e.*, the force-equilibrium position if spreading force is removed) by a long distance. Thus, the nets connecting to the module become very long. The force-vector modulation technique nullifies the huge forces before the next quadratic optimization iteration. As a result, modules with nullified spreading forces can return to the minimum-wirelength positions. Hence, the total wirelength can be significantly improved. Because the spreading forces of only a few percent of modules are nullified, module spreading is not seriously affected.

11.5.3 Nonquadratic techniques

Another category of analytical approach is to formulate the placement problem as a single nonlinear program as in Section 11.5.1. However, instead of exact wirelength metric and exact nonoverlapping constraints, approximations are used. In particular, the placement region is divided into bins, and the nonoverlapping constraints are replaced by bin density constraints. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the vectors of x- and y-coordinates of the modules, respectively. Then the problem can be formulated as follows:

$$\text{Minimize} \sum_{e \in E} c_e \times \text{WL}_e(\boldsymbol{x}, \boldsymbol{y})$$

$$\text{Subject to } D_b(\boldsymbol{x}, \boldsymbol{y}) = T_b \text{ for all bin } b$$

$\text{WL}_e()$ is a continuously differentiable function that may be more complicated than quadratic functions but may also be more accurate in approximating HPWL. T_b is the target density of bin b. $D_b()$ gives the density of bin b with respect to placement solution $\boldsymbol{x}$ and $\boldsymbol{y}$. The exact bin density function is a piece-wise linear function and hence is not differentiable. $D_b()$ is a smooth version of the exact one.

Examples of placers in this category are APlace [Kahng 2004], mPL [Chan 2005], and NTUPlace [Chen 2006]. To approximate the wirelength, APlace, mPL, and NTUPlace all use the **log-sum-exponential wirelength function** described in a patent by [Naylor 2001]. To smooth the density function, APlace and NTUPlace use a **bell-shaped function** proposed also by [Naylor 2001], and mPL uses **inverse Laplace transformation** [Evans 2002]. The wirelength approximation and density smoothing methods are described in the following.

11.5.3.1 *Log-sum-exponential wirelength function*

The log-sum-exponential function is defined as:

$$\text{LSE}_\alpha(z_1, \ldots, z_n) = \alpha \times \left(\log \left(\sum_{i=1}^{n} e^{z_i/\alpha} \right) \right)$$

It is an approximation of the maximum function:

$$\text{LSE}_\alpha(z_1, \ldots, z_n) \approx \max(z_1, \ldots, z_n)$$

α is a parameter controlling the accuracy of the approximation. As α converges to 0, the log-sum-exponential function converges to the maximum function. This is demonstrated in Figure 11.19, which shows for $\alpha = 0.1$ and for $\alpha = 2$, the log-sum-exponential function of two arguments.

The HPWL of a net e can be expressed in terms of the maximum function:

$$\begin{aligned} &\text{HPWL}_e(x_1, \ldots, x_n, y_1, \ldots, y_n) \\ &= \left(\max_{i \in e} \{x_i\} - \min_{i \in e} \{x_i\} \right) + \left(\max_{i \in e} \{y_i\} - \min_{i \in e} \{y_i\} \right) \\ &= \left(\max_{i \in e} \{x_i\} + \max_{i \in e} \{-x_i\} \right) + \left(\max_{i \in e} \{y_i\} + \max_{i \in e} \{-y_i\} \right) \end{aligned}$$

So HPWL can be approximated by the log-sum-exponential based function as follows:

$$\text{LSEWL}_{e,\alpha}(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

$$= \alpha \times \left(\log \left(\sum_{i \in e} e^{x_i/\alpha} \right) + \log \left(\sum_{i \in e} e^{-x_i/\alpha} \right) + \log \left(\sum_{i \in e} e^{y_i/\alpha} \right) + \log \left(\sum_{i \in e} e^{-y_i/\alpha} \right) \right)$$

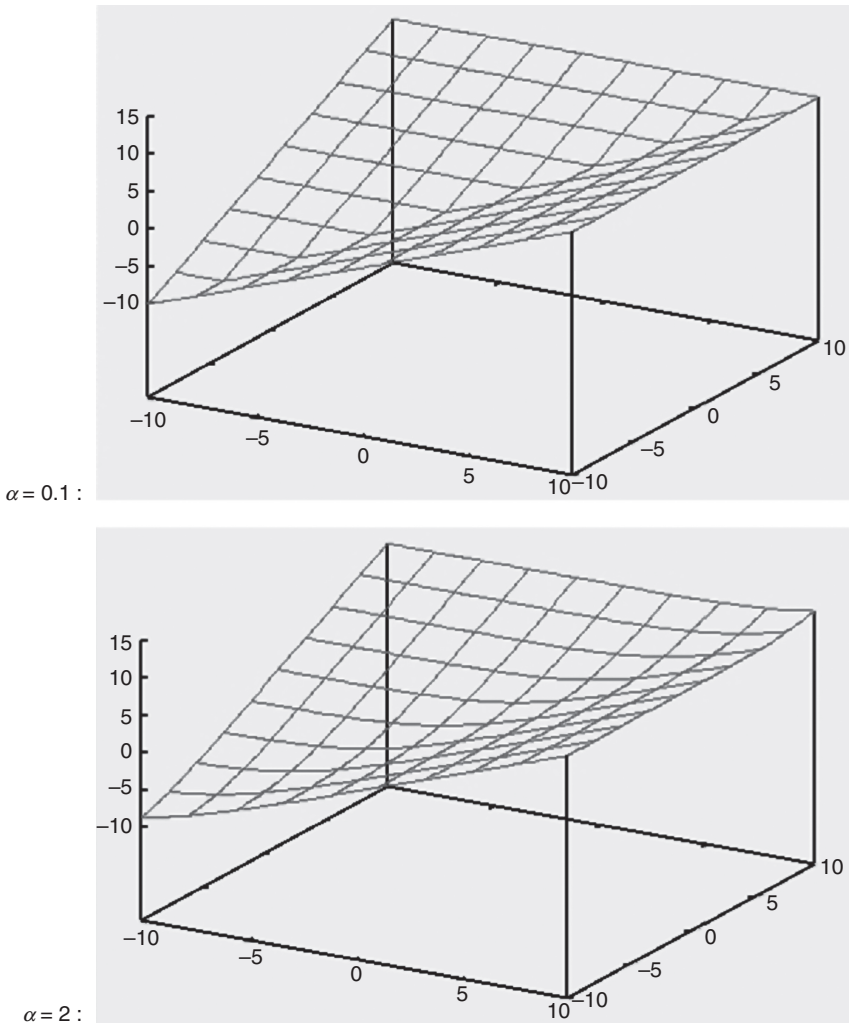

FIGURE 11.19

The log-sum-exponential function of two arguments for two different values of α.

$\text{LSEWL}_{e,\alpha}()$ is strictly convex, continuously differentiable, and converges to $\text{HPWL}_e()$ as α converges to 0 [Naylor 2001].

11.5.3.2 *Density constraint smoothing by bell-shaped function*

To illustrate the idea, assume for now that each module is much smaller than the bins and hence is considered to be a dot. Besides, assume each module has a unit area. For a bin b, let x_b be the x-coordinate of the center and w_b be the width of bin b. Then the overlap function $\Theta_x(b, i)$ in the x-direction

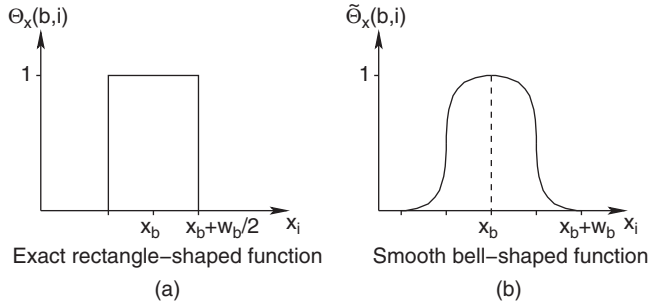

FIGURE 11.20

Overlap function between bin b and module i.

between bin b and module i is shown in Figure 11.20a. This function can be approximated by a smooth bell-shaped function $\tilde{\Theta}_x(b,i)$ as shown in Figure 11.20b. Let $d_x = |x_i - x_b|$. Then:

$$\tilde{\Theta}_x(b,i) = \begin{cases} 1 - 2 \times d_x^2/\omega_b^2 & \text{if } 0 \le d_x \le \omega_b/2 \\ 2 \times (d_x - \omega_b)^2/\omega_b^2 & \text{if } \omega_b/2 \le d_x \le \omega_b \\ 0 & \text{if } \omega_b \le d_x \end{cases} \tag{11.2}$$

The overlap function $\tilde{\Theta}_y(b,i)$ in the y-direction is defined similarly.

The density function of bin b can be written as follows:

$$D_b(x,\ y) = \sum\nolimits_{i \in V} C_i \times \tilde{\Theta}_x(b,i) \times \tilde{\Theta}_y(b,i)$$

where C_i is a normalization factor so that $\sum_b C_i \times \tilde{\Theta}_x(b,i) \times \tilde{\Theta}_y(b,i) = \omega_i \times h_i$ (*i.e.*, area of module i).

This idea can be extended to handle large modules [Kahng 2005]. For a module i with width w_i, the scope of the module in the x-direction is set to $w_b + w_i/2$ (*i.e.*, every bin within horizontal distance $w_b + w_i/2$ from the module's center is considered to be overlapping with the module). Then:

$$\tilde{\Theta}_x(b,i) = \begin{cases} 1 - a \times d_x^2 & \text{if } 0 \le d_x \le w_b/2 + w_i/2 \\ b \times (d_x - w_b - w_i/2)^2 & \text{if } w_b/2 + w_i/2 \le d_x \le w_b + w_i/2 \\ 0 & \text{if } w_b + w_i/2 \le d_x \end{cases} \tag{11.3}$$

where

$$a = 4/((w_b + w_i)(2w_b + w_i))$$

$$b = 4/(w_b(2w_b + w_i))$$

so that the function is continuous when $d_x = w_b/2 + w_i/2$. Note that Equation (11.3) is the same as Equation (11.2) if $w_i = 0$.

11.5.3.3 *Density constraint smoothing by inverse Laplace transformation*

Inverse Laplace transformation is a commonly used method to smooth functions. For a given placement solution $\boldsymbol{x}$ and $\boldsymbol{y}$ and for any location (x, y) in bin b, let $d(x, y)$ be the density at (x, y), *i.e.*, $d(x, y) = D_b(\boldsymbol{x}, \boldsymbol{y})$. The smoothing operator $\Delta_{\in}^{-1} d(x, y)$ is defined by solving the Helmholtz equation:

$$\begin{cases} \Delta\Psi(x, y) - \epsilon\Psi(x, y) = d(x, y) & (x, y) \in R \\ \dfrac{\partial\Psi}{\partial v} = 0 & (x, y) \in \partial R \end{cases} \tag{11.4}$$

where $\psi(x, y)$ is a smoothed version of $d(x, y)$, $\in > 0$ is a parameter controlling the smoothness, R is the placement region, ∂R is the boundary of R, v is the outer unit normal vector pointing outside the boundary, and $\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2$ is a differential operator.

The inverse operator $\Delta_{\epsilon}^{-1} d(x, y)$ is well defined as equation (11.4) has a unique solution for any $\epsilon > 0$. Because the solution of Equation (11.4) gains two more derivatives than $d(x, y)$, ψ is at least twice differentiable.

11.5.3.4 *Algorithms for nonlinear programs*

For nonquadratic techniques, as the objective function and all constraints are continuously differentiable, the resulting nonlinear program can be solved by any nonlinear programming algorithms. In APlace and NTUPlace, the nonlinear program is converted by the **quadratic penalty method** into a sequence of unconstrained minimization problems. Each unconstrained minimization problem has the following form:

$$\text{Minimize } \sum_{e \in E} c_e \times \text{WL}_e(\boldsymbol{x}, \boldsymbol{y}) + \beta \times \sum_b (D_b(\boldsymbol{x}, \boldsymbol{y}) - T_b)^2$$

The intuition of the quadratic penalty method is that any placement solution violating the density constraint for bin b will be charged a penalty of $(D_b(\boldsymbol{x}, \boldsymbol{y}) - T_b)^2$. β is a parameter to specify the importance of density constraints. Its value keeps increasing in the sequence of unconstrained problems to discourage uneven placement solutions. Each unconstrained problem is solved by the **conjugate gradient method** [Luenberger 1984].

In mPL, the nonlinear program is solved by the **Uzawa algorithm** [Arrow 1958] shown in Algorithm 11.2. In the Uzawa algorithm, $\boldsymbol{x}^k$ and $\boldsymbol{y}^k$ are the module locations at the k-th iteration, λ_b^k is the **Lagrange multiplier** at the k-th iteration, and α is a parameter to control the rate of convergence.

Algorithm 11.2 The Uzawa Algorithm

1. Initialize $\boldsymbol{x}^0$, $\boldsymbol{y}^0$, and λ_b^0 for all b
2. For $k = 0, 1, \ldots$
3. Find $\boldsymbol{x}^{k+1}$ and $\boldsymbol{y}^{k+1}$ by solving the following equality:

$$\sum\nolimits_{e \in E} c_e \times \nabla \mathrm{WL}_e\left(\boldsymbol{x}^{k+1},\ \boldsymbol{y}^{k+1}\right) + \sum\nolimits_b \lambda_b^k \times \nabla D_b\left(\boldsymbol{x}^k,\ \boldsymbol{y}^k\right) = 0$$

4. Set $\lambda_b^{k+1} = \lambda_b^k + \alpha \times \left(D_b\left(\boldsymbol{x}^{k+1},\ \boldsymbol{y}^{k+1}\right) - T_b\right)$ for all b

The nonquadratic techniques are elegant and comparable to the quadratic techniques in terms of wirelength. However, they are more complicated to implement and are usually more expensive computationally.

11.5.4 Extension to multilevel

To handle large-sized problems, a multilevel scheme is commonly used in analytical placement. The application of a multilevel scheme to placement is similar to its application to partitioning presented in Section 11.3.1. It consists of three phases. First, a hierarchy of coarser netlists is constructed by clustering heavily connected modules together. Second, an initial placement of the coarsest netlist is generated. Finally, the netlist is successively unclustered, and the placement at each level is refined. The multilevel scheme can improve both the runtime and the solution quality of analytical placement algorithms. Two popular clustering techniques for netlist coarsening are introduced in the following.

11.5.4.1 *First choice*

The **First Choice clustering technique** [Karypis 1997] first represents the netlist as a weighted graph by replacing the multi-pin nets with the clique model. The weight or **affinity** r_{ij} between any modules i and j in the graph is given by:

$$r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{|e| - 1}$$

Then the modules are traversed in an arbitrary order. Each module i is clustered with an unclustered neighbor j with the largest r_{ij}. After all modules are traversed, the affinity graph is updated, and the clustering process is repeated until the number of modules has reached the target. The intuition behind First Choice is that modules with high affinity should stay close together in a good placement solution.

First Choice is originally proposed for multilevel partitioning. Several modifications of First Choice targeting placement are presented in [Chan 2005]. To reduce variation in cluster size, the affinity between modules i and j is redefined as:

$$r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{(|e| - 1) \times \mathrm{area}(e)}$$

where area(e) is the total area of all modules in e. In addition, modules are visited in ascending order of module area (with preference to smaller module degree to break ties). This ordering is observed to balance the area of clusters better. If a good initial placement is provided, the proximity information between modules can be incorporated into the affinity as follows:

$$r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{(|e| - 1) \times \text{area}(e) \times \text{dist}(i,\, j)}$$

where dist(i, j) is the Euclidean distance between i and j.

11.5.4.2 *Best choice*

In the **Best Choice clustering technique** [Alpert 2005], the affinity is defined as:

$$r_{ij} = \sum_{e \in E \wedge i,\, j \in e} \frac{c_e}{|e| \times (\text{area}(i) \times \text{area}(j))}$$

where area(i) and area(j) are the areas of modules i and j, respectively. In addition to the indirect control of the cluster size by the affinity, Best Choice imposes a hard upper limit for cluster size. Moreover, the pair of modules with the largest affinity among all pairs is clustered and, in principle, the netlist is immediately updated. In other words, Best Choice always selects the globally best pair for clustering. In practice, as the immediate update is time-consuming, a **lazy updating technique** is proposed to reduce the runtime. The idea is that instead of explicitly recomputing the affinities of module pairs affected by a given cluster, they are marked as invalid and are updated only after they have been selected for clustering. Because the pair selection is based on invalid affinities, lazy updating may incur some errors. However, the dramatic reduction in runtime outweighs the small errors.

As with the modified First Choice affinity, the proximity information of a good initial placement can be incorporated into the Best Choice affinity in the same way.

11.6 LEGALIZATION

Given an illegal placement, legalization is a process to eliminate all overlaps by perturbing the modules as little as possible. In the partitioning-based approach, the modules in each subcircuit at the lowest level have to be arranged in the corresponding subregion. In the simulated annealing approach, it is possible that overlaps are allowed (but penalized) in intermediate steps. In analytical placement, the nonoverlapping constraints are always replaced by density constraints. Therefore, legalization is required for all three approaches.

For standard-cell placement, the Tetris legalization algorithm [Hill 2002] is very commonly used. In this algorithm, modules are first sorted in ascending x-coordinate. Then the modules are packed to the left one at a time into the row that minimizes the total displacement for that module. This simple greedy algorithm is extremely fast. However, it sometimes may result in very uneven

row lengths and hence may fail to pack all modules inside the placement region. This issue motivated a slight modification proposed in [Khatkhate 2004]. If the algorithm fails to pack all modules inside the placement region, the penalty for displacing a module in the vertical direction is gradually reduced. This modification encourages more even row lengths and so improves the chance of success. It is clear that the Tetris algorithm can also legalize mixed-size designs. It was suggested in [Khatkhate 2004] that the algorithm handles mixed-size designs well.

Because the Tetris algorithm is greedy in nature and attempts to pack modules to the left, it may perturb the original placement quite significantly. A more robust legalization method is proposed in [Ren 2005]. This method is based on a discrete approximation to a closed-form solution of the continuous diffusion equation. It generates a roughly legal placement. Then any legalizer can be applied to put modules onto rows without overlap. This diffusion-based method spreads the modules smoothly. Thus it helps preserve neighborhood characteristics of the original placement. As a result, the wirelength and timing of the resulting placement is better.

11.7 DETAILED PLACEMENT

Given a legalized placement solution, detailed placement further improves the wirelength (or other objectives) by locally rearranging the standard cells while maintaining legality. There may be significant room for wirelength improvement in a legalized global placement solution for several reasons. First, global placement typically uses inaccurate wirelength models (*e.g.*, cut cost, quadratic wirelength with clique net model, log-sum-exponential function). Second, global placement algorithms often place each cell into a subregion without paying much attention to the location of the cell within the subregion. Third, during legalization, the wirelength is likely to be worsened by the perturbations.

Simulated annealing can be easily adopted to perform detailed placement, but this technique is usually slow. Another technique is to iteratively consider different windows and use branch-and-bound to optimally rearrange the cells within each window [Caldwell 2000; Agnihotri 2003]. Because of the high computational complexity of branch-and-bound, a window can only contain up to 7 to 8 cells. Hence this technique is effective only in making very local modifications to the placement solution. Two detailed placement algorithms that work well in practice are presented in the following.

11.7.1 The Domino algorithm

A very high-quality yet efficient detailed placer is the Domino algorithm [Doll 1994]. Domino also uses a **sliding window approach** to iteratively refine a small region. For each region, the problem of assigning the cells to new locations is formulated as a **transportation problem**. To account for the different cell widths, each cell i with width w_i is divided into w_i unit-width subcells. Then

the problem is to simultaneously transport the subcells to unit-width locations in an overlap-free manner that minimizes a cost function approximating wirelength. This problem can be transformed into a **minimum cost maximum flow problem** on a network as shown in Figure 11.21. This network consists of a source node S, a set of cell nodes i, a set of location nodes k, and a destination node D. The capacity of the arc between node S and cell node i is w_i. Because each location can hold at most one subcell, all capacities of arcs leading from location nodes to node D are set to one. The cost of assigning a subcell of cell i to location k is c_{ik}, which is determined by a net model described later.

By solving the network flow problem, the subcells are assigned to locations. For all subcells of each cell, as they are associated with the same transportation cost and they are pulled towards the cheapest location by the transportation algorithm, they tend to lie side by side. Each cell is placed in the row holding most of its subcells. The x-coordinate of the cell is determined by the center of gravity of the subcells. Finally, the cells in each row of the region are packed according to their x-coordinates to prevent overlap and unused space.

The cost c_{ik} of assigning a subcell of cell i to location k is the total HPWL of all nets connected to cell i. During the evaluation of the cost c_{ik}, cell i is assumed to be at location k. However, the locations of other cells in the region are still unknown. Hence the HPWL of nets are estimated according to the following net model. To estimate the HPWL of a net e connected to cell i, let e_I be the subset of cells connected to net e inside the region. Consider the following three cases:

- Case 1: $|e_I| = 1$.
  In this case, the only cell inside the region that net e connects to is cell i. Therefore, the HPWL of net e can be calculated exactly.
- Case 2: $1 < |e_I| < |e|$.
  In this case, net reconnects to some cell(s) other than cell i both inside and outside the region. The unknown locations of cells in $e_I - \{i\}$ are estimated by their coordinates in the current placement. Then the HPWL of net e can be calculated.
- Case 3: $|e_I| = |e|$.
  In this case, net e connects only to cells inside the region. Again, the locations of cells in $e_I - \{i\}$ are estimated by their coordinates in the

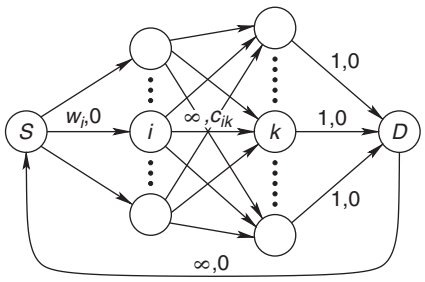

FIGURE 11.21

Transportation network in Domino. Arcs are labeled with "capacity, cost."

current placement. Besides, a virtual cell is introduced at the center of gravity of the cells in e_I with respect to the current placement. The HPWL of all cells in e together with the virtual cell is used as an estimate of the HPWL of net e.

An advantage of Domino over branch-and-bound based algorithms is that the network flow problem has a much lower computational complexity and hence much larger windows can be used. A larger window allows more cells to be placed simultaneously and potentially improves the wirelength. However, it also increases the runtime and results in a less accurate estimation of the HPWL in the cost function. In practice, a window size of approximately 20 to 30 cells per region yields a good tradeoff between wirelength and runtime.

The preceding description is a brief outline of the main ideas of the Domino algorithm. Another net model and theoretical analysis of the relations between the two net models and HPWL are presented in [Doll 1994]. Interested readers may refer to the original paper.

11.7.2 The FastDP algorithm

The FastDP algorithm [Pan 2005] is a greedy heuristic that can generate slightly better solutions than Domino and is an order of magnitude faster. The FastDP algorithm consists of four key techniques: **global swap, vertical swap, local reordering**, and **single-segment clustering**. The flow of FastDP is given in Algorithm 11.3.

Algorithm 11.3 The FastDP Detailed Placement Algorithm

```
1. Perform single-segment clustering
2. Repeat
3.     Perform global swap
4.     Perform vertical swap
5.     Perform local reordering
6. Until no significant improvement in wirelength
7. Repeat
8.     Perform single-segment clustering
9. Until no significant improvement in wirelength
```

Global swap is the technique that gives the most wirelength reduction. It examines all cells one by one. For each cell i, the goal is to move it to its **optimal region**. For a given placement, the optimal region of cell i is defined as the region such that if cell i is placed in it, the wirelength will be optimal. It can be determined on the basis of the median idea of [Goto 1981]. Let E_i be the set of nets connected to cell i. For each net $e \in E_i$, the bounding box excluding cell i

is computed. Let x_L^e and x_U^e be the x-coordinates of left and right boundaries, and y_L^e and y_U^e be the y-coordinates of lower and upper boundaries, respectively. Then the optimal x-coordinate for cell i is given by the median of the set of boundary coordinates $\{x_L^e : e \in E_i\} \cup \{x_U^e : e \in E_i\}$. In general, the optimal coordinate for cell i is a region rather than a single point as the number of elements in the set $\{x_L^e : e \in E_i\} \cup \{x_U^e : e \in E_i\}$ is even. Similarly, the optimal y-coordinate for cell i is given by the median of the set of boundary coordinates $\{y_L^e : e \in E_i\} \cup \{y_U^e : e \in E_i\}$. For example, the optimal region of a cell i connected to three nets $A = \{i, 1, 2, 3\}$, $B = \{i, 4, 5\}$, and $C = \{i, 6, 7\}$ is given in Figure 11.22.

Although it is desirable in terms of wirelength to move cell i to its optimal region, the optimal region may not have enough space to accommodate the cell. So in global swap, cell i attempts to swap with another cell or a space in the optimal region. A benefit function is computed for each cell and each space in the optimal region. If there exists a cell or a space with positive benefit, the one with the highest benefit will be swapped with cell i. The benefit function consists of two components. The first component is the improvement in total wirelength if the swap is performed. The second component is a penalty. Swapping cells of different sizes or swapping a big cell with a small space may create overlap. The overlap is resolved by shifting nearby cells away. The penalty is a function of the least amount of shifting required to resolve the overlap.

Vertical swap is similar to global swap. The difference is that a cell attempts to swap with a few nearby cells one row above/below its current position. Sometimes a cell fails to be moved by global swap, because there is no cell or space in the optimal region with a positive benefit. Vertical swap allows the cell to move toward its optimal region to reduce the vertical wirelength. In addition,

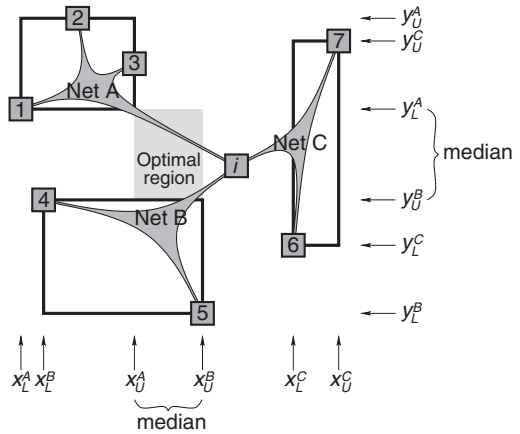

FIGURE 11.22

The optimal region of cell *i*.

vertical swap is much faster than global swap, because the number of candidates to be considered for swapping is much less.

Local reordering considers each possible group of n consecutive cells in a row. For each group, all possible left-right orderings of the cells are tried, and the one with the best wirelength is selected. Local reordering is a very inexpensive technique to locally minimize horizontal wirelength. In practice, n is set to 3. It is not necessary to use a larger n, because it is more efficient to fix nonlocal errors by global swap.

Single-segment clustering is a technique to minimize the horizontal wirelength by shifting the cells in a segment without changing the cell order. In FastDP, a segment is a maximal unbroken section of a standard cell row. Single-segment clustering examines each segment one by one. When a segment is considered, the locations of all cells in other segments are fixed. A very efficient algorithm based on a clustering idea is presented in [Pan 2005] to find the optimal nonoverlapping placement of the cells in a segment. Interested readers may refer to [Pan 2005] for the details.

11.8 CONCLUDING REMARKS

Placement is such a fundamental problem in VLSI design that there are so many different formulations and algorithms in the literature. This chapter is by no means a comprehensive survey on placement. In this chapter, total wirelength minimization for standard-cell design is focused. Algorithms for other design styles and other objectives are more or less extensions of the algorithms for this basic formulation.

Three popular approaches for global placement are presented. Other approaches that are not considered in this chapter are based on the eigenvalue method [Hall 1970], resistive network optimization [Cheng 1984], genetic algorithm [Cohoon 1987; Shahookar 1990], and artificial neural network [Yu 1989]. Those approaches are not successful in practice, but their insights may be intellectually interesting to readers.

A lot of useful and up-to-date information is collected in the book "Modern Circuit Placement" [Nam 2007]. That book contains descriptions of the underlying algorithms, implementation details, and latest experimental results for nine state-of-the-art academic placers. Another useful resource is a tutorial on large-scale circuit placement that summarizes results from recent optimality and scalability studies of placement tools and highlights recent techniques for optimization of wirelength, routability, and performance [Cong 2005]. A comprehensive survey of older placement techniques can be found in [Shahookar 1991].

Several placement benchmark suites are available in the public domain. The ISPD 2005 benchmarks [Nam 2005, 2007] and the ISPD 2006 benchmarks [Nam 2006, 2007] are derived from modern ASIC designs in IBM. They are mixed-size designs with both fixed and movable modules. The number of movable modules in the largest circuit is 2.2 million in the ISPD 2005 suite and 2.5 million in the ISPD

2006 suite. A major feature of the ISPD 2006 suite is that placement density targets are specified to address the routability concern; this feature is absent from the ISPD 2005 suite. The IBM-PLACE 2.0 [Yang 2002] and IBM-MSwPins [Adya 2004] benchmark suites are derived from smaller circuits released by IBM during ISPD 1998 [Alpert 1998]. The IBM-PLACE 2.0 benchmarks have standard cells only and have no connection to I/O pads, because all macros and the associated nets are removed from the netlists. They have exact pin locations. The IBM-MSwPins benchmarks contain large movable macros and many fixed pads distributed through the periphery. The Faraday Mixed-size benchmarks [Adya 2004] have sufficient routing information to run an industrial router on them after placement. Also, there are several variations of the PEKO benchmark suites [Chang 2003; Nam 2007], which are synthetic placement benchmarks with known optimal wirelength. They can be used to evaluate the optimality of placement algorithms.

These and many other benchmarks, as well as source codes/executables of many academic placers, can be downloaded from the "Wirelength-driven Standard-Cell Placement" slot of GSRC Bookshelf [Caldwell 2002].

11.9 EXERCISES

11.1. (Introduction) Consider a special placement problem that determines whether a standard-cell circuit without any net can be placed inside a given placement region. Prove that this problem is NP-complete.

11.2. (Problem Formulations) Prove that HPWL is a lower bound of RSMT wirelength. Prove that HPWL is the same as RSMT wirelength for 2- or 3-pin nets.

11.3. (Problem Formulations) Prove that RMST wirelength is an upper bound of RSMT wirelength. Give an example such that RMST wirelength overestimates RSMT wirelength by 50%.

11.4. (Global Placement: Partitioning-Based Approach) For the hypergraph in Figure 11.2b, assume all hyperedges have a weight of 1. What is the optimal bipartitioning if one partition should contain two vertices and the other should contain three?

11.5. (Global Placement: Partitioning-Based Approach) Draw the gain bucket data structures for the bipartition in Figure 11.3a.

11.6. (Global Placement: Partitioning-Based Approach) In the hMetis algorithm, a random bipartitioning is done in the initial partitioning phase. Would it be much better to replace the random bipartitioning by the FM algorithm?

11.7. (Global Placement: Partitioning-Based Approach) In Section 11.3, a placement approach based on bipartitioning is presented. In this question, a bipartitioning approach based on wirelength-minimized placement is considered. For a circuit with n modules, each module is first placed in one of the n integer coordinates $1, 2, \ldots, n$

on the x-axis. Then the modules in coordinates $1, \ldots, k$ form one subcircuit and those in coordinates $k+1, \ldots, n$ form another subcircuit, where k is chosen according to the size constraints. Show by constructing an example that even if optimal placement can be found, the cut cost of the bipartitioning solution by this approach can be $O(n^2)$ times that of the optimal solution.

11.8. (Global Placement: Simulated Annealing Approach) Prove that the penalty function C_3 of TimberWolf in Section 11.4.1 is equivalent to the following function:

$$C_3' = 2\theta \times \sum_r [l(r) - d(r)]^+$$

$$\text{where } [z]^+ = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

11.9. (Global Placement: Analytical Approach) Prove that the matrix $\boldsymbol{Q}$ defined in Section 11.5.2 is positive definite if all movable modules are connected to fixed modules either directly or indirectly.

11.10. (Global Placement: Analytical Approach) Consider the example in Figure 11.11. Assume all nets have a weight of 1. Assume module 4 is at (1, 0), module 5 is at (0, 3), and module 6 is at (4, 2).

1. Determine the locations of the movable modules such that the total weighted quadratic wirelength is minimized.
2. If module 1 is at (2, 2), module 2 is at (1, 3), and module 3 is at (3, 2), find the force vectors exerting on the movable modules.

11.11. (Global Placement: Analytical Approach) For each of the following net models, write the number of 2-pin nets and the number of extra variables introduced for a k-pin net as functions of k.

1. Clique
2. Star
3. Hybrid
4. BoundingBox

Plot the number of 2-pin nets for all 4 models in a graph for the range $2 \leq k \leq 15$.

11.12. (Global Placement: Analytical Approach) Prove that the function $\tilde{\Theta}_x(b, i)$ in Equation (11.3) is continuous when $d_x = w_b + w_i/2$. Is it differential when $d_x = w_b + w_i/2$?

11.13. (Global Placement: Analytical Approach) This question analyzes the error of the log-sum-exponential function defined in Section 11.5.3 as an approximation of the maximum function. The error function is defined as:

$$\mathrm{err}_\alpha(z_1, \ldots, z_n) = \mathrm{LSE}_\alpha(z_1, \ldots, z_n) - \max(z_1, \ldots, z_n)$$

Derive an upper bound and a lower bound of $\mathrm{err}_\alpha(z_1, \ldots, z_n)$ over all possible values of $z_1, \ldots, z_n$ as functions of n and α.

11.14. (Detailed Placement) Consider the FastDP algorithm in Section 11.7. Prove that the optimal region is given by the medians of the sets of boundary coordinates.

ACKNOWLEDGMENTS

I thank Natarajan Viswanathan of Iowa State University, Professor Cheng-Kok Koh of Purdue University, Dr. Laung-Terng Wang of SynTest Technologies, Inc., and Professor Yao-Wen Chang of National Taiwan University for carefully reviewing the chapter.

REFERENCES

R11.0 Books

[Arrow 1958] K. Arrow, L. Huriwicz, and H. Uzawa, *Studies in Nonlinear Programming,* Stanford University Press, Stanford, CA, 1958.

[Evans 2002] L. C. Evans, *Partial Differential Equations,* American Mathematical Society, Providence, RI, 2002.

[Garey 1979] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, New York, 1979.

[Luenberger 1984] D. G. Luenberger, *Linear and Nonlinear Programming,* second edition, Addison Wesley, Reading, MA, 1984.

[Nam 2007] G.-J. Nam and J. Cong, editors, *Modern Circuit Placement—Best Practices and Results,* Springer, Boston, 2007.

R11.1 Introduction

[Garey 1974] M. R. Garey, D. S. Johnson, and L. Stockmeyer, Some simplified NP-complete problems, in *Proc. ACM Symp. on Theory of Computing*, pp. 47–63, April–May 1974.

R11.2 Problem Formulations

[Chu 2008] C. Chu and Y.-C. Wong, FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design, *IEEE Trans. on Computer-Aided Design*, 27(1), pp. 70–83, January 2008.

[Guibas 1983] L. J. Guibas and J. Stolfi, On computing all northeast nearest neighbors in the L1 metric, in *Information Processing Letters*, 17(4), pp. 219–223, April 1983.

[Hwang 1976] F. K. Hwang, On Steiner minimal trees with rectilinear distance, *SIAM J. of Applied Mathematics*, 30(1), pp. 104–114, January 1976.

R11.3 Global Placement: Partitioning-Based Approach

[Agnihotri 2003] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden, Fractional cut: Improved recursive bisection placement, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 307–310, November 2003.

[Alpert 1997] C. J. Alpert, J.-H. Huang, and A. B. Kahng, Multilevel circuit partitioning, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 530–533, June 1997.

[Breuer 1977a] M. A. Breuer, A class of min-cut placement algorithms, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 284–290, June 1977.

[Breuer 1977b] M. A. Breuer, Min-cut placement, *J. of Design Automation and Fault Tolerant Computing*, 1(4), pp. 343–382, October 1977.

[Caldwell 2000] A. E. Caldwell, A. B. Kahng, and I. L. Markov, Can recursive bisection produce routable placements, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 477–482, June 2000.

[Dunlop 1985] A. E. Dunlop and B. W. Kernighan, A procedure for placement of standard-cell VLSI circuits, *IEEE Trans. on Computer-Aided Design*, 4(1), pp. 92–98, January 1985.

[Fiduccia 1982] C. M. Fiduccia and R. M. Mattheyses, A linear-time heuristic for improving network partitions, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 175–181, June 1982.

[Karypis 1997] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, Multilevel hypergraph partitioning: Application in VLSI domain, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 526–529, June 1997.

[Yildiz 2001a] M. C. Yildiz and P. H. Madden, Global objectives for standard cell placement, in *Proc. 11th ACM Great Lakes Symp. on VLSI*, pp. 68–72, March 2001.

[Yildiz 2001b] M. C. Yildiz and P. H. Madden, Improved cut sequences for partitioning based placement, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 776–779, June 2001.

R11.4 Global Placement: Simulated Annealing Approach

[Sechen 1986] C. Sechen and A. L. Sangiovanni-Vincentelli, TimberWolf 3.2: A new standard cell placement and global routing package, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 432–439, June 1986.

[Sun 1995] W.-J. Sun and C. Sechen, Efficient and effective placement for very large circuits, *IEEE Trans. on Computer-Aided Design*, 14(3), pp. 349–359, March 1995.

[Wang 2000] M. Wang, X. Yang, and M. Sarrafzadeh, Dragon2000: Standard-cell placement tool for large industry circuits, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 260–263, November 2000.

R11.5 Global Placement: Analytical Approach

[Alpert 2005] C. Alpert, A. Kahng, G.-J. Nam, S. Reda, and P. Villarrubia, A semi-persistent clustering technique for VLSI circuit placement, in *Proc. ACM Int. Symp. on Physical Design*, pp. 200–207, April 2005.

[Brenner 2005] U. Brenner and M. Struzyna, Faster and better global placement by a new transportation algorithm, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 591–596, June 2005.

[Chan 2005] T. Chan, J. Cong, and K. Sze, Multilevel generalized force-directed method for circuit placement, in *Proc. ACM Int. Symp. on Physical Design*, pp. 185–192, April 2005.

[Chen 2006] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, A high quality analytical placer considering preplaced blocks and density constraint, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 187–192, November 2006.

[Eisenmann 1998] H. Eisenmann and F. Johannes, Generic global placement and floorplanning, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 269–274, June 1998.

[Hu 2002] B. Hu and M. Marek-Sadowska, FAR: Fixed-points addition and relaxation based placement, in *Proc. ACM Int. Symp. on Physical Design*, pp. 161–166, April 2002.

[Kahng 2004] A. B. Kahng and Q. Wang, Implementation and extensibility of an analytical placer, in *Proc. ACM Int. Symp. on Physical Design*, pp. 18–25, April 2004.

[Kahng 2005] A. B. Kahng and Q. Wang, Implementation and extensibility of an analytic placer, *IEEE Trans. on Computer-Aided Design*, 24(5), pp. 734–747, May 2005.

[Karypis 1997] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, Multilevel hypergraph partitioning: Application in VLSI domain, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 526–529, June 1997.

[Kleinhans 1991] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, GORDIAN: VLSI placement by quadratic programming and slicing optimization, *IEEE Trans. on Computer-Aided Design*, 10(3), pp. 356–365, March 1991.

[Kowarschik 2001] M. Kowarschik and C. Weiβ, DiMEPACK—a cache-optimized multigrid library, in *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, pp. 425–430, June 2001.

[Mo 2000] F. Mo, A. Tabbara, and R. Brayton, A force-directed macro-cell placer, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 177–180, November 2000.

[Nam 2006] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng, A fast hierarchical quadratic placement algorithm, *IEEE Trans. on Computer-Aided Design*, 25(4), pp. 678–691, April 2006.

[Naylor 2001] W. Naylor, Non-linear Optimization System and Method for Wire Length and Delay Optimization for an Automatic Electric Circuit Placer, U.S. Patent No. 6,301,693. Oct. 9, 2001.

[Quinn 1975] N. R. Quinn, The placement problem as viewed from the physics of classical mechanics, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 173–178, June 1975.

[Sigl 1991] G. Sigl, K. Doll, and F. M. Johannes, Analytical placement: A linear or a quadratic objective function, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 427–431, June 1991.

[Spindler 2006] P. Spindler and F. Johannes, Fast and robust quadratic placement combined with an exact linear net model, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 179–186, November 2006.

[Viswanathan 2004] N. Viswanathan and C. Chu, FastPlace: Efficient analytical placement by use of cell shifting, iterative local refinement and a hybrid net model, in *Proc. ACM Int. Symp. on Physical Design*, pp. 26–33, April 2004.

[Viswanathan 2007a] N. Viswanathan, M. Pan, and C. Chu, FastPlace: An efficient multilevel force-directed placement algorithm, in Gi-Joon Nam and Jason Cong, editors, *Modern Circuit Placement—Best Practices and Results*, pp. 193–228, Springer, Boston, 2007.

[Viswanathan 2007b] N. Viswanathan, G.-J. Nam, C. Alpert, P. Villarrubia, H. Ren, and C. Chu, RQL: Global placement via relaxed quadratic spreading and linearization, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 453–458, June 2007.

[Vygen 1997] J. Vygen, Algorithms for large-scale flat placement, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 746–751, June 1997.

[Xiu 2004] Z. Xiu, J. D. Ma, S. M. Fowler, and R. A. Rutenbar, Large-scale placement by grid-warping, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 351–356, June 2004.

R11.6 Legalization

[Hill 2002] D. Hill, Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design, U.S. Patent No. 6,370,673. April 9, 2002.

[Khatkhate 2004] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C. K. Koh, and P. H. Madden, Recursive bisection based mixed block placement, in *Proc. ACM Int. Symp. on Physical Design*, pp. 84–89, April 2004.

[Ren 2005] H. Ren, D. Z. Pan, C. Alpert, and P. Villarrubia, Diffusion-based placement migration, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 515–520, June 2005.

R11.7 Detailed Placement

[Agnihotri 2003] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden, Fractional cut: Improved recursive bisection placement, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 307–310, November 2003.

[Caldwell 2000] A. E. Caldwell, A. B. Kahng, and I. L. Markov, Optimal partitioners and end-case placers for standard-cell layout, *IEEE Trans. on Computer-Aided Design*, 19(11), pp. 1304–1314, November 2000.

[Doll 1994] K. Doll, F. M. Johannes, and K. J. Antreich, Iterative placement improvement by network flow methods, *IEEE Trans. on Computer-Aided Design*, 13(10), pp. 1189–1200, October 1994.

[Goto 1981] S. Goto, An efficient algorithm for the two-dimensional placement problem in electrical circuit layout, *IEEE Trans. on Circuits and Systems*, 28(1), pp. 12–18, January 1981.

[Pan 2005] M. Pan, N. Viswanathan, and C. Chu, An efficient and effective detailed placement algorithm, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 48–55, November 2005.

R11.8 Concluding Remarks

[Adya 2004] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov, Unification of partitioning, floorplanning and placement, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 550–557, November 2004.

[Alpert 1998] C. J. Alpert, The ISPD98 Circuit Benchmark Suite, in *Proc. ACM Int. Symp. on Physical Design*, pp. 80–85, April 1998. http://vlsicad.ucsd.edu/UCLAWeb/cheese/ispd98.html.

[Caldwell 2002] A. E. Caldwell, A. B. Kahng, and I. L. Markov, Toward CAD-IP reuse: The MARCO GSRC bookshelf of fundamental CAD algorithms, in *IEEE Design and Test*, pp. 72–81, 2002. http://www.gigascale.org/bookshelf/.

[Chang 2003] C.-C. Chang, J. Cong, and M. Xie, Optimality and scalability study of existing placement algorithms, in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, pp. 621–627, January 2003.

[Cheng 1984] C. Cheng and E. Kuh, Module placement based on resistive network optimization, *IEEE Trans. on Computer-Aided Design*, 3(3), pp. 218–225, July 1984.

[Cohoon 1987] J. P. Cohoon and W. D. Paris, Genetic placement, *IEEE Trans. on Computer-Aided Design*, 6(6), pp. 956–964, November 1987.

[Cong 2005] J. Cong, J. R. Shinnerl, M. Xie, T. Kong, and X. Yuan, Large-scale circuit placement, in *ACM Trans. on Design Automation of Electronics Systems*, 10(2), pp. 389–430, April 2005.

[Hall 1970] K. M. Hall, An r-dimensional quadratic placement algorithm, *Management Science*, 17 (3), pp. 219–229, November 1970.

[Nam 2005] G.-J. Nam, C. J. Alpert, P. Villarubbia, B. Winter, and M. Yildiz, The ISPD2005 placement contest and benchmark suite, in *Proc. ACM Int. Symp. on Physical Design*, pp. 216–220, April 2005. http://www.sigda.org/ispd2005/contest.htm.

[Nam 2006] G.-J. Nam, The ISPD 2006 placement contest: Benchmark suite and results, in *Proc. ACM Int. Symp. on Physical Design*, p. 167, April 2006. http://www.sigda.org/ispd2006/contest.html.

[Shahookar 1990] K. Shahookar and P. Mazumder, A genetic approach to standard cell placement using metagenetic parameter optimization, *IEEE Trans. on Computer-Aided Design*, 9(5), pp. 500–511, May 1990.

[Shahookar 1991] K. Shahookar and P. Mazumder, VLSI cell placement techniques, *ACM Computing Surveys*, 23(2), pp. 143–220, June 1991.

[Yang 2002] X. Yang, B.-K. Choi, and M. Sarrafzadeh, Routability-driven white space allocation for fixed-die standard-cell placement, in *Proc. ACM Int. Symp. on Physical Design*, pp. 42–49, April 2002.

[Yu 1989] M. L. Yu, A study of the applicability of Hopfield decision neural nets to VLSI CAD, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 412–417, June 1989.