Chapter 4

MULTILEVEL CIRCUIT PLACEMENT

Tony F. Chan
Mathematics Department, UCLA
Los Angeles, CA 90095-1555
chan@math.ucla.edu

Jason Cong
Computer Science Department, UCLA
Los Angeles, CA 90095-1596
cong@cs.ucla.edu

Tim (Tianming) Kong
Aplus Design Technologies, Inc.
Los Angeles, CA 90024
kongtm@aplus-dt.com

Joseph R. Shinnerl
Computer Science Department, UCLA
Los Angeles, CA 90095-1596
shinnerl@cs.ucla.edu

Keywords: VLSICAD, multiscale methods, large-scale optimization, placement

Introduction

Following logic synthesis and circuit design, an integrated circuit is represented as a collection of interconnected rectangular modules of fixed dimensions. Timing constraints on signal propagation paths along sequences of connections are also specified. The task of *circuit placement*

is to arrange the modules inside a prescribed rectangular region such that no two modules overlap, timing constraints are satisfied, and the estimated total wirelength needed to implement the connections is minimized. Thus, an algorithm for placement derives a suitable spatial characterization of a given circuit from a logical-temporal one.

Assume given a simplified *netlist* description of an integrated circuit:

- A fixed rectangular region $\mathcal{R}$ in the Manhattan plane;
- a set $V_c = \{v_1, \ldots, v_N\}$ of N rectangular *modules* called *cells*;
- a set $V_p = \{v_{N+1}, \ldots, v_{\bar{N}}\}$ of fixed connection points along the boundary of $\mathcal{R}$ called *pads*;
- a set $\mathcal{E} = \{e_1, \ldots, e_M\}$ of M subsets of $\mathcal{V} \equiv V_c \cup V_p$ called *nets*.

The nets $e_i \in \mathcal{E}$ specify the connectivity of the circuit; each cell typically belongs to several nets and has a distinct connection point or *pin* on its boundary for each one. The pads are used for input/output (I/O) to and from the entire circuit. The terminology is illustrated in Figure 4.1.

For simplicity, we omit here the notation needed to support any timing constraints on the signal-propagation paths. In practice, this information is quite important, especially in performance optimization. The timing-independent problem remains quite challenging, however, as for large circuits, the $N(N-1)/2$ pairwise nonoverlap constraints are typically the greatest source of difficulty.

When the cells are positioned in $\mathcal{R}$, each net $e \in \mathcal{E}$ is assigned a "bounding-box" wirelength estimate $\ell(e)$ equal to the half-perimeter of the smallest rectangle circumscribing its cells and pads:

$$\ell(e) = (x_{\max}(e) - x_{\min}(e)) + (y_{\max}(e) - y_{\min}(e)), \tag{4.1}$$

where $x_{\max}(e)$ denotes the maximum x-coordinate of any side of any cell or pad in net e, etc.. The corresponding wirelength estimate ℓ for the entire circuit is obtained by direct summation over all nets:

$$\ell = \sum_{\text{nets } e \in \mathcal{E}} \ell(e). \tag{4.2}$$

With this terminology and these assumptions, the *wirelength-driven* placement problem can be restated as follows.

> Given a netlist description $(\mathcal{V}, \mathcal{E})$ of an integrated circuit in a placement region, $\mathcal{R}$, position the cells $v \in \mathcal{V}$ in $\mathcal{R}$ to minimize the circuit's total bounding-box wirelength ℓ in such a way that no two modules overlap.

For simplicity, we refer to wirelength-driven placement simply as placement throughout the chapter. Alternative formulations may emphasize more specific objectives, such as signal delay, area, power, or routability.

The problem and its significance are illustrated by the tiny example in Figures 4.1 and 4.2. Nets are shown as color-coded connections. Figure 4.2 displays a poor placement. Excess total wirelength degrades circuit performance and makes routing difficult (Chapter 5). Indeed, for large circuits, placements as bad as the one in Figure 4.2 are often *unroutable*: the nets cannot physically be implemented due to the high wiring density required. Figure 4.1 shows a good placement of the same circuit.

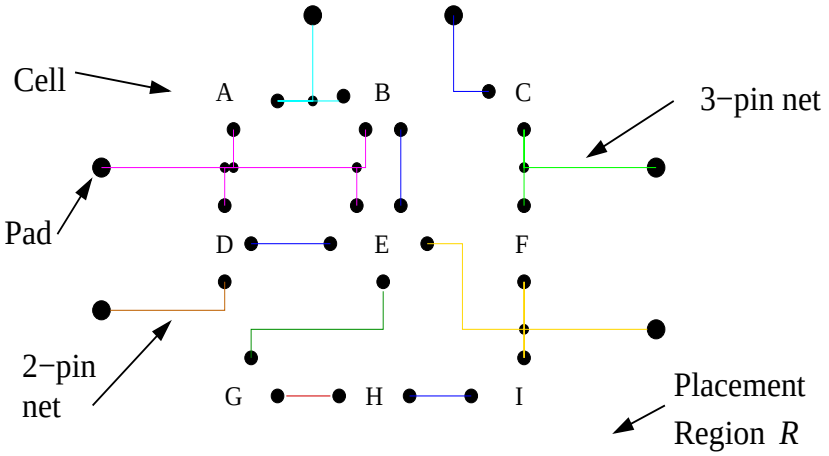

Figure 4.1. A Good Placement

Circuit placement is NP-hard. Current circuit designs require placement solutions for N and M of order 10^6, and these sizes continue to grow with Moore's Law. In this chapter, we give an overview of a few of today's leading methods for large-scale placement followed by a detailed description of our own multilevel placement method, mPL.

1. Problem Formulation and Approximations

Circuit placement can be formulated precisely as a nonlinear integer-programming problem as follows.

$$\begin{array}{lll} \min\limits_{x,y\in\mathbf{R}^n} & f(x,y) & \\ \text{subject to} & c_i(x,y) \geq 0. \quad i = 1,\ldots,m & \text{(NIP)} \\ & (x,y) \in \mathcal{D} \subset \mathbf{R}^{2n}, & \end{array} \qquad (4.3)$$

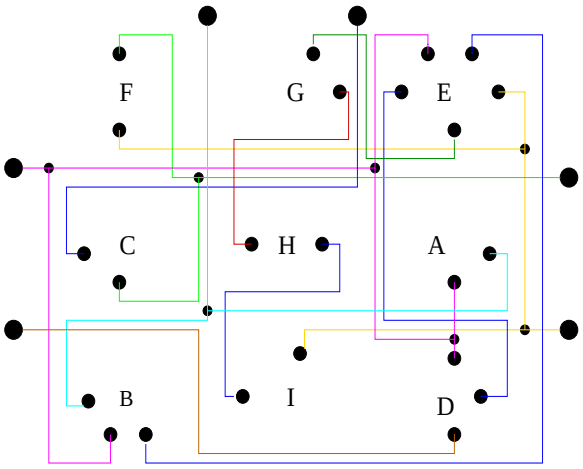

Figure 4.2. A Bad Placement of the Same Circuit in Figure 4.1

where $(x, y) \in \mathcal{D} \subset \mathbf{R}^{2n}$ holds the x and y coordinates of the cells' centers, and the objective f and the constraints c_i are scalar-valued functions defined on $\mathcal{D}$. Set $\mathcal{D}$ is disconnected because cells must ultimately be aligned in rows; in some cases, the column positions may also be restricted. The *feasible region* for NIP is denoted by $\mathbf{F}$ and is the intersection of $\mathcal{D}$ with $\{(x, y) \,|\, c_i(x, y) \geq 0 \text{ for all } i\}$; *feasible configurations* of the cells satisfy all the constraints, including pairwise nonoverlap. Circuit placement is naturally represented by discrete and nonsmooth formulations for f and c_i. For large designs, however, smooth approximations can be effective. Specific expressions for f and c_i are discussed in Section 1.1 below.

We assume cells must be placed in fixed orientation in alignment with the boundary of the placement region — no rotation or reflection (flipping) is allowed. When changes in the orientations and shapes of the placement region and the cells are allowed, the problem is usually called *floorplanning*.

1.1 Objective and Constraint Representations

In practice it is not uncommon for one algorithm to employ more than one form of objective function or overlap-constraint functions. The most popular formulations and variations are stated here.

1.1.1 Wirelength Estimation. A precise estimate of wirelength can only be made after routing (Chapter 5). For a net of degree d, the accuracy of its bounding-box wirelength estimate (4.1) can be enhanced by scaling by $\sqrt{d}$. During placement, however, the form of estimate used may vary, even across different components of the same program. In this subsection, we consider smooth estimates of wirelength.

A quadratic $q(x, y)$ is often used for the wirelength objective $f(x, y)$ in (4.3) as follows:

$$q(x,y) = \tfrac{1}{2} \sum_{\text{cells } i,j} \gamma_{ij} \left((x_i - x_j)^2 + (y_i - y_j)^2 \right). \tag{4.4}$$

Each constant weight γ_{ij} can be selected to reflect (i) the number and vertex degrees of the different nets which contain both cell i and cell j as well as (ii) the relative importance of those nets with regard to timing. Let (x, y) be denoted simply as z, ignore for the moment all fixed pads, and consider (4.4) expressed in the form $q(z) = \frac{1}{2} z^T Q z + b^T z$. The matrix Q is the well-known *graph Laplacian* of the circuit:

$$q_{ij} = -\gamma_{ij} \text{ for } i \neq j, \quad \text{and} \quad q_{ii} = \sum_{j \neq i} \gamma_{ij}.$$

Matrix Q is weakly diagonally dominant and, hence, symmetric positive-semi-definite. When the pads are included, Q is symmetric positive definite, and the unique solution of the linear system $Qz = -b$ is an unconstrained optimal-wirelength placement.

Quadratic wirelength is considerably different from bounding-box wirelength, (4.2). One way to make the approximation closer is scale the terms in the sum, as illustrated in Section 2.2.1 below. Another is the star-wirelength formulation, in which the wirelength of each net e is taken to be the sum of the distances of its cells' positions to their common average position, (x_e, y_e). That is, with $|e|$ the number of cells in net e, $(x(v), y(v))$ the center of cell v, and

$$(x_e, y_e) = \left(\frac{1}{|e|} \sum_{\text{cells } v \in e} x(v), \quad \frac{1}{|e|} \sum_{\text{cells } v \in e} y(v) \right),$$

we calculate Euclidean star wirelength as

$$\ell_2^* = \sum_{\text{nets } e \in \mathcal{E}} \sum_{\text{cells } v \in e} \sqrt{(x(v) - x_e)^2 + (y(v) - y_e)^2 + \epsilon},$$

and Manhattan star wirelength as

$$\ell_1^* = \sum_{\text{nets } e \in \mathcal{E}} \sum_{\text{cells } v \in e} \sqrt{(x(v) - x_e)^2 + \epsilon} + \sqrt{(y(v) - y_e)^2 + \epsilon}\ ,$$

where ϵ is used to guarantee smoothness in the Euclidean case when all cells are concentric and in the Manhattan case when all cells are colinear. If smoothness is not required, ϵ can be set to zero.

1.1.2 Nonoverlap Constraints. For rectangular cell v_i, denote its center by (x_i, y_i), its width by w_i, its height by h_i, its left and right sides by $x_{\min}(v_i) = x_i - w_i/2$ and $x_{\max}(v_i) = x_i + w_i/2$, and its top and bottom sides by $y_{\min}(v_i) = y_i - h_i/2$ and $y_{\max}(v_i) = y_i + h_i/2$, respectively. Then for cells v_i and v_j, the overlap area between them is

$$a(v_i, v_j) = a_x(v_i, v_j) \cdot a_y(v_i, v_j), \tag{4.5}$$

where $a_x(v_i, v_j) =$

$$\max\left\{0,\ \left(\min\{x_{\max}(v_i), x_{\max}(v_j)\} - \max\{x_{\min}(v_i), x_{\min}(v_j)\}\right)\right\},$$

and $a_y(v_i, v_j)$ is defined similarly.

In this exact formulation, overlap is modeled by $N(N-1)/2$ nonsmooth equality constraints of the form $c_{ij}(x_i, x_j, y_i, y_j) = 0$. Traditionally, use of the exact formulation is limited to detailed placement (Section 1.3, below), when cell moves can be restricted enough that overlap need be considered simultaneously only for small neighborhoods of cells. An efficient $\mathcal{O}(N)$ approximation to this pairwise formulation is also possible for global placement, however, and is discussed in Section 3.2.

Instead of directly targeting overlap, most algorithms focus on gradually evening out the distribution of cell area in the placement region. A rectangular grid is defined over the region by uniformly spaced horizontal and vertical gridlines. Each rectangle in this grid is referred to as a *bin*. The number of bins used may vary, but, during global placement, it is typically less than the number of cells. In the most direct formulation, bin b_i is assigned an inequality constraint of the form

$$c_i(x, y) \equiv \text{area}(b_i) - \text{cell-area}(b_i) \geq 0,$$

with

$$\text{cell-area}(b_i) = \sum_{\{\text{cells } v_j \in \mathcal{V} \mid a(b_i, v_j) > 0\}} a(b_i, v_j),$$

where $a(b_i, v_j)$ is calculated from (4.5). Provided simple data structures are maintained for keeping track of which bins and cells overlap, evaluation of *all* such c_i over the entire grid is $\mathcal{O}(N)$. Many variations on this theme have been applied; cf. Sections 2.2.2, 2.4.2, and 2.4.1. Although such an approach is usually not a practical means of removing *all* overlap, it is quite effective in preparing an initial configuration for use by legalization algorithms relying on localized cell moves.

1.2 Hypergraph and Graph Models

Ignoring all spatial information, including cell dimensions and pin, pad, and cell positions, we may refer to cells and pads as *vertices* and to nets as *hyperedges*. This abstraction of the circuit's connectivity is the *hypergraph* model introduced in Chapters 1 and 3: $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. Vertices are weighted by their cells' areas; hyperedges can be weighted to reflect their cardinalities or their signal timing. Most nets in a circuit consist of just two or three cells, but a significant fraction are considerably larger, and some contain over 100 cells. Although the total number of hyperedges possible in an arbitrary hypergraph is 2^N, for large, practical circuits, the number of nets M is typically quite comparable to N and usually not more than about $3N$.

The fact that a net may connect more than two cells implies that a graph model of the circuit is inaccurate. However, graph approximations such as the *clique model* we describe here are often used effectively for many purposes. Vertices of the hypergraph are used as vertices of the graph. Every hyperedge of cardinality $k < k_{\max}$ is replaced by a clique; hyperedges larger than $k_{\max}$ are replaced by chains. Each edge in the same clique or chain is weighted the same amount, typically $1/(k-1)$ or $2/(k(k-1))$. When two or more distinct hyperedges share two distinct hypergraph vertices, this process introduces more than one edge between the two corresponding vertices in the graph. To maintain the graph structure, these "parallel edges" are replaced by a single edge whose weight is the sum of the parallel edges' weights. Vertices in the graph are weighted by the areas of their corresponding cells.

The question of how well any graph can approximate a given hypergraph is subject to debate. However, the leading methods for hypergraph partitioning (Chapter 3) and circuit placement all make use of graph approximation in various ways.

1.3 Global Placement and Detailed Placement

As practiced today, circuit placement typically consists of two distinct phases: *global* placement and *detailed* placement. The purpose of global placement is to rapidly identify a promising region of the solution space small enough to be explored thoroughly during detailed placement. The purpose of detailed placement is to rapidly transform the result of global placement into a feasible placement while increasing the wirelength as little as possible or, if possible, continuing to decrease it.

A detailed placement algorithm by itself is essentially useless without a good, nearly feasible initial point generated by global placement. Hence, although strict feasibility is not expected from the global placement, its

iterates must approach feasibility sufficiently fast to support an efficient transition to the detailed method. In practice, global placement proceeds until (i) the rate of wirelength reduction has become too slow, and (ii) constraint violations are sufficiently small in some norm. For instance, suppose overlap is initially removed by the enforcement of bin-density constraints (Section 1.1.2). The sizes of the bins are selected according to the problem domain (the type of circuit); each bin may hold anywhere from 5 to 50 cells or more. Global placement terminates once each bin is between, say, 50 and 100% full; *and* its last few iterations have produced little or no wirelength gain.

Global placement is usually considered more difficult and more important than detailed placement. In principle, a detailed-placement algorithm can be transformed into a complete global-and-detailed placement method simply by making it the relaxation component of a multilevel algorithm. All that is needed are coarsening and interpolation strategies. In practice, such an approach may not be competitive, depending on the power of the detailed-placement algorithm. Successful methods tend to employ complementary techniques at different stages and scales. Simple detailed-placement strategies tightly constrained by discrete feasibility constraints may lack the power to explore the solution space at coarser levels sufficiently thoroughly.

2. Contemporary Methodology

A variety of algorithms have been successfully applied to placement, each customized to accomodate the scale and complexity of the problem. The focus in this section is primarily on global placement.

2.1 Annealing Methods

Simulated annealing (SA, [Kirkpatrick et al., 1983]) has been applied to circuit placement very successfully [Sechen, 1988]. SA-based placement algorithms are readily adapted to handle any known form of constraint. In addition, they possess excellent "hill-climbing" properties (they can readily escape local attraction basins), and their run-time properties are readily tuned to meet users' demands. A leading commercial package, `iTools` [Swartz, 2002] is based on simulated annealing. SA-based methods share the following features.

1 A weighted penalty function objective, e.g., [Sun and Sechen, 1995] $f = W + \mu P_0 + \lambda P_1$, where W represents wirelength, P_0 represents excessive cell overlap, and P_1 represents path-based timing violations.

2 A neighborhood structure, i.e., for each given configuration, a set of "neighboring" configurations obtained by simple "state changes" effected by simple cell moves. These moves are generated in a constrained but random fashion. Moves leading to improved or "downhill" configurations, as determined by the change in the weighted penalty objective value, are accepted with very high probability, often 1.0. Moves leading to worse or "uphill" configurations are accepted with probability according to the current value of the "temperature" parameter.

3 An annealing schedule, i.e., a set of rules by which a temperature is steadily reduced. Initially, the temperature is relatively high, and large-distance uphill moves are more likely to be accepted. Once the temperature is sufficiently close to zero, uphill moves are never accepted, and the system settles into a final local minimum.

The primary challenge to SA-based methods is scalability. In practice the temperature must typically be decreased far too rapidly [Stander and Silverman, 1994] to support any analytical guarantee of any particular level of solution quality [Geman and Geman, 1984]. Three-level clustering has been used to accelerate SA-based placement [Sun and Sechen, 1995].

Recently, two multiscale placement algorithms using SA relaxation have been reported [Sankar and Rose, 1999; Chang et al., 2002]; we describe them in Section 2.5.

2.2 Analytical Methods

Methods closely connected with continuous approximation and mathematical programming are traditionally referred to as "analytical." Most of these require simplifying assumptions on the constraints, e.g., linearity or "smoothness" (continuous differentiability to first or perhaps second order).

2.2.1 GORDIAN and GORDIAN-L. The GORDIAN and GORDIAN-L algorithms provide elegant examples of top-down, hierarchical algorithms that do *not* rely on any bottom-up clustering to form their hierarchies. The principal difference between them is that GORDIAN [Kleinhans et al., 1991] uses the quadratic wirelength objective (4.4), whereas GORDIAN-L [Sigl et al., 1991] uses an iteratively refined piecewise "linear" approximation to the wirelength, obtained by termwise

scaling as follows:

$$q_{k+1}(x,\, y) = \tfrac{1}{2} \sum_{\text{cells } i,j} \gamma_{ij} \left(\frac{(x_i - x_j)^2}{|x_i^k - x_j^k|} + \frac{(y_i - y_j)^2}{|y_i^k - y_j^k|} \right), \qquad (4.6)$$

where $(x_k,\, y_k)$ denotes the constant previous iterate. Compared to (4.2), optimization of the "linear" wirelength objective (4.6) used by GORDIAN-L is generally observed to lead to higher-quality placements than optimization of the quadratic wirelength. However, the quadratic objective's aggressive penalization of long wires is considered advantageous in some contexts, e.g., in the presence of tight timing constraints.

The definiteness properties of the wirelength matrix Q (Section 1.1.1) are unchanged by the termwise scaling employed in (4.6). Hence, the same fast numerical techniques can be used to solve the sequences of linear systems of equations arising in both formulations. These are based on preconditioned conjugate gradients [Golub and Van Loan, 1996].

Aside from the wirelength model, the main steps of the algorithms are the same. Initially, quadratic wirelength is minimized subject to the single linear constraint that the center-of-mass (assuming a uniform mass/area density) of all cells lies at the center of the placement region. The placement region is then quadrisected into four equal-area rectangular subregions, and partitioning is performed in order to assign every cell to exactly one of the four subregions. At the next stage, the objective (quadratic or "linear") is again minimized over the entire placement region; however, this time four center-of-mass constraints are enforced, one for each subregion. Recursively quadrisecting subregions and adding corresponding center-of-mass constraints gradually evens out the cell distribution. Cells are not restricted to remain within their assigned subregions during each global minimization; hence, later stages can "correct" possibly poor assignments made by earlier partitionings. Eventually the recursive divisions produce a sufficiently evenly distributed placement for the effective use of localized, network-flow techniques to obtain completely legal placements (DOMINO, [Doll et al., 1994]).

Minimizing a convex quadratic function subject to linear equality constraints can be done optimally by solving just one linear system of equations. Careful decomposition of these equations leads to significant acceleration, but not directly, however, to scalability. Significant improvements to the GORDIAN framework (quadratic minimization combined with recursive quadrisection and partitioning) have been made since its publication, leading to the efficient solution of problems with over one million movable cells [Vygen, 1997; Brenner and Rohe, 2002]. A multilevel implementation of GORDIAN or GORDIAN-L has yet to be reported.

2.2.2 KRAFTWERK. The KRAFTWERK algorithm [Eisenmann and Johannes, 1998] introduces forces between cells in order to remove overlap during global placement. Given a positive-definite quadratic-wirelength objective $f(z) = \frac{1}{2}z^T Qz + b^T z$ defined over placements $z \equiv (x, y) \in \mathbf{R}^{2N}$, *every* possible placement z is, in principle, obtainable simply by appropriate definition of the right-hand side vector $-b$ in the optimality condition $Qz = -b$. Initially, cells are arranged without regard for overlap by minimization of the quadratic subject to a single center-of-mass constraint. For each given placement z_k, a distribution of forces b_k based on local cell densities is calculated and added to the right-hand side. Solving the updated equations gives the next iterate in the sequence.

Timing constraints can be handled by adjusting the coefficients in Q to weight some nets more than others. In this case, Q changes from one iteration to the next.

The method for calculating forces proposed in [Eisenmann and Johannes, 1998] is elegant and highly efficient. A uniform bin grid (Section 1.1.2) is defined over the placement region. The density of a bin is defined to be proportional to the number of cell centers lying within its boundaries. Interpreting the bin densities as evaluation points of a field potential, forces are defined by the density potential's first derivatives. A Poisson equation modeling the density can be solved explicitly, leading to convolution integrals for the forces. For these, specialized, fast $\mathcal{O}(N \log N)$ evaluation algorithms are available.

Compared to GORDIAN-L-DOMINO, the wirelengths obtained by KRAFTWERK-DOMINO are improved by about 5 to 15%.

2.3 Partitioning-Based Methods

Suitably abstracted forms of circuit placement are equivalent to hypergraph partitioning (Chapter 3), in the following sense. For convenience, assume the cells and blocks are uniform and square. Placement is simply a k-way partitioning problem with k sufficiently large, roughly the same as the number of cells. For correspondence in the net-cut and wirelength objectives, take the unit of distance to coincide with the width and height of each block of the partition, and take the cutsize associated with each net to be the sum of the maximum number of blocks spanned in the vertical and horizontal directions. Two-way partitioning, on the other hand, can be viewed simply as placement in one dimension with the pairwise nonoverlap constraints relaxed to a single area-balancing constraint. The discrete net-cut objective can be expressed as continu-

ous wirelength, if the two regions are separated from each other by one unit of distance.

The traditional min-cut objective of partitioning and half-perimeter objective of placement are related by *Rent's Rule*, as follows. Given an arbitrary rectangle inscribed within the boundaries of an actual physical integrated circuit, the number of wires N_p crossing the rectangle's perimeter follows the the number of cells N_c within that rectangle, on average over all such rectangles, according to the formula

$$N_p = A_{pc} N_c^r$$

for constant A_{pc} and some fixed value of r near 0.5.

These observations have not been lost on practioners. The close connection between high-quality placements and high-quality partitions was recognized early [Breuer, 1977; Cheng and Kuh, 1984; Dunlop and Kernighan, 1985]; partition-driven placement remains a dominating paradigm. Partitioning has traditionally been viewed as being simpler than placement — a means of reducing the placement problem to smaller subproblems. In principle, any effective method for partitioning can be directly transformed to a method for placement by simple top-down recursive application. That is, apply the partitioning method to the initial circuit (level H), then to each of its output subregions (level $H-1$), and so on to lower and lower levels until these are sufficiently small that detailed methods can be efficiently used. Connections between cells in a given subregion being partitioned to cells in other regions can be considered by introducing "dummy terminals" along the subregion boundary [Dunlop and Kernighan, 1985].

In 1997, [Alpert et al., 1997] and [Karypis et al., 1997] independently reported successful implementations of multilevel algorithms for hypergraph partitioning. The dramatic improvement of these techniques (Chapters 2 and 3) over preceding methods has outpaced progress in placement algorithms, making partitioning-based placement methods especially attractive. Two recent independent packages for placement based on multilevel circuit partitioning have achieved excellent results: CAPO [Caldwell et al., 2000] and DRAGON [Wang et al., 2000].

2.3.1 CAPO. Because of its simplicity, flexibility, and power, recursive bipartitioning has long been a prominent model for placement algorithms. The idea behind CAPO [Caldwell et al., 2000] is to explore the limits of this framework using a powerful multilevel method of partitioning with enhancements to the underlying FM-based relaxation (the Fiduccia-Mattheyses (FM) method for hypergraph bipartitioning is introduced in Chapters 2 and 3). Complementary analytical placement

techniques, e.g., quadratic programming, are deliberately avoided. Primary goals of CAPO are simplicity in formulation, speed in execution, and routability of the final result. The most significant enhancements to FM include block-splitting heuristics and hierarchical tolerance computation. That is, cutlines across blocks are carefully chosen, and FM is repeated several times on each block, first with a loose balancing tolerance and subsequently with progressively tighter tolerances. Once blocks are reduced to fewer than 35 movable cells each, time-limited branch-and-bound techniques are used to obtain optimal or near-optimal partitions. Rather than enforce nonoverlapping directly, sufficient *relative whitespace*, $1 - (\text{total cell area})/(\text{block area})$, is maintained in every block to ensure a high likelihood of routability in the final placement result.

2.4 Hybrid Methods

In practice, it is rare that just one placement algorithm or tool is used in the design of a real circuit. Hybrid methods combine different and complementary algorithms to efficiently explore a larger fraction of the solution space than might otherwise be possible. Multiple search heuristics can accelerate the exploration of complex solution spaces in a fashion not unlike that of multiscale algorithms.

2.4.1 MONGREL. In MONGREL [Hur and Lillis, 2000], the goal of global placement is to correctly assign each of the N cells to exactly one of approximately $N/10$ uniform, rectangular bins covering the placement area. The principal optimizations used during global placement are

1 linear-programming-based (LP) subset relaxation,

2 feasible max-gain-path movement of cells one by one from their starting locations to their optimal locations as determined in step 1, and

3 further improvement by FM partitioning across adjacent bins.

The first two of these steps are explained in more detail below. Detailed placement is based on a novel dynamic-programming approach to reordering the cells within a given row. The row reordering uses one level of clustering, and the LP relaxations are performed on subsets of widely varying sizes. Thus, the techniques seem well suited to a globally hierarchical formulation. But aside from these limited connections, there is no explicit attempt at any recursive aggregation or multilevel optimization. The authors describe their approach as "middle-down."

The LP relaxations and overlap removal are natural candidates for relaxation in a multilevel approach. The first step is the selection of a set of movable cells M and the identification of the fixed cells F with which they share hyperedges. (Without the fixed cells, the optimal relaxation is less useful: all cells are placed at the exact same position.) In MONGREL, M is selected as follows. Initially, M is identified with a randomly selected net. Cells in nets intersecting with M are then added to M until a prespecified size limit is reached. Let E_M denote the set of nets that intersect with the final M: $E_M \equiv \{e \in \mathcal{E} \mid e \cap M \neq \emptyset\}$. The relaxation problem can then be decoupled into two one-dimensional problems, one in the x-direction and one in y. In the x-direction, the problem can be stated as follows in terms of dummy variables l_e and r_e associated with each net $e \in E_M$:

$$
\begin{array}{ll}
\min_{l_e, r_e, x_v} \sum_{e \in E_M} r_e - l_e & \\
\text{subject to} & \\
l_e \leq x_v \leq r_e & \text{for all } e \in E_M \text{ and all } v \in e, \\
x_v = X_v & \text{for all } v \in F,
\end{array}
$$

where X_v denotes the given locations of the fixed cells, and r_e and l_e denote the respective right-hand and left-hand boundaries of net e. In [Hur and Lillis, 1999], it is shown that this particular LP has an efficient network-flow-based solution. Once these optimal relaxed locations of the cells have been determined, cells are moved one at a time in some unspecified order to their relaxed locations. After each move, feasibility, as defined by bin congestion limits, is immediately restored, if necessary, by a cell-by-cell migration process. In this scheme, directed bin paths of maximum gain are found from congested bins to uncongested bins. The weight associated with directed edge (a, b) in each such path is the greatest possible reduction in wirelength obtained by moving some cell not in M from bin a to neighboring bin b. During wirelength calculation, cells in M are assumed to occupy their relaxed locations, even if they have not yet actually been moved. To limit calculation costs, only monotone paths are considered: for any edge (a, b) in a path leading from bin b_{over} to bin b_{under}, the Manhattan distance to b_{under} must be reduced: $\|b - b_{\text{under}}\|_1 < \|a - b_{\text{under}}\|_1$. Because cells are moved one at a time and feasibility is immediately restored after each move, a sequence of feasible configurations is examined; the best of these is used as the final result of the relaxation. As there is no guarantee that each configuration improves the wirelength (the maximum gain along a path may be negative), the best configuration obtained may not be the last one in the sequence.

2.4.2 DRAGON. Dragon [Wang et al., 2000; Yang, 2002] is based on top-down recursive quadrisection (4-way partitioning) with wirelength improvement at each level by low-temperature simulated annealing. After each quadrisection, adjacent blocks are swapped when such swapping reduces wirelength. During quadrisection, connections between cells in different blocks are ignored. During block swapping, blocks are not required to remain within their (larger) ancestor blocks from preceding (coarser) levels. This last feature gives operations at later, finer levels some ability to correct wrong or premature localizations performed at the earlier, coarser levels. Experiments reported by the authors suggest that net-cut is a more useful objective than wirelength at the earlier, coarser levels. Global placement ends once the blocks are small enough to contain about 7 or fewer cells. At the last step of global placement, individual cells within blocks are exchanged with cells in neighboring blocks. In detailed placement, the cells in each bin are spread out in a random order to remove overlap. Localized permutations of small cell subsets are enumerated, either vertically or horizontally, to further reduce wirelength, but these local searches are performed in a fast, greedy fashion, without further annealing.

Dragon might be viewed as a multilevel algorithm, except that its hierarchy is ostensibly constructed from the top down rather than from the bottom up. In the reported experiments for Dragon, hMETIS (Chapter 3) is the actual partitioner used. As hMETIS is a multilevel partitioner, it employs recursive aggregation (First-Choice clustering) in its hierarchy construction. Thus, Dragon inherits to some extent this bottom-up hierarchy from hMETIS but does not explicitly use it during its top-down refinement phase.

2.5 Multilevel Simulated-Annealing-Based Methods

The flexibility and relative ease of implementation of SA make an attractive method of relaxation in a multilevel algorithm. SA-based multilevel circuit placement has been studied by [Sankar and Rose, 1999] as a means of generating useful FPGA placements extremely fast and by [Chang et al., 2002] as an efficient and effective means of incorporating routing congestion control into placement.

2.5.1 Ultra-Fast Multilevel Placement: SR99. Sankar and Rose have combined fast recursive clustering with simulated annealing to produce good placements extremely rapidly [Sankar and Rose, 1999]. For brevity, we refer to their algorithm as SR99. The emphasis

in this work is on speed and how to trade speed for quality. On one circuit with over 100,000 logic gates, SR99 obtains in just ten seconds a wirelength within 33% of that obtained by a competitive tool, VPR [Betz and Rose, 1997] in 524 seconds. When allowed to run the same amount of time as VPR, SR99 obtains comparable wirelength quality.

Clustering in SR99 proceeds as follows. Each cluster c begins as a single randomly selected "seed" cell. Each cell b connected to the still unfinished cluster c is then weighted as follows:

$$w_b = A_{bc} + \sum_{\{\text{nets } e \mid b \in e \text{ and } c \in e\}} \frac{1}{|e| - 1}.$$

This weight consists of two terms: (i) the number of nets A_{bc} that can be *absorbed* by clustering b with c (because b is the only cell these nets contain that is not already in c). (ii) the strength of its connection to the cluster, measured by summation over shared nets, with small nets favored over large ones. The cell with the largest such weight is then added to the cluster, and the weights of neighboring cells are updated. The clustering process continues until a preset threshold count is reached. Hence, all clusters except the last one have the same number of cells; this number is restricted to be a perfect square (4, 9, 16, 25, ...) in order to facilitate grid resizing.

In the FPGA context for which SR99 is intended, the problem formulation is slightly different. First, all cells have the same dimensions; hence, all clusters but one will have the same area as well. Second, pads are allowed to move.

An initial constructive placement is performed at the coarsest level as follows. First, pads are placed randomly along the boundary of the placement region. Second, cells are partitioned into three sets: (i) those connected to output pads, (ii) those connected to input pads, (iii) those connected only to other cells and not to any pads. Finally, the cells in these sets are placed, in order: cells in set (i) are placed first, then cells in set (ii), then cells in set (iii). Each time a cell is placed, all connections to it are ignored except those to pads or to other cells that have already been placed. Each cell is placed as close as possible to the mean position of the pads and already-placed cells to which it is connected.

At subsequent finer levels, initial constructive placements are formed in a similar fashion to the coarse-level algorithm, except that no connectivity need be ignored. Mean position calculations use parent-cluster centers for cells that have not yet been placed. Pads are also placed in this way.

Fast simulated annealing as in VPR [Betz and Rose, 1997] is the method of relaxation applied to the initial placements at each level. The

inherited clustering from the adjacent coarser level is not necessarily preserved: the relaxations do more than simply position the components of the parent clusters. A few different temperature-reduction methods are compared for their time-quality trade-off properities. Overall, temperature reduction in SR99 is more aggressive than in VPR. The simplest means of trading run-time for placement quality in this framework is to increase the number of moves per temperature.

To our knowledge, SR99 is the first implementation of multilevel circuit placement. In their experiments, the authors observe that their method is initially more efficient than the analogous nonhierarchical algorithm VPR in the following sense. Initially, the multilevel method reduces wirelength much faster than the flat method, but both methods eventually achieve the same final wirelength quality in the same total amount of time. Thus, Sankar and Rose present multilevel placement mainly as a means of accelerating the early stages of existing placement algorithms. They also propose its use in high-quality wirelength and routability estimation.

2.5.2 mPG: Physical Hierarchy Generation with Routing Congestion Control. In a routed circuit (Chapter 5), relatively long wires are referred to collectively as the *global interconnect*. As indicated in the preface to this book, the global interconnect has come to dominate device delay as the single most important factor in the determination of an integrated circuit's performance. This shift has put pressure on designers to model the interconnect earlier and more accurately in the design cycle [Cong, 2001].

The aim of mPG [Chang et al., 2002] is to efficiently incorporate interconnect planning and routability estimation into a fast, competitive method of placement. In mPG a multilevel placement algorithm is augmented by layer assignment and fast incremental global routing. First-choice coarsening (Chapter 3) is employed to generate the cluster hierarchy. A fixed, hierarchical set of bin-density constraints controls overlap; this same bin structure is used at every level as a means of estimating the distribution of "long" wires, i.e., those that connect cells in different bins. Relaxation is SA-based. Adapting fast algorithms for computing rectilinear Steiner trees [Cong et al., 1999], the routing gives estimates of wiring congestion along bin boundaries. The mPG objective can be tuned to vary the emphasis placed on wirelength and congestion.[1] Emphasizing wirelength leads to slightly improved wirelength with significant speed-up (3–6×) as compared to GORDIAN-L-DOMINO. Empha-

[1] The authors refer to the congestion-driven version of their method as mPG-cg.

sizing congestion increases run-time significantly but yields placements with significantly improved routability: 53–79% less overflow and 3–6% less routed wirelength.

The following general conclusions can be drawn from the work on mPG. First, a careful implementation of multilevel placement can produce an order-of-magnitude speed up without any degradation in solution quality. Second, a multilevel formulation readily supports the incorporation of additional, tunable, and complex objectives and constraints. In particular, it can be used to efficiently support more accurate interconnect planning and estimation earlier in the physical-design cycle.

3. mPL: A Multilevel Placement Algorithm

Our work has been focused primarily on multiscale global placement. For large classes of straightforward continuous formulations of placement, each of the $\mathcal{O}(N!)$ possible configurations of cells will correspond to a constrained local minimizer of a penalized wirelength objective (see below). It seems quite unlikely that this "complexity barrier" can be overcome by any acceleration technique that does not use a hierarchical formulation. Without coarse-scale approximation, how can we efficiently identify the most profitable large-scale moves? All the approaches described in the preceding sections have been used to develop highly successful methods in the 100K-cell range; some have also been extended to handle million-cell circuits. However, their scalability to larger sizes is not clear.

mPL evolved from an effort to apply nonlinear programming to the placement problem. Early emphasis in the work was placed on *global* iterations, i.e., steps in which every movable object is moved simultaneously, without unnecessary localization restrictions or simplifying linearizations. Early experiments suggested that, while such an approach might be effective for circuits with up to a few thousand cells, the intricacy of the feasible region due to the pairwise nonoverlap constraints precludes any scalability to large circuits. Even on problems with just a few hundred cells, local pointwise approximations are meaningful only in very small neighborhoods, necessitating very small step sizes and, hence, slow convergence.

Our first attempt to employ a multiscale approach was rather crude and met with only partial success. The speed and scalability of mPL 1.0 were superior, beating other methods by factors of 10 or more on the largest problems tested. However, the total wirelengths obtained by mPL 1.0, while roughly comparable to those of GORDIAN-L-DOMINO, were some 15–20% greater than those reported by other researchers the

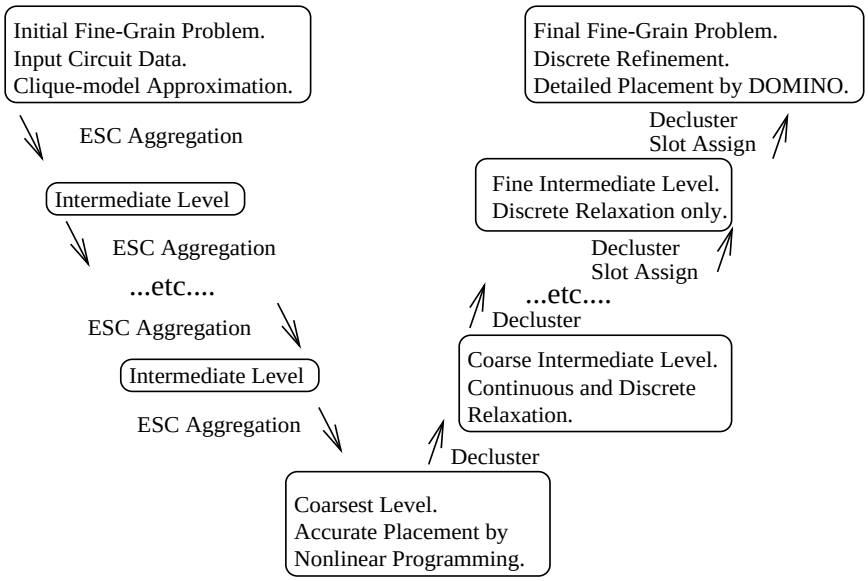

Figure 4.3. The V-Cycle Flow of mPL

same year. An explanation for this gap and a description of our recent efforts to fill it without any loss of speed or scalability are given in the conclusion to the chapter.

The mPL multilevel placement algorithm (Version 1.0) [Chan et al., 2000] has the following flow.

1 Recursive edge-separability-based clustering [Cong and Lim, 2000] (Section 3.1).

2 Nonlinear-programming-based relaxation at coarser levels.

3 Interpolation to finer levels by simple declustering; legalization by linear assignment.

4 Local-exchange-based discrete relaxation (Section 3.3).

This flow is illustrated in Figure 4.3. A hierarchy of placement problems is constructed by recursively clustering cells according solely to their connectivity. At the coarsest level (approximately 500 cells), a high-quality placement of clusters is performed by a customized interior-point solution of a nonlinear-programming model (Section 3.2 below). To transfer a coarse-level solution to its adjacent finer level, each cluster is replaced by its component or "child" clusters positioned concentrically at the parent cluster's center. A uniform bin grid is laid down, and the declustered solution is then legalized by a straightforward linear assignment of

child clusters to nearby bins: the cost of assigning a cluster to a bin is simply the Manhattan distance of the corresponding displacement. The complexity of this step is reduced at larger levels as necessary by a few iterations of hypergraph partitioning: hMETIS is used for this. Following the legalization, several sweeps of small cell-subset permutations are done to improve the wirelength.

Many contemporary placement algorithms perform optimizations at every level of a hierarchy. To our knowledge, mPL, mPG, and SR99 are the only successful packages to build their hierarchies from the bottom up and to use *placements* at coarser levels, rather than just partitions, to build placements at finer levels.

3.1 Bottom-Up Hierarchy Construction

Edge-separability clustering (ESC) [Cong and Lim, 2000] is applied recursively to build the mPL graph-model hierarchy. Advantages of ESC are its speed and the degree to which relatively large-scale information is used. ESC is based on CAPFOREST [Nagamochi and Ibaraki, 1992], an $\mathcal{O}(n \log n)$ method for approximating the xy-min-cut for every edge $e = \{x, y\}$ in an undirected, edge-weighted graph, as described below.

The vertex weights are used in ESC to maintain cluster-area balance.

3.1.1 Global *xy*-Min-Cut Estimation by CAPFOREST.

Suppose then that the hypergraph representation of our circuit has been transformed to an undirected, vertex-weighted and edge-weighted graph $G = (V, s, E, w)$, where $s : V \to [0, \infty)$, and $w : E \to [0, \infty)$. For any vertex $x \in V$, the *weighted degree* $\delta(x)$ of x is the sum of $w(e)$ over all e incident on x. Given edge $e = \{x, y\}$, the *xy-min-cut* of e, $\lambda(e)$, is the least total edge weight of any set of edges $F \in E$ that, if removed from G, will separate x and y into distinct connected components of $G \setminus F \equiv (V, s, E \setminus F, w|_{E \setminus F})$, where $w|_{E \setminus F}$ denotes the restriction of the weight function w to the undeleted edges. Since $\delta(x)$ is just the cost of removing x from G, $\lambda(e)$ satisfies the following bounds for edge $e = \{x, y\}$:

$$w(e) \leq \lambda(e) \leq \min\{\delta(x), \delta(y)\}. \tag{4.7}$$

With $n = |V|$ and $m = |E|$, precise determination of the xy-min-cut for just one edge requires $\mathcal{O}(mn \log(n^2/m))$ operations [Goldberg and Tarjan, 1988]. However, the CAPFOREST method obtains useful approximations $q(e)$ to the xy-min-cuts for *all* edges $e \in E$ in $\mathcal{O}(m + n \log n)$ steps. For most practical large-scale circuits, $m = \mathcal{O}(n)$, so the bound simplifies to $\mathcal{O}(n \log n)$. The CAPFOREST method is summarized in Figure 4.4.

Vertex labels r used in CAPFOREST reflect connectivity only and are independent of any vertex s already in G. Initially, all vertex labels r are set to zero. Vertex labels r and edge labels q are fixed only once their corresponding vertices or edges are marked. Most r-labels are increased several times before being fixed. Edge labels $q(e)$, however, are set only once.

A starting vertex v_0 is picked at random and marked. Its neighbors are assigned initial r-labels equal to the weights of the edges connecting them to v_0, but they are not yet marked. The edges e incident on v_0 are assigned xy-min-cut estimates $q(e) = w(e)$ and are marked. Subsequently, at each iteration, an unmarked vertex x of maximum label $r(x)$ adjacent to the current set of marked vertices is selected and marked. Every unmarked vertex y adjacent to x along an edge $e = \{x, y\}$ (e is unmarked, necessarily) then has its label $r(y)$ increased by addition of $w(e)$; each such edge e is then assigned xy-min-cut estimate $q(e) = r(y)$ and marked. The iterations continue until all vertices are marked. The order in which vertices are visited is known as the *maximum adjacency* (MA) ordering, because at each step the unmarked vertex selected has the highest total edge weight connecting it to the set of marked vertices at that step. The result of the process is the new edge-labeling, q; each $q(e)$ is an estimate of the xy-min-cut for that edge. Nagamochi

```
given an undirected, edge-weighted graph G = (V, E, w)
Label all vertices and edges unmarked.
Initialize r(v) = 0 for each v ∈ V.
while there exist unmarked vertices
    Select unmarked vertex x of maximum label r(x)
    for each unmarked vertex y adjacent to x along (unmarked) edge e
        r(y) = r(y) + w(e)
        q(e) = r(y)
        Mark e
    end for
    Mark x
end while
return q
```

Figure 4.4. The CAPFOREST algorithm for all-edges xy-mincut estimation

and Ibaraki [Nagamochi and Ibaraki, 1992] show that the xy-min-cut estimates q do not exceed the true xy-min-cuts they are meant to approximate:

$$w(e) \leq q(e) \leq \lambda(e). \tag{4.8}$$

Statistics from experiments [Lim, 2000] suggest that $q(e)$ is typically about 90% of $\lambda(e)$.

3.1.2 Edge-Separability Clustering (ESC). Clustering is done by merging vertices joined by high-rank edges, as defined below, subject to upper-bound constraints on cluster area. These bounds must be specified in such a way that it is possible to cluster at least some specified fraction of the vertices. This minimum coarsening rate guarantees scalability of the overall algorithm, as long as the relaxations used are scalable.

Edge rank $\rho(e)$ is computed as the quotient of CAPFOREST xy-min-cut estimate $q(e)$ and minimum vertex weight $s(v)$ as follows:

$$\rho(e) = \frac{q(e)}{\min\{s(x), s(y)\}}. \tag{4.9}$$

Ranking the edges in this way encourages the merging of tightly connected vertex pairs as well as vertex pairs with one or both vertices small in area. The denominator helps maintain area balance among clusters.

The multilevel ESC clustering scheme used in MPL is summarized in Figure 4.5. At each level, CAPFOREST is used to build a max-heap of "contractible" edges according to (4.9). An edge is considered contractible if its xy-min-cut estimate is larger than the minimum weighted degree of all vertices in G and the sum of its vertices' areas does not exceed the prespecified area bound. Edges in the heap are removed in sequence from the top (highest-rank edge first) and examined for feasibility of contraction according to the cluster-area bound. If an edge removed from the heap can be feasibly contracted, it is; otherwise, it is simply discarded. To contract an edge $e = \{x, y\}$, we remove y, add $s(y)$ to $s(x)$, and replace each edge $\{y, z\}$, $z \neq x$, by a corresponding edge $\{x, z\}$ with the same weight, if $\{x, z\}$ does not already exist. If edge $\{x, z\}$ already exists, then its weight is simply increased by adding to it the weight of $\{y, z\}$. In this case, the min-cut estimate $q(\{x, z\})$ is replaced by the maximum of $q(\{x, z\})$ and $q(\{y, z\})$ in order to keep the estimate as large as possible without exceeding the true xz-min-cut. Once all edges in the heap have been considered, the current clustering phase ends with a coarsening of the original graph. Repeating the clustering in a recursive fashion produces a telescoping sequence of coarsened graphs. The recursion terminates when a prespecified target size for the coarsest level is attained.

3.1.3 Variations and Comparisons. Since our application of ESC does not require spatial information, no optimization is done during

given an undirected, vertex-weighted and edge-weighted graph $G = (V, s, E, w)$
Fix $\epsilon \in (0, 1)$. Let $l = 0$.
while the coarse-level limit has not been reached
  Set the cluster-area limit to ensure contractibility of at least $\epsilon|E|$ edges.
  Run CAPFOREST to estimate all edges' xy-min-cuts.
  Heapify the edges according to (4.9).
  while the edge heap is nonempty
    if the top edge can be contracted without exceeding the area limit
      Contract the edge
    else
      Discard the edge
    Pop the heap
  end while
  Store the resulting graph for level l. Let $l = l + 1$.
end while

Figure 4.5. The multilevel ESC algorithm for recursive graph coarsening

the coarsening phase. Optimization (relaxation) begins at the coarsest level once the entire hierarchy is constructed and continues at every level of the interpolation phase. In most PDE-based multilevel algorithms, relaxation is considered essential during both the coarsening and interpolation phases at each level of the hierarchy. Subsequent V-cycles after the first are also generally found to be effective. In our implementation, however, neither relaxation during coarsening nor multiple V-cycles provided any significant gain.

We attempted to use spatial information during the ESC coarsening as follows. First, at the finest level we calculated initial positions for the cells by minimizing unconstrained quadratic wirelength. Edge-rank calculation in ESC (4.9) was modified as follows:

$$\rho_g(e) = \frac{q(e)}{(1 + d(x, y)^\alpha) \min\{s(x), s(y)\}}, \tag{4.10}$$

where $d(x, y)$ denotes the distance between x and y, and α is fixed; we used $\alpha = 0.3$. The modified edge rank also encourages clustering of cells that are in close proximity. We refer to this modification of ESC as ESCg.

The results of our clustering experiments are shown in Table 4.1. The "RandomMatch" procedure serves as a basis for comparison; rankings are assigned to the edges at random. CPU times are quite comparable for the different methods and are therefore omitted. For a specification of the circuit sizes, see Table 3.4 in Section 3.4 below.

Table 4.1. Comparison of Different Clustering Algorithms

circuit	MESC	MESCg	RandomMatch
struct	7.68e+05	7.64e+05	8.24e+05
biomed	3.86e+06	3.90e+06	4.09e+06
avqsmall	1.25e+07	1.29e+07	1.32e+07
avqlarge	1.37e+07	1.39e+07	1.44e+07

The results are surprising in a few respects. First, the result obtained with ESCg was typically worse than that obtained with plain ESC. This result suggests that the information obtained by unconstrained quadratic minimization reflects mainly connectivity and is not sufficiently different from the information used by ESC. Second, the result obtained with ESC was only about 6% better than that obtained by random clustering. This result may suggest that the multilevel method is fairly insensitive to the choice of clustering. However, it may also indicate simply that ESC, despite its clear advantages, is relatively ineffective for placement.

3.2 Nonlinear Programming by a Penalized Interior-Point Method

A nonlinear-programming formulation supports the accurate representation of nonconvex constraints and the simultaneous movement of all movable cells. These *global iterations* ostensibly hold some advantage over other approaches: configuration changes based on information from *all* current cell positions might be expected to improve beyond what can be done by moving only a small fraction of cells based on information from only part of the circuit. However, due to the very high complexity of the feasible region, the usual pointwise information from function values and derivatives is useful only in a very small neighborhood of the point of evaluation. Aside from the relatively large number of operations per iteration required, the number of iterations increases dramatically with problem size due to the small step sizes required for consistent decrease in the objective. Hence, our use of nonlinear programming in MPL has thus far been limited to the coarsest levels in the multilevel hierarchy, in which the total number of movable cells is no more than one or two thousand.

We refer to the smooth reformulation of NIP as NP, for which $\mathcal{D} = \mathbf{R}^{2n}$, the chip-boundary constraints being included in the inequalities

$c_i(x,y) \geq 0$. The *strict interior* of $\mathbf{F}$ is then simply $\mathcal{S} = \{(x,y) \mid c_i(x,y) > 0 \text{ for all } i\} \subset \mathbf{F}$.

To avoid nonsmooth nonoverlap constraints, we approximate the rectangular cells by circular disks. All cells' pin locations are fixed at their respective cells' centers. Different disks may have different radii, but each disk's radius is fixed. This approximation leads to $N(N-1)/2$ pairwise nonoverlap constraints of the form

$$c_{ij}(x,y) = (x_i - x_j)^2 + (y_i - y_j)^2 - (\rho_i + \rho_j) \geq 0 \qquad (i > j), \quad (4.11)$$

where $\rho_i = \frac{1}{2}(w_i + h_i)$ is cell i's radius, and (x_i, y_i) is its center. Efficient $\mathcal{O}(N)$ handling of these constraints is discussed below.

Quadratic wirelength is used mainly for convenience in mPL. Smooth approximations to the bounding-box wirelength can also be used.

While it is mathematically possible to formulate alignment constraints using continuous functions [Cheng and Kuh, 1984], the computational effort required to approximately solve them seems likely to be excessive. Hence, for global placement, we ignore the row-alignment constraints altogether. We have not attempted to reformulate detailed placement for NP. For detailed placement, we use DOMINO [Doll et al., 1994] in an off-the-shelf fashion.

3.2.1 A Penalized Interior-Point Method for Placement.

General methods for the solution of NP based on *sequential unconstrained minimization techniques* (SUMT) were thoroughly analyzed by Fiacco and McCormick in the late 1960's [Fiacco and McCormick, 1968]. In the 1980's, it was discovered that these methods lead to polynomial-time formulations on convex problems [Karmarkar, 1984; Gill et al., 1986]. Since then, they have been successfully applied to many large-scale problems, both convex and nonconvex. Although, strictly speaking, these methods are guaranteed to converge only to some *local* solution to NP, in practice they are designed to avoid poor-quality solutions and seek out near-global ones. The basic idea is shown in Figure 4.6. Consider interior-point methods first. Suppose for the moment that the feasible region, $\mathbf{F}$, is connected. Although some solutions to NP may exist strictly interior to $\mathbf{F}$, most solutions of interest can typically be expected to lie on the boundary of $\mathbf{F}$, where $c_i(x,y) = 0$ for some i. Given a starting point (x_0, y_0) strictly interior to $\mathbf{F}$, we construct a sequence of strictly interior points $\{(x_k, y_k)\}$ converging to a local solution (x^*, y^*) of NP as follows. Iterate (x_k, y_k) is computed as an approximation to an unconstrained minimizer of the *logarithmic barrier function*

$$B_{\mu_k}(x,y) = f(x,y) - \mu_k \sum_i \ln c_i(x,y), \quad (4.12)$$

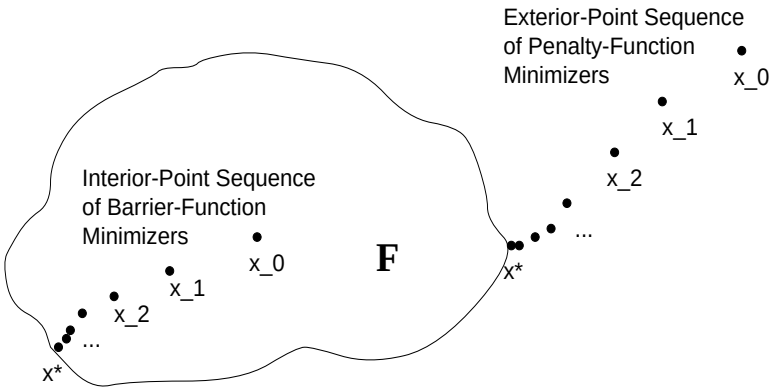

Figure 4.6. Sequential Unconstrained Minimization Techniques (SUMT)

where each positive value of the *barrier parameter* μ_k defines a distinct *barrier subproblem*: minimization of B_{μ_k} over all (x, y) interior to $\mathbf{F}$. The barrier function resembles f in most of $\mathcal{S}$ but diverges to $+\infty$ near its boundary. As long as f is bounded below and $\mathbf{F}$ is bounded (in extent), $B_\mu(x, y)$ must have an unconstrained minimizer $(x(\mu_k), y(\mu_k))$ in $\mathcal{S}$. As $B_\mu(x, y)$ may have several unconstrained minimizers, the particular one that $(x(\mu_k), y(\mu_k))$ represents is typically determined by the starting point $(x(\mu_{k-1}), y(\mu_{k-1}))$ and other details of the unconstrained minimization algorithm used to solve the subproblem. As μ_k decreases, the divergence in B_{μ_k} becomes ever more abrupt, allowing $(x(\mu_k), y(\mu_k))$ to approach the boundary of $\mathbf{F}$. In fact, under very mild conditions, the sequence $(x(\mu_k), y(\mu_k))$ converges to a local solution to NP as $\mu_k \to 0^+$ [Fiacco and McCormick, 1968].

In an analogous fashion, a sequence of *exterior-point* iterates can be constructed as unconstrained minimizers of a *quadratic penalty function*,

$$P_{\mu_k}(x, y) = f(x, y) + \frac{1}{2\mu_k} \sum_i \max\{-c_i(x, y), 0\}^2. \tag{4.13}$$

Robust convergence results exist in this framework as well, but for inequality constraints, the nonsmoothness in P_{μ_k} along the boundary of $\mathbf{F}$ cannot be removed, making its minimization somewhat more difficult than that of B_{μ_k}. When the constraints are *equalities*, $c_i(x, y) = 0$, the penalty $\max\{-c_i(x, y), 0\}$ can be replaced by $c_i(x, y)$, and the penalty is smooth. Hence, we restrict the use of the quadratic penalty to equality constraints, if any.

Conceptually, we may think of the barrier transformation as introducing a short-range repulsive force between cells; i.e., we may think of

the log-barrier method as an extension of previously attempted force-directed methods for circuit placement [Quinn and Breuer, 1979; Eisenmann and Johannes, 1998]. Properly placing the barrier subproblem within the larger nonlinear programming context, however, makes standard techniques more accessible and removes the need for *ad hoc* strategies for selecting parameter values. While recent research has largely been concerned with the favorable asymptotic aspects of SUMT, other additional advantages may be less familiar. In particular, effective implementations for complex problems with large numbers of local extrema can readily target high-quality regions of the solution space and exclude local solutions of poor quality.

There are two apparent obstacles to the application of an interior-point method to circuit placement. First, although finding a strictly interior point (x_0, y_0) for circuit placement is typically not difficult, proper initialization of the method requires that the point either be near a high-quality solution or be sufficiently far from the boundary that a trajectory toward high-quality solutions can be found. For the trajectory to be followed computationally, it must not approach tangency to the boundary. Second, moving continuously from one feasible configuration to another is practically impossible without temporarily violating several nonoverlap constraints along the way. That is, $\mathbf{F}$ for placement is highly disconnected, and some technique is needed to allow both feasible and infeasible iterates in the sequence. Fortunately, both these obstacles are readily overcome via the introduction of a single additional *artificial variable* ξ as follows:

$$B_{\mu_k}(x, y, \xi) = f(x, y) - \mu_k \sum_i \ln(c_i(x, y) + \xi) + \frac{1}{2\mu_k}\left(\xi + \xi^2\right). \quad (4.14)$$

This modification to B_{μ_k} corresponds to the following transformation to the original nonlinear programming problem:

$$\begin{array}{ll} \min\limits_{(x,y,\xi)\in\mathbf{R}^{2n+1}} & f(x, y) \\ \text{subject to} & c_i(x, y) + \xi \geq 0 \quad i = 1, \ldots, m \\ & \xi = 0. \end{array} \quad (4.15)$$

The initial value of ξ is chosen large enough that cells can overlap freely. To visualize the process, it is helpful to omit the equality constraint on ξ from consideration of the feasible region; hence, let the *inequality-based* feasible region $\mathbf{E}$ be defined as $\{(x, y, \xi) \,|\, c_i(x, y) + \xi \geq 0 \text{ for all } i\}$. Because in (x, y, ξ)-space $\mathbf{E}$ is connected, there is no difficulty in finding a good starting point strictly interior to $\mathbf{E}$ or in moving continuously between configurations, as long as ξ remains sufficiently large. As ξ is

reduced, the original constraints are gradually enforced, albeit indirectly. Thus we can apply an interior-point algorithm to (4.15). The final iterate $(\bar{x}, \bar{y})$ lies in a connected component of the true feasible region, $\mathbf{F}$. This component is attained once ξ is driven to zero and holds a low-cost placement. As ξ decreases, we can visualize the corresponding subproblems' feasible regions in (x, y, ξ)-space as gradually morphing from a simply connected region into a multiply disconnected one. The decrease in ξ, however, may not be completely monotonic. Iterate (x_k, y_k, ξ_k) may become infeasible when μ is reduced from μ_k to μ_{k+1}. Increasing ξ_k to compensate is an effective solution.

Compared to (4.13), a linear term in ξ has been added to the penalty to promote steady decrease in constraint violations even as ξ is driven toward zero. For general problems, an additional stage following elimination of ξ once $\xi = 0$ in which iterates remain strictly interior to $\mathbf{F}$ may further reduce the objective. On placement problems, however, so little continuous feasible movement of cells is possible that no processing is useful without $\xi > 0$.

A Newton-based linesearch method, as illustrated in Figure 4.7, is used for numerical solution of the penalized barrier subproblem (4.15).

given iterate $x \in \mathbf{R}^n$
while not converged
  Calculate *search direction* $p \in \mathbf{R}^n$.
  Calculate *steplength* $\alpha \in (0, \infty)$.
  Let $x = x + \alpha p$.
end while

Figure 4.7. Generic Linesearch Algorithm

Each iteration consists of two phases: calculation of the search direction, p, and calculation of the "steplength," α. The update to the current iterate is simply $x \leftarrow x + \alpha p$. The "steplength" α is computed using an algorithm specially developed to exploit the structure of the barrier function's singularity at the boundary of $\mathbf{F}$ [Murray and Wright, 1994].

3.2.2 Numerical Solution. The phrase "Newton-based" means that p eventually approximates a solution p_μ to the Newton equations for (4.15) at the current iterate,

$$H_\mu p_\mu = -g_\mu, \tag{4.16}$$

where $H_\mu \equiv H_\mu(x, y, \xi)$ denotes the Hessian of B_μ at (x, y, ξ), $g_\mu \equiv g_\mu(x, y, \xi)$ denotes the gradient of B_μ at (x, y, ξ), and, for brevity, we write μ for μ_k. Traditionally, a Newton-based method is favored be-

cause, once a sufficiently small neighborhood of a solution is reached (the "domain of convergence"), α can be set to 1.0, and the pure Newton step p_μ produces quadratic convergence to the solution. Outside this neighborhood, however, the pure Newton step is of little use. Away from the solution, calculation of p and α must be done in a way that guarantees sufficient reduction in the objective or, more generally, in a *merit function*, so as to ensure convergence. Fortunately, there is a simple criterion for the detection of a domain of convergence: numerical positive definiteness [2] in H_μ. Hence, practical algorithms are centered on linear-system solvers designed to quickly detect indefiniteness in H_μ, i.e., *negative curvature* in B_μ, when it exists, and in this case to quickly calculate a useful alternate search direction. In order that eventual convergence to a local minimizer can be assured, the resulting so called *modified* Newton direction must have sufficient descent ($g_\mu^T p < -\delta_1 \|p\| \|g\| < 0$), and sufficiently negative curvature ($p^T H_\mu p < -\delta_2 \|p\|^2 < 0$). The partial 2nd-order information used by a modified Newton direction typically leads to a faster approach to the domain of convergence for Newton's method than that obtained by simple steepest descent [Gill et al., 1981].

An important component of this approach is the calculation of a *direction of sufficiently negative curvature*, i.e., d such that $d^T H_\mu d < -\epsilon \|d\|^2$ for fixed ϵ independent of iteration index k. Such directions are particularly useful for avoiding local minima on problems with many local extrema. In our formulation of placement, local minimizers are ubiquitous, and aggressive use of directions of negative curvature is essential.

Our method for approximately solving (4.16) is based on a Lanczos formulation of conjugate gradients with extensions [Nash, 1984; Kong and Shinnerl, 2001]. Each step requires the evaluation of a matrix-vector product with the barrier function Hessian, H_μ. As this matrix is too large and dense to be efficiently stored and manipulated, a directed difference is used:

$$H_\mu v \approx \frac{1}{\epsilon} \left(g((x, y, \xi) + \epsilon v) - g(x, y, \xi) \right) . \tag{4.17}$$

Evaluation of this expression is the bulk of the computational effort in our approach and is discussed below. When sufficient indefiniteness in H_μ is detected, the iterations are modified and quickly terminated, producing a direction of both sufficient descent and sufficiently negative curvature. Once a neighborhood of a local minimizer is found, the Hessian of the barrier function becomes positive definite, and an ap-

[2] If the smallest eigenvalue of H_μ is not large enough, rounding error may interfere with the detection of positive definiteness in H_μ.

proximation to the Newton step is obtained. Early termination of the conjugate gradient (CG) steps can be used to make the approximation just accurate enough to ensure superlinear convergence to barrier subproblem solutions. To limit the number of CG iterations — we typically perform no more than ten — effective preconditioning of (4.16) is essential. For this purpose, generic incomplete LU (truncated Gaussian elimination) preconditioners [Saad, 1996; Lin and Moré, 1999] for (4.16) are sufficient.

It is well known that, as μ decreases to zero, ill conditioning in (4.16) increases as $\mathcal{O}(1/\mu)$. In order to preserve accuracy once μ is sufficiently small, the search direction must be decomposed into two separately computed well-conditioned orthogonal components. Such a decomposition may require an explicit approximation of the tightest constraints (those whose values are closest to zero) and their tangent space, and such calculations may be prohibitively expensive without ad hoc techniques. In our case, however, only relatively large values of μ are needed in $\mathrm{BP}(\mu, \xi)$, and thus no such orthogonal decomposition is necessary.

Evaluations of the barrier function and its derivatives (4.17) are the most computationally intensive tasks associated with our approach. Direct evaluation is $\mathcal{O}(N^2)$ due to the pairwise nonoverlap constraints. We have implemented special force-directed acceleration techniques that render these evaluations $\mathcal{O}(N)$ [Greengard, 1988; Appel, 1985; Kong, 2001]. However, the associated proportionality constants make these techniques attractive only once N exceeds roughly 1000. As our nonlinear programming formulation and solver are useful only for N below or slightly above this threshold, these techniques do not currently provide us with much gain.

3.3 Discrete Refinement

At all levels of refinement, localized permutations of cells are used to improve total wirelength at very little cost in runtime. First we partition the placement area into a set of regular slots and assign cells into slots by linear assignment. Then we permute small subsets of cells to decrease the objective.

The algorithm is based on the concepts of ϵ-neighborhood and λ-exchange [Goto, 1981]. Holding all cell coordinates except v's fixed in (4.4), we can compute v's optimal slot location directly by a simple 2-dimensional quadratic wirelength minimization. Suppose v's optimal slot location is row r, column c. Cells located in slots at row i, column j where $|i-r|+|j-c| \leq \epsilon$ are called ϵ-neighbors of cell v. For example, in Figure 3.3, suppose the optimal slot location of cell A is occupied

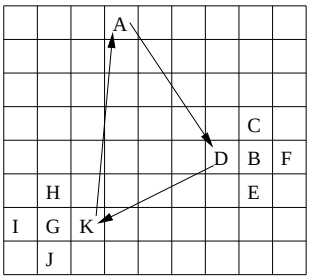

Figure 4.8. ϵ-neighborhood

by cell B. A's 1-neighbors are $\{B, C, D, E, F\}$. Similarly, assuming that D's optimal slot is occupied by G, we say cell D's 1-neighbors are[3] $\{G, H, I, J, K\}$.

In [Goto, 1981], starting from a cell v_1, we compute all of its ϵ-neighbors; then for each cell in v_1's ϵ-neighborhood, compute its ϵ-neighbors, and so on. This procedure generates a search tree, each leaf of which defines a cell-exchange sequence. For example, part of the search tree from cell A is shown in Figure 4.8. With $\epsilon = 1, \lambda = 3$, leaf K defines the following exchange sequence: $A \rightarrow D \rightarrow K \rightarrow A$, i.e., move cell A to D's slot, move D to K's slot, and move K to A's slot. The best cell exchange sequence will be chosen, or no exchange is made, if the original placement has a smaller cost. This method has two major drawbacks: first, the size of the search tree grows very quickly with slight increase of ϵ and λ; second, the cell exchange sequence may not be the best possible. Intuitively, moving a cell into its ϵ-neighborhood has a high probability of reducing the objective function value, but the last step, moving cell v_λ to the slot of v_1, may not be good.

To address these problems, we revise the λ-exchange procedure in [Goto, 1981] as follows. Suppose v_1 is the starting cell. We compute its ϵ-neighbors and randomly pick one cell, say v_2, from these. Then for v_2, we compute its ϵ-neighbors, and randomly pick one cell, and continue in this fashion until we have λ cells. Then for these λ cells, we try all of their placement permutations (the total number is $\lambda!$) and exchange cells according to the least cost permutation. For example, suppose we pick cells A, D, and K. Then all six permutations will be tried: no exchange, $A \leftrightarrow D$, $A \leftrightarrow K$, $D \leftrightarrow K$, $A \rightarrow D \rightarrow K \rightarrow A$, $A \rightarrow K \rightarrow D \rightarrow A$. The

[3]Our definition is different from the one in [Goto, 1981], where ϵ is defined to be the number of child nodes for each node in the search tree. For this example, one would say $\{B, C, D, E, F\}$ are 5-neighbors of A.

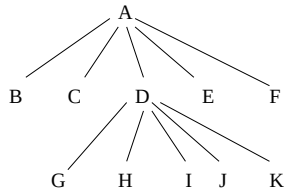

Figure 4.9. Search Tree from A

number of solutions we search still goes exponentially with λ, but not with ϵ.

The benefit of randomly selecting ϵ-neighbors of the optimal slot is that it supports multiple passes across the placement region. In our current implementation, we use up to 10 passes with $\epsilon = 1, \lambda = 5$. Fewer passes are used if they do not reduce the wirelength significantly. Experimentally we found that our algorithm usually finds better solutions.

3.4 Numerical Experiments

Table 3.4 displays the layout parameters of our test circuits. The first four circuits (*struct, biomed, avqsmall, avqlarge*) are the largest ones from the 1993 MCNC layout benchmark sets [Kozminski, 1991]. The results for *struct* and *biomed* were obtained *without* use of FMM. The next five circuits are from the ISPD 1998 benchmark suite [Alpert, 1998]. We tested our algorithms on all 18 circuits in the ISPD98 suite. The results shown are for five of these circuits and demonstrate our algorithm's scalability. Both the solution-quality and runtime trends are the same on the circuits for which results are not shown.

We ran all experiments on a Sun UltraSparc-2/168MHz workstation with 512MB memory. After the global placement, we use DOMINO for detailed placement. CPU times are in seconds; all times listed include the time spent in detailed placement. Wire length of each net is computed as the half perimeter of the enclosing bounding box.

3.4.1 Impact of Nonlinear Programming. For circuit *biomed*, we present more detailed run-time data illustrating the impact of coarse-level constrained nonlinear programming on the final placement result. For this circuit, we ran MPL six times. The first column of the results (Table 4.3) shows the number of levels at which nonlinear programming was used. In the first run, nonlinear programming was not used at all; hence, 0 appears in the first column. In the second run (level = 1), the NP-engine was run only at the coarsest level. In the

Table 4.2. Test Circuits

circuit	# cells	# nets	# pads	# rows
struct	1888	1920	64	21
biomed	6417	5742	97	46
avqsmall	21854	22124	64	80
avqlarge	25114	25384	64	80
ibm04	27220	31970	287	116
ibm07	45639	48117	287	151
ibm09	53110	60902	285	162
ibm14	147088	152772	517	271
ibm17	184752	189581	743	303

Table 4.3. Impact of Nonlinear Programming on Circuit *biomed*: 15% Reduction in Wirelength

levels	wirelength	CPU
0	4.35e+06	134.2
1	4.04e+06	136.6
2	4.01e+06	127.3
3	3.99e+06	136.9
4	3.91e+06	163.3
5	3.86e+06	336.9

third run (level = 2), the NP-engine was run only at the coarsest and the second coarsest levels, and so on. As the number of levels to which the NP-engine is applied increases, lower wirelength results are obtained at a cost of increased CPU time. Similar trends were observed on other circuits.

Surprisingly, however, the wirelength/runtime tradeoff curve is not completely monotone. For example, in Table 4.3, using two levels of nonlinear programming is slightly faster than using just one level. The reason, in this particular instance, is that the second-coarsest nonlinear programming problem reduces the total overlap much more than the first. Consequently, less time is required for the discrete refinement step to converge; slot-exchange produces little gain and is terminated earlier. Thus, at coarser levels, the use of nonlinear programming is cost effective, but eventually, at finer levels, its run time becomes excessive.

3.4.2 Comparison with GORDIAN-L. We compared our algorithm with GORDIAN-L followed by DOMINO; these programs use the

Table 4.4. Comparison with GORDIAN-L-DOMINO

circuit	mPL-DOMINO		GORDIAN-L vs. mPL	
	wirelength	CPU	wirelength	CPU
struct	7.68e+05	53.0	0.99	0.46
biomed	3.86e+06	336.9	1.01	0.69
avqsmall	1.25e+07	893.5	0.94	2.12
avqlarge	1.37e+07	814.0	0.95	2.56
ibm04	7.40e+06	1453.3	0.93	2.45
ibm07	1.10e+07	602.0	0.99	13.9
ibm09	1.27e+07	680.4	0.93	19.7
ibm14	4.33e+07	2965.7	0.94	20.5
ibm17	7.43e+07	8941.1	0.91	14.8

PROUD circuit format [Tsay and Kuh, 1991; Tsay et al., 1988b; Tsay et al., 1988a]. This format specifies only the relative locations of the pads, not absolute locations. To ensure a unique solution, quadratic placement requires fixed pads. We first ran GORDIAN-L-DOMINO on the test circuits and extracted the pads' locations from the output. With these pad locations fixed, we then ran mPL for global placement followed by DOMINO [Doll et al., 1994] for detailed placement. This means of comparison may put mPL at some disadvantage, since it might otherwise choose the absolute pad locations by itself to further reduce wirelength.

The results are summarized in Table 3.4.2. Columns "GORDIAN-L vs. mPL" shows the results of GORDIAN-L-DOMINO divided by those obtained by our approach. Thus, values larger than 1.0 mean that our approach is better or faster.

Our approach gives results on average 4.9% worse than GORDIAN-L-DOMINO but uses significantly less CPU time, especially as the number of cells increases. For example, for the largest circuit *ibm17* with around 184K cells, our approach can obtain a placement in about 2.5 hours with relatively good quality, only 9.4% percent from the result of GORDIAN-L-DOMINO. This suggests our approach has better scalability and can be used to achieve accurate placement estimation during earlier design phases.

4. Conclusions

The results listed in the preceding section highlight both the promise and the challenge of applying multilevel optimization to circuit placement. Simple relaxations, coarsenings, and interpolations dramatically accelerate *initial* progress toward good solutions. That is, obtaining an

objective value within roughly 5–10% of the known best value, say, $\hat{\ell}$, can usually be done with only 5–10% of the run-time used by leading methods to obtain $\hat{\ell}$. Sustaining the rapid pace of reduction in the objective all the way to and below $\hat{\ell}$, however, is far less easy and appears to require more sophisticated techniques.

Our experience with mPL shows the importance of employing relaxations appropriate to the multilevel framework. Our attempt to perform *global relaxations*, in which all clusters at a given level are moved simultaneously, appears impractical, except as a solver at the coarsest level. Pointwise information changes too rapidly to support efficient continuous nonlinear approximation simultaneously over the entire circuit. Because our first nonlinear programming model could not be used efficiently at levels with more than about 1000 cells, we relied mainly on the more limited discrete relaxation described in Section 3.3. This technique produced rapid gain initially but lacked power at later stages. What seems needed is a method of relaxation with the correct balance of power and scalability, closer perhaps to the network-flow formulations in [Doll et al., 1994] and [Hur and Lillis, 2000]. We are currently investigating this approach as part of the following list of enhancements.

1 Floorplanning at the coarsest levels: support variable and diverse cluster shapes and sizes. Modify the shapes, sizes, and orientations during relaxation.

2 Bin-density-based overlap control with local relaxations by linear and nonlinear programming (Section 2.4.1).

3 Alternative smoothings of the bounding-box wirelength objective during continuous relaxation (Section 1.1.1).

4 First-Choice coarsening (Chapter 3) alternating on the primal and dual hypergraph representations. The dual of hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is $\mathcal{H}^* = (\mathcal{E}, \mathcal{V}^*)$, where each $v \in \mathcal{V}$ is identified with a set $v^* \equiv \{e \in \mathcal{E} \,|\, v \in e\}$, and $\mathcal{V}^* \cong \mathcal{V}$ is the set of all such v^*.

5 Aggregation and interpolation by fractional representation rather than deterministic clustering: AMG-style interpolation (Chapter 1). I.e., associate each cell at a finer level with one *or more* clusters at the adjacent coarser levels by a convex combination of weights. The interpolated position of the cell is the correspondingly weighted average of its associated clusters.

6 Multiple V-cycles with reclustering and multilevel relaxation at each level of every interpolation phase (Chapter 1).

As the size and complexity of integrated circuits continue to grow, the correlation between the high-level abstract models and their final physi-

cal implementations becomes more difficult to maintain. For consistency and efficiency in the final design, more feedback from later stages must be propagated to the earlier stages, and more iterations over the design flow are required. This increase in iterations increases the need for fast and scalable algorithms.

Chapters 2 and 3 of this book describe multilevel algorithms for graph and hypergraph partitioning. Multilevel methods for partitioning achieve superior objective values in runtimes orders of magnitude less than those of the previously leading algorithms. As this book goes to press, the published results in multilevel circuit placement are somewhat less mature, but they suggest a strong trend toward a comparable level of success.

Acknowledgments

The authors gratefully acknowledge the support of Intel Corporation, the National Science Foundation (NSF #CCR-9901153), and the Semiconductor Research Consortium (SRC #99-TJ-686).

References

Alpert, C., Huang, J.-H., and Kahng, A. (1997). Multilevel circuit partitioning. In *Proc. 34th IEEE/ACM Design Automation Conf.*

Alpert, C. J. (1998). The ISPD98 circuit benchmark suite. In *Proc. Intl Symposium on Physical Design*, pages 80–85.

Appel, A. (1985). An efficient program for many-body simulation. *SIAM J. Sci. Stat. Comp.*

Betz, V. and Rose, J. (1997). VPR: A new packing, placement, and routing tool for FPGA research. In *Proc. Intl. Workshop on FPL*, pages 213–222.

Brenner, U. and Rohe, A. (Apr 2002). An effective congestion-driven placement framework. In *Proc. International Symposium on Physical Design.*

Breuer, M. (Oct 1977). Min-cut placement. *J. Design Automat. Fault Tolerant Comp.*, 1(4):343–362.

Caldwell, A., A.B.Kahng, and Markov, I. (2000). Improved algorithms for hypergraph partitioning. In *Proc. IEEE/ACM Asia South Pacific Design Automation Conf.*

Chan, T., Cong, J., Kong, T., and Shinnerl, J. (Nov 2000). Multilevel optimization for large-scale circuit placement. In *Proc. IEEE International Conference on Computer Aided Design*, pages 171–176, San Jose, CA.

Chang, C., Cong, J., Pan, Z., and Yuan, X. (Apr 2002). Physical hierarchy generation with routing congestion control. In *Proc. ACM International Symposium on Physical Design*, pages 36–41, San Diego, CA.

Cheng, C. and Kuh, E. (Jul 1984). Module placement based on resistive network optimization. *IEEE Transactions on Computer-Aided Design*, CAD-3(3).

Cong, J. (2001). An interconnect-centric design flow for nanometer technologies. *Proceedings of the IEEE*, 89(4):505–528.

Cong, J., Kahng, A. B., and Leung, K. S. (1999). Efficient algorithms for the minimum shortest path steiner arborescence problem with applications to vlsi physical design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 17(1):24–39.

Cong, J. and Lim, S. (Jan 2000). Edge separability based circuit clustering with application to circuit partitioning. In *Proc. Asia South Pacific Design Automation Conference*, pages 429–434, Yokohama Japan.

Doll, K., Johannes, F., and Antreich, K. (Oct 1994). Iterative placement improvement by network flow methods. *IEEE Transactions on Computer-Aided Design*, 13(10).

Dunlop, A. and Kernighan, B. (Jan 1985). A procedure for placement of standard-cell vlsi circuits. *IEEE Transactions on Computer-Aided Design*, CAD-4(1).

Eisenmann, H. and Johannes, F. M. (1998). Generic global placement and floorplanning. In *Proc. 35th ACM/IEEE Design Automation Conference*, pages 269–274.

Fiacco, A. V. and McCormick, G. P. (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley and Sons, Inc., New York, London, Sydney and Toronto.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 721–741.

Gill, P. E., Murray, W., Saunders, M. A., Tomlin, J., and Wright, M. H. (1986). On projected Newton methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36:183–209.

Gill, P. E., Murray, W., and Wright, M. H. (1981). *Practical Optimization*. Academic Press, London and New York. ISBN 0-12-283952-8.

Goldberg, A. and Tarjan, R. (1988). A new approach to the maximum flow problem. *Journal of the ACM*, pages 921–940.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, third edition.

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

Greengard, L. (1988). *The Rapid Evaluation of Potential Fields in Particle Systems*. M.I.T. Press, Cambridge, Massachusetts.

Hur, S.-W. and Lillis, J. (Jun 1999). Relaxation and clustering in a local search framework: Application to linear placement. In *Proc. ACM/IEEE Design Automation Conference*, pages 360–366, New Orleans, LA.

Hur, S.-W. and Lillis, J. (Nov 2000). Mongrel: Hybrid techniques for standard-cell placement. In *Proc. IEEE International Conference on Computer Aided Design*, pages 165–170, San Jose, CA.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395.

Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. (1997). Multilevel hypergraph partitioning: Application in vlsi domain. In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 526–529.

Kirkpatrick, S., Jr. C. G., and Vecci, M. (1983). Optimization by simulated annealing,. *Science*, 220:671ff.

Kleinhans, J. M., Sigl, G., Johannes, F. M., and Antreich, K. J. (1991). Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Trans. on Computer-Aided Design*, CAD-10:356–365.

Kong, T. (Oct 2001). An adaptation of the fast multipole method to a nonanalytic potential field for overlapping disks in the plane. Report 010038, Computer Science Dept., University of California, Los Angeles.

Kong, T. and Shinnerl, J. (Nov 2001). Implementation of lanczos-based preconditioned modified conjugate gradients. Report 010037, Computer Science Dept., University of California, Los Angeles.

Kozminski, K. (1991). Benchmarks for layout synthesis. In *Proc. 28th ACM/IEEE Design Automation Conference*, pages 265–270.

Lim, S. (2000). *Performance-Driven Circuit Partitioning*. PhD thesis, Computer Science Dept., University of California, Los Angeles.

Lin, C.-J. and Moré, J. J. (1999). Incomplete cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing*, 21(1):24–45.

Murray, W. and Wright, M. H. (1994). Line search procedures for the logarithmic barrier function. *SIAM J. on Optimization*, 4, number 2:229–246.

Nagamochi, H. and Ibaraki, T. (1992). Computing edge connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Math.*, pages 54–66.

Nash, S. G. (1984). Newton-type minimization via the Lanczos method. *SIAM J. on Numerical Analysis*, 21:770–788.

Quinn, N. and Breuer, M. (1979). A force-directed component placement procedure for printed circuit boards. *IEEE Trans. on Circuits and Systems CAS*, CAS-26:377–388.

Saad, Y. (1996). *Iterative methods for sparse linear systems*. PWS publishing, Pacific Grove, California.

Sankar, Y. and Rose, J. (1999). Trading quality for compile time: Ultra-fast placement for FPGAs. In *FPGA '99, ACM Symp. on FPGAs*, pages 157–166.

Sechen, C. (1988). *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer Academic Publishers.

Sigl, G., Doll, K., and Johannes, F. M. (1991). Analytical placement: A linear or a quadratic objective function? In *Proc. 28th ACM/IEEE Design Automation Conference*, pages 427–432.

Stander, J. and Silverman, B. (1994). Temperature schedules for simulated annealing. *Statistics and Computing*, pages 21–32.

Sun, W.-J. and Sechen, C. (Mar 1995). Efficient and effective placement for very large circuits. *IEEE Trans. on Computer-Aided Design*, pages 349–359.

Swartz, W. (2002). http://www.internetcad.com.

Tsay, R., Kuh, E., , and Hsu, C. (1988a). Proud: A fast sea-of-gates placement algorithm. *IEEE Design and Test of Computers*, pages 44–56.

Tsay, R. and Kuh, E. S. (1991). A unified approach to partitioning and placement. *IEEE Trans. Circuits and Systems*, 38(5):521–533.

Tsay, R., Kuh, E. S., and Hsu, C. (1988b). Proud: A sea-of-gates placement algorithm. *IEEE Design and Test of Computers*, 12:44–56.

Vygen, J. (1997). Algorithms for large-scale flat placement. In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 746–751.

Wang, M., Yang, X., and Sarrafzadeh, M. (Apr 2000). Dragon2000: Fast standard-cell placement for large circuits. *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 260–263.

Yang, X. (2002). *Congestion and Timing Optimization for Standard-Cell Placement*. PhD thesis, Computer Science Dept., University of California, Los Angeles.