

---

# Multiscale Optimization in VLSI Physical Design Automation

Tony F. Chan<sup>1</sup>, Jason Cong<sup>2</sup>, Joseph R. Shinnerl<sup>2</sup>, Kenton Sze<sup>1</sup>, Min Xie<sup>2</sup>, and Yan Zhang<sup>2</sup>

<sup>1</sup> UCLA Mathematics Department, Los Angeles, California 90095-1555, USA.

{chan,nksze}@math.ucla.edu

<sup>2</sup> UCLA Computer Science Department, Los Angeles, California 90095-1596, USA.

{cong,shinnerl,xie,zhangyan}@cs.ucla.edu

**Summary.** The enormous size and complexity of current and future integrated circuits (IC's) presents a host of challenging global, combinatorial optimization problems. As IC's enter the nanometer scale, there is increased demand for scalable and adaptable algorithms for VLSI *physical design*: the transformation of a logical-temporal circuit specification into a spatially explicit one. There are several key problems in physical design. We review recent advances in multiscale algorithms for three of them: partitioning, placement, and routing.

**Key words:** VLSI, VLSICAD, layout, physical design, design automation, scalable algorithms, combinatorial optimization, multiscale, multilevel

## 1 Introduction

In the computer-aided design of very-large-scale integrated circuits (VLSI-CAD), *physical design* is concerned with the computation of a precise, spatially explicit, geometrical layout of circuit modules and wires from a given logical and temporal circuit specification. Mathematically, the various stages of physical design generally amount to extremely challenging mixed integer--nonlinear-programming problems, including large numbers of both continuous and discrete constraints. The numbers of variables, nonconvex constraints, and discrete constraints range into the tens of millions and beyond. Viewed discretely, the solution space grows combinatorially with the number of variables. Viewed continuously, the number of local extrema grows combinatorially. The principal goals of algorithms for physical design are (i) speed and scalability; (ii) the ability to accurately model and satisfy complex physical constraints; and (iii) the ability to attain states with low objective values subject to (i) and (ii).

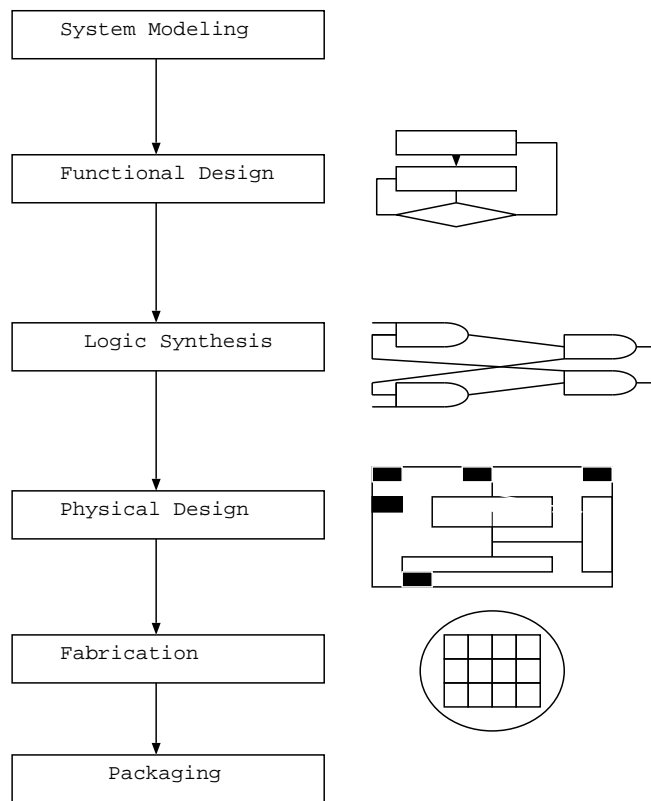
Highly successful multiscale algorithms for circuit partitioning first appeared in the 1990s [CS93, KAKS97, CAM00]. Since then, multiscale metaheuristics for VLSICAD physical design have steadily gained ground. Today they are among the leading methods for the most critical problems, including partitioning, placement and routing. Recent experiments strongly suggest, however, that the gap between optimal and attainable solutions remains quite substantial, despite the burst of progress in the last decade. Thus, an improved understanding of the application of multiscale methods to the large-scale combinatorial optimization problems of physical design is widely sought [CS03].

A brief survey of some leading multiscale algorithms for the principal stages of physical design — partitioning, placement, and routing — is presented here. First, the role of physical design in VLSICAD is briefly described, and recent experiments revealing a large optimality gap in the results produced by leading placement algorithms are reviewed.

### 1.1 Overview of VLSI Design

As illustrated in Figure 1, VLSI design can be divided into the following steps: system modeling, architectural synthesis, logic synthesis, physical design, fabrication, packaging.

- *System modeling.* The concepts in the designer’s mind are captured as a set of computational operations and data dependencies subject to constraints on timing, chip area, etc.
- *Functional Design.* The resources that can implement the system’s operations are identified, and the operations are scheduled. As a result, the *control logic* and *datapath* interconnections are also identified. Functional design is also called *high-level synthesis*.
- *Logic synthesis.* The high-level specification is transformed into an interconnection of gate-level boolean primitives — *nand*, *xor*, etc. The circuit components that can best realize the functions derived in functional design are assembled. Circuit delay and power consumption are considered at this step. The output description of the interconnection between different gate-level primitives is usually called a *netlist* (Section 1.2).
- *Physical design.* The actual spatial layout of circuit components on the chip is determined. The objectives during this step usually include total wirelength, maximum signal propagation time (“performance”), etc. Physical design can be further divided into steps including partitioning, floorplanning, placement, and routing; these are described in Section 1.2 below.
- *Fabrication.* Fabrication involves the deposition and diffusion of material onto a silicon wafer to achieve desired electronic circuit properties. Since designs will make use of several layers of metal for wiring, masks mirroring the layout on each metal layer will be applied in turn to produce the required interconnection pattern by photolithography.



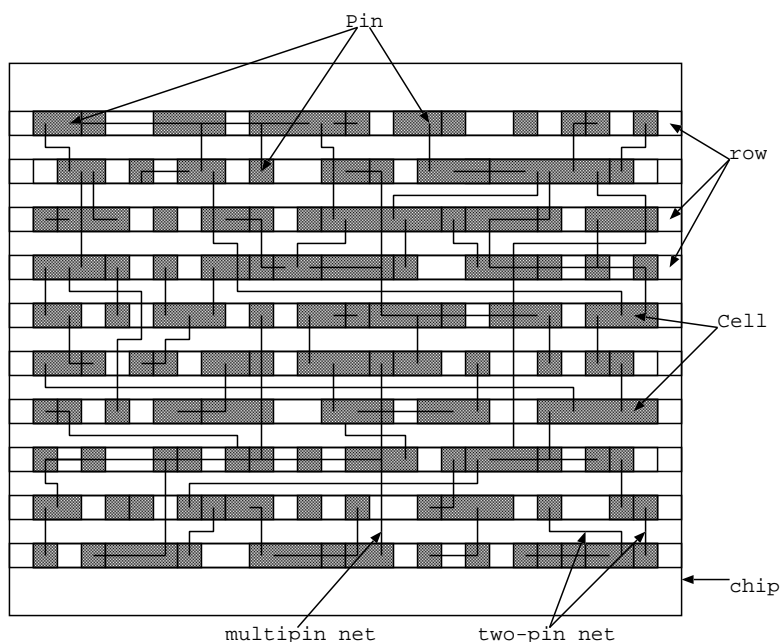
**Fig. 1.** VLSI design includes system specification, functional design, logic design, physical design, fabrication, and packaging.

- *Packaging.* The wafer is diced into individual chips, which are then packaged and tested.

As the fundamental physical barriers to continued transistor miniaturization begin to take shape, efforts in synthesis and physical design have intensified. The main component stages of physical design are reviewed next in more detail.

## 1.2 Overview of VLSI Physical Design

At the physical level, an integrated circuit is a collection of rectangular modules connected by rectangular wires. The wires are arranged in parallel, horizontal layers stacked along the  $z$  axis; the wires in each layer are also parallel. Each module has one face in a prescribed rectangle in the  $xy$ -plane known as the *placement region*. However, different modules may intersect with different numbers of metal wiring layers. After logic synthesis, most modules are

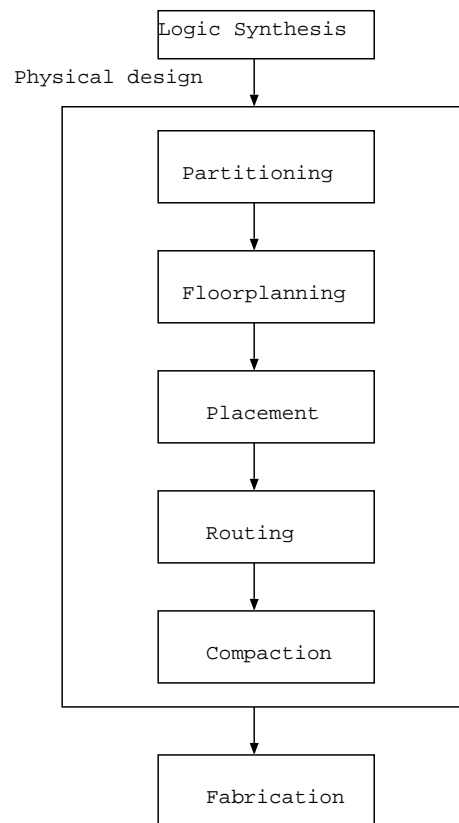


**Fig. 2.** A 2-D illustration of the physical elements of an integrated circuit. The routing layers have been superimposed.

selected from a *cell library* and assigned to logic elements as part of a process known as *technology mapping*. These modules are called *standard cells* or simply *cells*. Their widths ( $x$ -direction) may vary freely, but their heights ( $y$ -direction) are taken from a small, discrete set. Other, larger modules may represent separately designed elements known as *IP blocks* (intellectual-property blocks) or *macros*; the heights of these larger blocks typically do not fall within the standard-cell heights or their integer multiples. The *area* of a module refers to the area of its cross-sections in the  $xy$ -plane.

A signal may propagate from a source point on a module to any number of sinks on other modules. The source and sinks together define a *net*. At steady state, a net is an equipotential of the circuit. A connection point between a net and a module is called a *pin*. Hence, a net may be abstracted as either a set of pins, or, less precisely, as the set of modules to which these pins belong. The *netlist* specifies the nets of a circuit as lists of pins and is a product of logic synthesis (Section 1.1). The physical elements of an IC are illustrated in Figure 2.

As illustrated in Figure 3, VLSI physical design proceeds through several stages, including partitioning, floorplanning, placement, routing, and compaction [DiM94, She99].



**Fig. 3.** Stages in the physical design of integrated circuits.

### Partitioning

Due to the complexity of integrated circuits, the first step in physical design is usually to divide a design into subdesigns. Considerations include area, logic functionality, and interconnections between subdesigns. Partitioning is applied recursively until the complexity in each subdesign is reduced to the extent that it can be handled efficiently by existing tools.

### Floorplanning

The shapes and locations of the components within each partitioning block are determined at this stage. These components are also called blocks and may be reshaped. Floorplanning takes as input a set of rectangular blocks, their fixed areas, their allowed shapes expressed as maximum aspect ratios, and the connection points on each block for the nets containing it. Its output includes the shape and location of each block. Constraints may involve the location

of a block and/or adjacency requirements between arbitrary pairs of blocks. The blocks are not allowed to overlap. Floorplanning is typically limited to problems with a few hundred or a few thousand blocks. As such, it is typically used as a means of coarse placement on a simplified circuit model, either as a precursor to placement or as a means of guiding logic synthesis to a physically reasonable solution.

### Placement

In contrast to floorplanning, *placement* treats the shapes of all blocks as fixed; i.e., it only determines the location of each block on the chip. The variables are the  $xy$ -locations of the blocks; most blocks are standard cells (Section 1.2). The  $y$ -locations of cells are restricted to standard-cell rows, as in Figure 2. Placement instance sizes range into the tens of millions and will continue to increase.

Placement is usually divided into two steps: global placement and detailed placement. Global placement assigns blocks to certain subregions of the chip without determining the exact location of each component within its subregion. As a result, the blocks may still overlap. Detailed placement starts from the result of global placement, removes all overlap between blocks, and further optimizes the design. Placement objectives include the estimated total wirelength needed to connect blocks in nets, the maximum expected wiring congestion in subsequent routing, and/or the timing performance of the circuit. A simplified formulation of placement is given in Section 1.5.1.

### Routing

With the locations of the blocks fixed, their interconnections as specified by the netlist must be realized. That is, the shapes and locations of the metal wires connecting the blocks must be determined. This wiring layout is performed not only within the placement region but also in a sequence of parallel metal *routing layers* above it. Cells constitute routing *obstacles* in layers which pass through them. Above the cells, all the wires in the same routing layer are parallel to the same coordinate axis, either  $x$  or  $y$ . Routing layers alternate in the direction of their wires. Interlayer connections are called *vias*.

The objective of routing is to minimize the total wirelength while realizing all connections subject to wire spacing constraints within each layer. In addition, the timing performance of the circuit may also be considered. Routing is usually done in two steps, global routing and detailed routing. Global-routing algorithms determine a route for each connection in terms of the regions it passes through, without giving the exact coordinates of the connection. During this phase, the maximum congestion in each region must be kept below a certain limit. The goal of detailed routing is to realize a point-to-point path for each net following the guidance given by the global routing. It is in this step that the geometric location and shape of each wire is determined. Due

to the sequential nature of most routing algorithms, a 100% completion rate may not be obtained for many designs. An additional step called rip-up and reroute is used to remove a subset of connections already made and find alternate routes, so that the overall completion rate can be improved. The rip-up and reroute process works in an iterative fashion until either no improvement can be obtained or a certain iteration limit is reached.

### Compaction

Compaction is used to reduce the white space on the chip so that the chip area can be minimized. This step involves heavy manipulation of geometric objects. Depending on the movement these geometric objects are allowed, compaction can be categorized into 1-D compaction or 2-D compaction. However, the chip area for many of the designs are given as fixed. In this case, instead of compacting the design, intelligent allocation of white space can be adopted to further optimize certain metrics, e.g., routing congestion, maximum temperature, etc.

### 1.3 Hypergraph Circuit Model for Physical Design

An integrated circuit is abstracted more accurately as a hypergraph than as a graph, because each of its nets may connect not just a pair of nodes but rather an arbitrarily large subset of nodes. The details of the abstraction depend on the point in the design flow where it is used. A generic definition of the hypergraph concept is given here. In later sections, specific instances of it are given for partitioning, placement, and routing.

A *hypergraph*  $H = \{V, E\}$  consists of a set of vertices  $V = \{v_1, v_2, \dots, v_n\}$  and a set of hyperedges  $E = \{e_1, e_2, \dots, e_m\}$ . Each *hyperedge*  $e_j$  is just some subset of  $V$ , i.e.,  $e_j = \{v_{j_1}, v_{j_2}, \dots, v_{j_k}\} \subset V$ . Each hyperedge corresponds to some net in the circuit. Each vertex  $v_i$  may have weight  $w(v_i)$  associated with it, e.g., area; each hyperedge  $e_j$  may have weight  $w(e_j)$  associated with it, e.g., timing criticality. In either case, the hypergraph itself is said to be weighted as well. The number of vertices contained by  $e_j$  (we will also say “connected by”  $e_j$ ) is called the *degree* of  $e_j$  and is denoted  $|e_j|$ . The number of hyperedges containing  $v_i$  is called the *degree* of  $v_i$  and is denoted  $|v_i|$ .

Every hypergraph, weighted or unweighted, has a dual. The dual hypergraph  $H' = \{V', E'\}$  of a given hypergraph  $H = \{V, E\}$  is defined as follows. First, let  $V' = E$ ; if  $H$  is weighted, then let  $w(v'_i) = w(e_i)$ . Second, for each  $v_i \in V$ , let  $e'_i \in E'$  be the set of  $e_j \in E$  that contain  $v_i$ . If  $H$  is weighted, then let  $w(e'_i) = w(v_i)$ . It is straightforward to show that  $H''$ , the dual of  $H'$ , is isomorphic to  $H$ .

### 1.4 The Gigascale Challenge

Since the early 1960s, the number of transistors in an integrated circuit has doubled roughly every 18 months. This trend, known as Moore’s Law, is ex-

pected to continue into the next decade. Projected statistics from the 2003 International Technology Roadmap for Semiconductors (ITRS 2003) are summarized in Table 1.

Production year	2003	2004	2005	2006	2007	2008	2009
DRAM 1/2 pitch (nm)	100	90	80	70	65	57	50
M transistors/chip	153	193	243	307	386	487	614
Chip size (mm <sup>2</sup> )	140	140	140	140	140	140	140
Local clock (MHz)	2976	4171	5204	6783	9285	10972	12369
Wiring levels	13	14	15	15	15	16	16

**Table 1.** Circuit Statistics Projections from ITRS 2003 [itr].

Over 40 years of this exponential growth have brought enormous complexity to integrated circuits, several hundred million transistors integrated on a single chip. Although the power of physical-design algorithms has also increased over this period, evidence suggests that the relative gap between optimal and achievable widens with increasing circuit size (Section 1.5). As the number and heterogeneity of devices on chip continues to increase, so does the difficulty in accurately modeling and satisfying various manufacturability constraints.

Typically, constraints in VLSICAD physical design are concerned with module nonoverlap (“overlap”), signal propagation times (“timing”), wiring congestion (“routability”), and maximum temperature. A detailed survey of the modeling techniques currently used for these conditions is beyond the scope of this chapter. However, the practical utility of any proposed algorithm rests largely in (a) its scalability and (b) its ability to incorporate constraint modeling efficiently and accurately at every step.

Recent research [CCX03b, JCX03] strongly suggests that, over the last few decades, advances in algorithms for physical design have not kept pace with increasing circuit complexity. These studies are reviewed next.

### 1.5 Quantifying the Optimality Gap

As even the simplest formulations of core physical-design problems are NP-hard [She99], practical algorithms rely heavily on heuristics. Meaningful bounds on the deviation from optimal are not yet known for these algorithms as applied to the design of real circuits. However, a recent optimality study of VLSI placement algorithms shows a large gap between solutions from state-of-the-art placement algorithms and the true optima for a special class of synthetic benchmarks.

In this section, the wirelength-driven placement problem is introduced for the purpose of summarizing this study. In Section 3, the role of placement in VLSICAD is considered in more detail, and some recent multiscale algorithms for it are reviewed.

### 1.5.1 The Placement Model Problem

In the given hypergraph-netlist representation  $H = (V, E)$  of an integrated circuit, we require for placement that each  $v_i$  has a given, fixed rectangular shape. We assume given a bounded rectangle  $R$  in the plane whose boundaries are parallel to coordinate axes  $x$  and  $y$ . The orientation of each  $v_i$  will also be assumed prescribed in alignment with the boundaries of  $R$ , although in some cases flipping  $v_i$  across coordinate axes may be allowed. The length of  $v_i$  along the  $x$ -axis is called its *width*; its length along the  $y$ -axis is called its *height*. The vertices  $v_i$  are typically represented at some fixed level of abstraction. Possible levels are, from lowest to highest, transistors, logic gates, standard cells, or macros (cf. Section 1.2). As IC's become more heterogeneous, the mixed-size problem, in which elements from several of these levels are simultaneously placed, increases in importance. The coverage in this chapter assumes the usual level of standard cells.

Interconnections among placed cells (Section 4) are ultimately made not only within  $R$  but also in multiple routing layers above  $R$ ; each routing layer has the same  $x$  and  $y$  coordinates as  $R$  but a different  $z$  coordinate. For this reason, the total area of all  $v_i \in V$  may range anywhere from 50% to 98% or more of the area of  $R$ . With multiple routing layers above placed cells, making metal connections between all the pins of a net can usually be accomplished within the bounding box of the net. Therefore, the most commonly used estimate of the length of wire  $\ell(e_i)$  that will be required for routing a given net  $e_i = \{v_{i_1}, v_{i_2}, \dots, v_{i_j}\}$  is simply the half-perimeter of its bounding box:

$$\ell(e_i) = \left( \max_k x(v_{i_k}) - \min_k x(v_{i_k}) \right) + \left( \max_k y(v_{i_k}) - \min_k y(v_{i_k}) \right). \quad (1)$$

### Wirelength-Driven Model Problem

In the simplest commonly used abstraction of placement, total 2D-bounding-box wirelength is minimized subject to the pairwise nonoverlap, row-alignment, and placement-boundary constraints. Let  $w(R)$  denote the width (along the  $x$ -direction) of the placement region  $R$ , and let  $Y_1, Y_2, \dots, Y_{n_r}$  denote the  $y$ -coordinates of its standard-cell rows' centers; assume every cell fits in every row. With  $(x_i, y_i)$  denoting the center of cell  $v_i$  and  $w_i$  denoting its width, this *wirelength-driven* form of placement may be expressed as

$$\begin{aligned} \min_{(x_i, y_i)} \sum_{e \in E} w(e)\ell(e) & \quad \text{for } \ell(e) \text{ defined in (1)} \\ \text{subject to } y_i \in \{Y_1, \dots, Y_{n_r}\} & \quad \text{all } v_i \in V \\ 0 \leq x_i \leq w(R) - w_i/2 & \quad \text{all } v_i \in V \\ |x_i - x_j| > (w_i + w_j)/2 \text{ or } y_i \neq y_j & \quad \text{all } v_i, v_j \in V. \end{aligned} \quad (2)$$

Despite its apparent simplicity, this formulation captures much of the difficulty in placement. High-quality solutions to (2) generally serve as useful starting

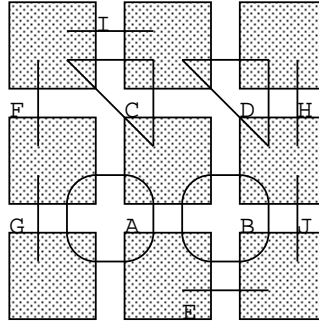


Fig. 4. PEKO generation for  $p = 9$ ,  $D = (6,2,2)$ .

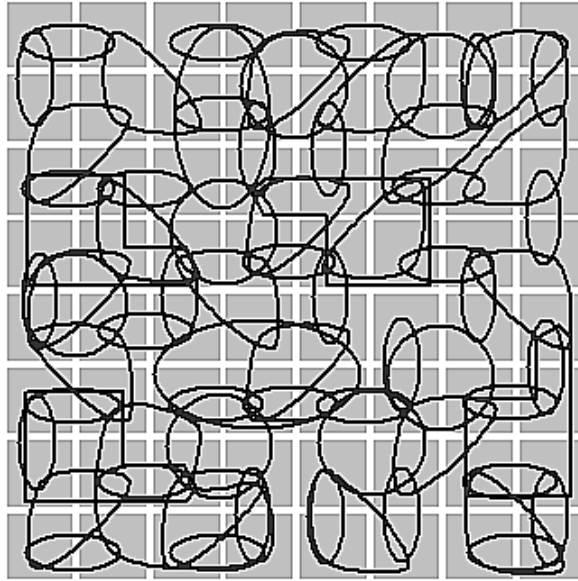
points for more elaborate models of real circuits. The optimality gap for most leading academic tools applied to (2) has been observed to be quite large for the PEKO benchmarks discussed next.

### 1.5.2 Placement Examples with Known Optima (PEKO)

Placement algorithms have been actively studied for the past 30 years. However, there is little understanding of how far computed solutions are from optimal. It is also not known how much the deviation from optimality is likely to grow with respect to problem size. Recently, significant progress was made toward answers to these questions using cleverly constructed placement examples with known optima (PEKO) [CCX03a].

The construction of PEKO can be stated as follows. Given a netlist  $N$ , let  $D(N) = (d_2, d_3, \dots, d_n)$  be the *Net Distribution Vector (NDV)*, where  $d_k$  is the total number of  $k$ -pin nets in the netlist. PEKO examples have all cells of equal size. Given a number  $p$  and a vector  $D$ , we construct a placement example with  $p$  placeable cells such that (i) its netlist has  $D$  as its *NDV* and (ii) it has a known placement of optimal half-perimeter wirelength. The cells are first arranged in a nearly square rectangular region as a regular 2-D array of uniform rows and columns, except possibly the last row, which may not be filled. After that, nets are defined one by one on the cells in such a way that the bounding box for each net has minimal perimeter. Each  $k$ -pin net connects cells within a region of size  $\lceil \sqrt{k} \rceil \times \lceil k / \lceil \sqrt{k} \rceil \rceil$  (or  $\lceil k / \lceil \sqrt{k} \rceil \rceil \times \lceil \sqrt{k} \rceil$ ). The wirelength for each  $k$ -pin net thus constructed is optimal. In the end, the specific netlist is extracted from this placed configuration.

Figure 4 shows an example, where  $p = 9$ ,  $D = (6, 2, 2)$ . Net A is a 4-pin net. Accordingly, it will connect four cells located in a  $2 \times 2$  rectangular region. In Figure 4, it connects the four cells in the lower left corner. The other 4-pin net, B, is placed on the lower right corner. Using the same method, the two 3-pin nets are generated as C and D respectively. This process is repeated until the *NDV* is exhausted. The total wirelength for this example is  $6 \cdot 1 + 2 \cdot 2 + 2 \cdot 2$



**Fig. 5.** An  $8 \times 8$  PEKO instance.

= 14. To mimic real circuits, the *NDV* used in [CCX03a] to generate PEKO are extracted from real circuits [Alp98].

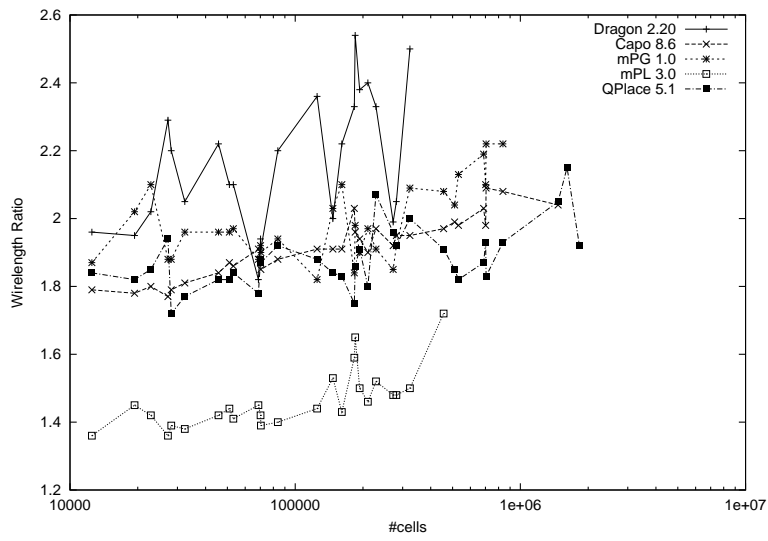
Another, still tiny example with a more realistic *NDV* is shown in Figure 5.

Four state-of-the-art placers from academia including Dragon [WYS00a], Capo [CKM00], mPL [CCKS00], mPG [CCPY02], and one industrial placer, QPlace [Cad99] are compared on PEKO. Figure 6 compares their ratios of attained wirelengths to optimal wirelengths for several instances of PEKO of different size. Overall, their wirelengths are 1.59 to 2.40 times the optimal in the worst cases and are 1.43 to 2.12 times the optimal on average. As for scalability, the average solution quality of each tool deteriorates by an additional 9% to 17% when the problem size increases by a factor of 10. These results indicate significant room for improvement in existing placement algorithms. Run times are shown in Figure 7.

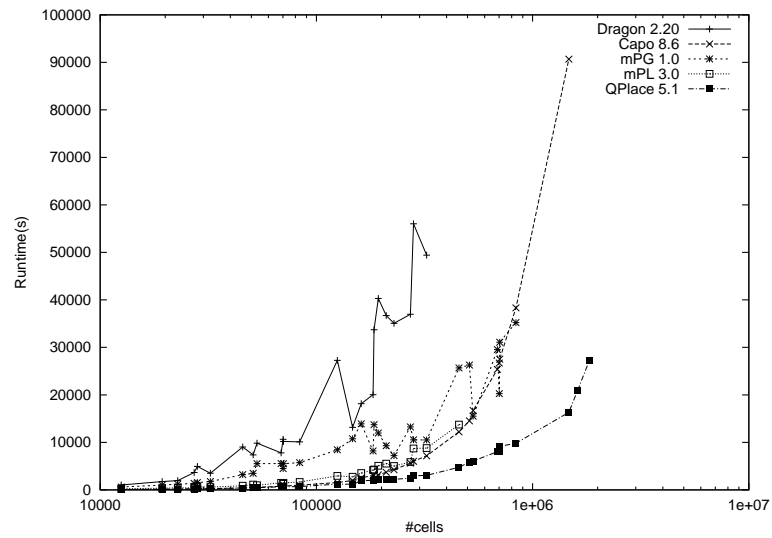
These results have generated great interest among industrial designers and academic researchers, with over 150 downloads by major universities and EDA and semiconductor companies, e.g., Cadence, Synopsys, Magma, IBM, and Intel, etc., in just the first year of their release. There have also been three EE times articles covering the study [Goe03c, Goe03b, Goe03a].

## 1.6 Multiscale Optimization — Opportunities and Obstacles

Concurrently with the steady advances in VLSI design, multiscale methods have emerged as a means of generating scalable solutions to many diverse



**Fig. 6.** Wirelength ratios vs. numbers of cells on PEKO test cases. Each ratio is the wirelength attained by the given tool divided by the optimal wirelength on the indicated benchmark.



**Fig. 7.** Run time vs. numbers of cells on PEKO test cases.

mathematical problems in the gigascale range. However, multiscale methods for PDEs are not readily transferred to the large-scale combinatorial optimization problems common to VLSICAD. A lack of continuity presents one obstacle. The presence of myriad local extrema presents another. Nevertheless, in recent years, multiscale methods have also been successfully applied to VLSICAD [CS03]. In circuit partitioning, hMetis and MLpart [KAKS97, CAM00] produce the best cutsize minimization, and MLPR [CW02] produces the best balance of timing delay and cutsize. Significant progress has also been made in multiscale placement [SR99, CCK<sup>+</sup>03, SWY02, CCPY02, KW04, HMS04] and routing [CFZ01, CXZ02, CL04].

Hierarchical levels of abstraction are indispensable in the design of gigascale complex systems, but hierarchies must properly represent physical relationships, viz., interconnects, among constituent parts. The flexibility of the multiscale heuristic provides the opportunity both to merge previously distinct phases in the design flow and to simultaneously model very diverse, heterogeneous kinds of constraints.

### 1.7 An Operative Definition of Multiscale Optimization

The engineering complexity of VLSICAD has led researchers to a large variety of algorithms and terminology quite distinct from what exists in other areas. In this section, we take a fairly inclusive view and try not to impose artificial distinctions. By *multiscale optimization*, we mean

- (i) the use of optimization at every level of a hierarchy of problem formulations, wherein
- (ii) each variable at any given coarser level represents a subset of variables at the adjacent finer level.

In particular, each coarse-level formulation can be viewed directly as a coarse representation of the original problem. Therefore, coarse-level solutions implicitly provide approximate solutions at the finest level as well. While many long-standing paradigms in VLSICAD employ optimization hierarchically, multiscale algorithms as defined above are a relatively new phenomenon. This distinction is considered further in Section 3.2.

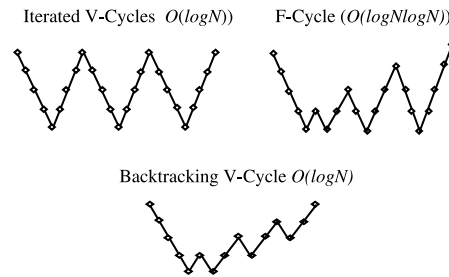
The terms *multilevel* and *multiscale* are used synonymously.

#### Characterization of Multiscale Algorithms

1. *Hierarchy Construction.* Although the construction is usually from the bottom up by recursive aggregation, top-down constructions are also possible, as described in Section 3.4 below.
2. *Relaxation.* In the combinatorial setting, the purpose of intralevel optimization is not generally viewed as error smoothing but rather the efficient, iterative exploration of the solution space at that level. Continuous, discrete, local, global, stochastic, and deterministic formulations may be used in various combinations.

3. *Interpolation.* A coarse-level solution can be transferred to and represented at its adjacent finer level in a variety of ways. The simplest and most common is simply the placement of all components of a cluster concentrically at the cluster's center. This choice amounts to a piecewise-constant interpolation.
4. *Iteration Flow.* The levels of the hierarchy may be traversed in different ways. A single pass from the coarsest to the finest level is still the most common. Alternatives include standard flows such as a single V-cycle, multiple V-cycles, W-cycles, and the full multigrid (FMG) F-cycle (see Figure 8 on Page 16).

The forms taken by these components are usually tightly coupled with the diverse objective models and constraint models used by different algorithms.



**Fig. 8.** Some iteration flows for multiscale optimization.

## 2 Multiscale Hypergraph Partitioning

Given a hypergraph  $H = \{V, E\}$ , the hypergraph partitioning problem is to partition  $V$  into disjoint subsets  $P = \{V_1, V_2, \dots, V_k\}$ , i.e.,

$$V_i \cap V_j = \phi \text{ for } i \neq j, \quad \text{and} \quad \cup V_i = V,$$

subject to certain objectives and constraints. Each block  $V_i$  of partition  $P$  is also called a partition. The area of partition  $V_i$  is defined as

$$\text{area}(V_i) = \sum_{v \in V_i} \text{area}(v)$$

The objective in hypergraph partitioning is usually to minimize the *cutsizes*, the number of hyperedges that connect vertices from two different partition blocks.

Typical constraints in the hypergraph partitioning problem include the following.

- The relative area of each partition block. E.g.,  $l_i \times \text{area}(V) \leq \text{area}(V_i) \leq u_i \times \text{area}(V)$  for user-specified parameters  $l_i$  and  $u_i$ .
- The number of partition blocks  $k$ . When  $k = 2$ , the problem is called *bipartitioning*. When  $k > 2$ , the problem is called *multiway partitioning*. Multiway partitioning problems are often reduced to a recursive sequence of bipartitioning problems.

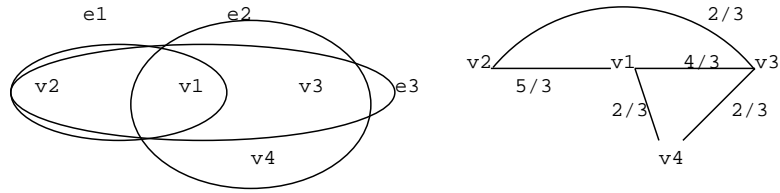
We focus on bipartitioning algorithms for the rest of this section.

## 2.1 Early Multiscale Hypergraph Partitioning: FMC

An early effort to accelerate the standard Fiduccia-Mattheyses Algorithm (FM, Figure 10 on Page 19) by recursive clustering was made without any deliberate connection to existing multiscale algorithms [CS93]. To facilitate coarsening, a graph corresponding to the original hypergraph is constructed. A bottom-up clustering algorithm is applied to recursively collapse small cliques into clusters. An iterative-refinement algorithm is then applied to the clustered hypergraph to optimize the cutsizes, satisfying the area constraint at the same time. This refinement is coupled with a recursive declustering process until the partitioning result on the original hypergraph is obtained. This early work is referred to as FMC (FM with clustering).

### 2.1.1 Hypergraph-to-graph transformation

Coarsening decisions in FMC are guided by the strength of connection between pairs of vertices. Because a hyperedge may connect more than 2 vertices, the hypergraph is first approximated by a graph. Following the so called *clique-model* approximation, each  $r$ -vertex hyperedge becomes a clique on the same  $r$  vertices in the graph. Since a hyperedge is now represented as a union of  $\binom{r}{2}$  edges, their weights should be properly scaled. Several weighting schemes have been proposed [CP68, Don88, HK72]. The one adopted in FMC assigns  $\frac{2}{r}$  to each edge of the clique, so that the total weight of the  $r$ -clique is the number of connections required to connect  $r$  vertices, namely,  $r - 1$ . When the same two vertices belong to several hyperedges, the edge-weight contributions from all of them are summed to determine the edge weight. For a pair of vertices  $v_1$  and  $v_2$ , the final edge weight between them is thus calculated as  $w(v_1, v_2) = \sum_{v_1 \in e, v_2 \in e} \frac{2}{|e|}$ . Figure 9 gives an example of the clique model approximation to a 4-vertex hypergraph. The edge weight between  $v_1$



**Fig. 9.** Transformation from a hypergraph to graph. Each hyperedge is transformed into a clique between all the vertices in the hyperedge.

and  $v_2$  is  $2/2+2/3 = 5/3$ , whose two components correspond to hyperedges  $e_1$  and  $e_3$ , respectively.

As most real circuits contain a small fraction of high-degree nets — these range in cardinality from 6 to a few hundred vertices — the actual size of the largest clique could easily reach several tens of thousands. Practical graph-based approximations therefore do not use cliques to model high-degree nets; stars are a popular alternative. However, large nets are empirically observed to have little impact on the result of partitioning. In FMC, all nets of degree more than 5 are simply ignored.

### 2.1.2 Coarsening

The coarsening scheme in FMC recursively collapses small cliques in a graph into clusters. The intuition comes from the theory of a random graph of  $n$  nodes and edge probability  $p$ . Let  $X_r$  be the expected number of  $r$ -cliques. Then for most  $n$ , there exists a threshold  $r_0$  such that  $X_{r_0} \gg 1$  and  $X_{r_0+1} < 1$ . The threshold  $r_0$  is calculated as

$$r_0 = 2 \log_b n - 2 \log_b \log_b n + 2 \log_b \frac{e}{2} + 1 + o(1),$$

where  $b = 1/p$ . In other words, the value of  $r_0$  is an approximation of the size of the largest clique in the graph. It is empirically observed that  $r_0$  is usually no more than 5 for typical hypergraphs found in VLSICAD.

Starting from the transformed graph, the coarsening searches for  $r_0$ -cliques and  $(r_0 + 1)$ -cliques for clustering. However, only cliques meeting the following criteria are accepted.

- *Area limit.* No cluster's area may exceed this fixed fraction of the total area of the original graph.
- *Size limit.* No cluster may contain more than this total number of vertices from the original graph.
- *Density threshold.* The density of a cluster with  $c$  nodes, defined as the sum of the weights of its edges divided by  $\binom{c}{2}$ , must equal or exceed a fixed fraction of the density of the whole graph,  $\alpha \times D$ , where  $D$  is the

Algorithm: FM refinement Initial partition generation Compute the cutsizes reduction of all vertex movements <b>While</b> there exists a movable vertex <b>do</b> find $v$ with the maximum reduction in cutsizes move $v$ to the destination partition and fix $v$ record the movement and the intermediate cutsizes update cutsizes the reduction of vertices connected with $v$ <b>End While</b> Find the minimum intermediate cutsizes $c_{min}$ Apply the sequence of moves which leads to $c_{min}$
---

**Fig. 10.** The FM partitioning algorithm for hypergraphs and graphs [FM82].

total edge weight of the graph divided by  $\binom{n}{2}$ ,  $n$  the total number of vertices in the graph. The parameter  $\alpha$  is empirically determined. The density threshold is imposed to ensure that the vertices in a cluster are strongly connected. It also prevents cliques introduced by high-cardinality hyperedges from automatically becoming clusters.

After each pass of clustering, the threshold  $r_0$  is recomputed for the clustered graph, and another pass is made with the new threshold on the clustered graph. The process terminates once the number of clusters produced is too small. A weighted matching is then applied to reduce the number of single unclustered vertices. Each qualifying pair is collapsed to a cluster. This step helps to balance the size of the clusters and further reduces the number of nodes in the clustered graph.

### 2.1.3 Initial Solution, Declustering, and Iterative Refinement

After the final clustered graph is obtained, partitioning is performed on it by FM (Figure 10). This refinement is iterated for several passes. Afterwards, the coarsened hypergraph is recursively declustered following the coarsening hierarchy. Because many of the clusters may be very large in area, the area-balance constraint is not strictly enforced at coarser levels. At each level of hierarchy, the refinement is repeated on the clusters at that level to improve area balance and further reduce cutsizes. This process of gradual declustering and refinement is repeated until the original hypergraph is restored, where further FM refinement is performed on the entire hypergraph.

### 2.1.4 Impact of Coarsening on the FM algorithm

The final partitioning results for FMC are compared with those for pure FM. The characteristics of the hypergraphs used for comparison are given in Table 2. On average, the coarsening step helps to reduce the best cut size by

hypergraph	#vertices	#hyperedges	Best Cut		Avg. Cut	
			FM	FMC	FM	FMC
8870	502	494	17	15	27.4	17.3
bm1	882	902	65	53	82.9	69.6
PrimGA1	733	902	48	49	69.2	64.7
PrimSC1	733	902	46	45	74	56.2
5655	921	760	54	48	67.9	62.6
Test04	1515	1658	44	44	46	48.4
Test03	1607	1618	93	64	137.9	80.1
Test02	1663	1721	121	80	181.1	105.9
Test06	1752	1674	60	64	79.9	74.8
Test05	2595	2751	42	42	61.3	57.5
19ks	2844	3282	151	129	173.7	156.1
PrimGA2	3014	3029	246	130	284	181.3
PrimSC2	3014	3029	199	130	277.2	229.1
industry2	12142	12949	458	315	812.6	402.3
Avg.			16.60%		21.20%	

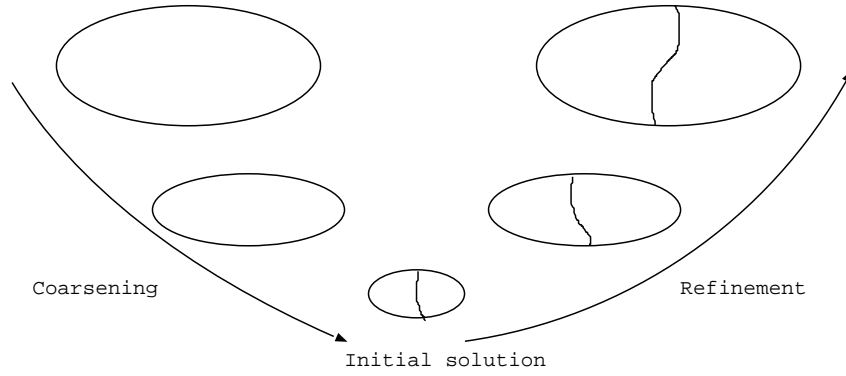
**Table 2.** Impact of coarsening on FM algorithm. On average, coarsening helps to reduce the best cut size by 16.6%. It reduces the average cut size by 21.2% [CS93].

16.6%. It reduces the average cut size by 21.2%. The impact of coarsening on the refinement stage is obvious in this scenario. The results are very promising; however, they do not demonstrate the full power of multiscale methods. There are several limitations in this work. First, the coarsening scheme only collapses cliques. This artificially imposed constraint will eliminate many possible coarsening candidates which might lead to better final cutsize. Second, its control on the cluster size is limited. Since a whole clique is collapsed each time, the coarsened graph tends to become more uneven in cluster size.

To our knowledge, FMC is the first application of multiscale optimization to circuit partitioning. Following FMC, there have been several efforts to apply multiscale methods to the hypergraph partitioning problem [AHK97, KAKS97, CLW99, CL00, CLW00, CAM00]. Among the most successful are hMetis [KAKS97] and MLpart [CAM00], described next.

## 2.2 hMetis

hMetis [KAKS97, Kar99, Kar02] was first proposed for bipartitioning and later extended to multiway partitioning. Its iteration flow is a sequence of the traditional multiscale V-cycles shown in Figure 11. It constructs a hierarchy of partitioning problems by recursive clustering, generates multiple initial-solution candidates, then recursively declusters and refines at each level. It improves its candidate solutions at each level of declustering by FM-based local relaxation (Figure 10). A certain fraction of the poorest solution candidates is discarded after relaxation so as not to incur excessive run-time at



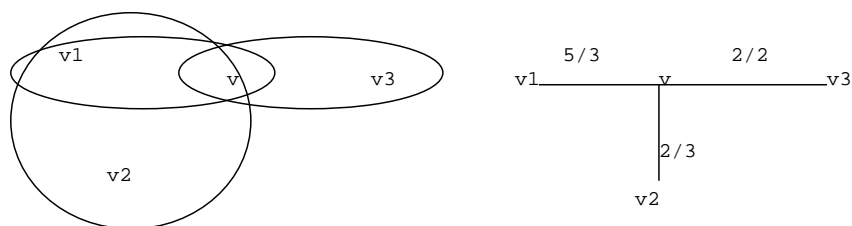
**Fig. 11.** V-cycle of hMetis. It consists of a coarsening phase, an initial partition generation, and a refinement phase.

the adjacent finer level. Each V-cycle after the first improves on the result of its predecessor by restricting its coarsening to preserve the initial given partition.

### 2.2.1 First-Choice Coarsening

In First-Choice Clustering, each vertex is associated with one of its most closely-connected neighbors, irrespective of whether that neighbor has already been associated with some other vertex. The function  $\varphi : V \times V \rightarrow [0, \infty)$  defining how closely two vertices are connected is called the vertex *affinity*. It defines a weighted *affinity graph* over the vertices of the hypergraph, as follows. Two vertices  $u$  and  $v$  are connected in the affinity graph if and only if they both belong to some common hyperedge, in which case the weight of the edge joining them is  $\varphi(u, v)$ . Given  $\varphi$ , First-Choice Clustering proceeds as follows. For each vertex  $v \in V$ , its incident edge of highest weight in the affinity graph is marked. Once every vertex has had its incident edge of highest weight marked, the marked edges define a new, highly disconnected subgraph whose connected components define clusters at the next coarser level. Hyperedges at the next coarser level are obtained by directly transferring the hyperedges at the current level. I.e., if the vertices in the hyperedge  $e_i = \{v_{i_1}, v_{i_2}, \dots, v_{i_j}\}$  are assigned to the clusters  $\bar{v}_{i_1}, \bar{v}_{i_2}, \dots, \bar{v}_{i_k}$ , then net  $e_i$  becomes net  $\bar{e}_i = \{\bar{v}_{i_1}, \bar{v}_{i_2}, \dots, \bar{v}_{i_k}\}$  at the coarser level. Further discussion of hyperedge coarsening appears in Section 3.3 below.

Figure 12 gives an example of this scheme. Vertex  $v$  has connections with  $v_1, v_2, v_3$  in the transformed graph. Among them,  $v_1$  has the strongest connection. Therefore,  $v$  will be clustered with  $v_1$ .



**Fig. 12.** First-Choice Clustering. Vertex  $v$  will be clustered with vertex  $v_1$  since the edge between  $v$  and  $v_1$  has the highest weight among all the edges incident to  $v$ .

### 2.2.2 Initial partition generation

At the coarsest level, where the number of clusters is small compared with the original hypergraph, hMetis generates a pool of candidate solutions and refines all of them. In the end, the ten best initial solutions are kept and propagated to the adjacent finer level. At each level, all candidate solutions are improved by iterative refinement, and some of the worst candidates are discarded prior to interpolation to the next level, so that the total run time is not compromised. The pool of candidates is thus gradually pared down to a single best solution at the finest level.

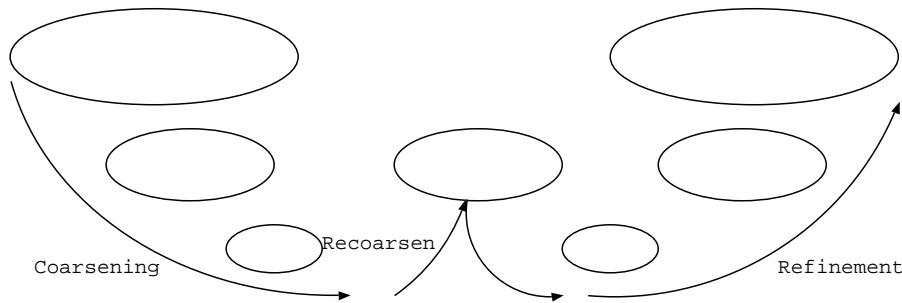
Since many clusters are relatively large in area, moving them across the partitioning boundary will often result in an area-constraint violation. On the other hand, forbidding such moves may lead the partitioning solution to a local minimum. As a compromise, hMetis allows intermediate solutions to violate the area constraints, as does FMC. However, it saves those solutions that satisfy the area constraints in the end.

### 2.2.3 Iterative Refinement

Two algorithms are used during the refinement at each level. The first is based on FM with early termination. The vertices are visited in random order. Each vertex is moved across the cutline if the move reduces cutsize without violating the area constraints. As soon as  $k$  vertices are moved without improvement in cutsize, the refinement stops. The maximum number of passes is limited to two. The second algorithm, Hyperedge Refinement (HER), simultaneously moves groups of vertices in the same hyperedge across the partitioning boundary so that this hyperedge is no longer cut. Compared with FM, HER targets hyperedges rather than individual vertices to reduce the cutsize.

### 2.2.4 Iterative V-cycle

After the first V-cycle, hMetis applies additional V-cycles to further refine the cutsize until no appreciable improvement can be obtained. In the coarsening



**Fig. 13.** Iterative flow of hMetis. The recoarsening does not have to start from the finest level. It can start in the middle of the V-cycle as well.

phase of each subsequent V-cycle, the partitioning result from the preceding V-cycle is preserved — only vertices that belong to the same partition block will be considered for clustering. The partitioning  $P_{i+1}$  of the next level coarser hypergraph  $H_{i+1}$  is derived directly from the partitioning result  $P_i$  from  $H_i$ . The coarsening thus obtained differs from that used in the preceding V-cycle and often leads to improved results in the subsequent interpolation and refinement at each level.

Due to complexity considerations, the recoarsening does not necessarily start from the finest level. It can start in the middle of the V-cycle instead, as in the standard W-cycle flow for numerical PDE solvers [BHM00]. Figure 13 gives such an example. Instead of letting the V-cycle finish and starting another, the recoarsening can start from a level between the coarsest and finest, and go through the following refinement stage, as in a normal V-cycle.

### 2.2.5 Comparison of hMetis with other Partitioning Algorithms

The improvement of hMetis in run time and quality over the previous state of the art is dramatic. Numerical experiments compare hMetis to CDIP-LA3 and CLIP-PROP [DD97], PARABOLI [RDJ94], GFM [LLC95], GMetis [AK96]. Hypergraph characteristics for the test cases and the corresponding results for these algorithms are given in Table 3. Due to complexity issues, some algorithms do not produce reasonable output for some hypergraphs. Overall, the cutsize produced by hMetis is 4% to 23% less than that produced by previous partitioning algorithms. The speed and scalability of hMetis is even more impressive, as its run time (not shown in Table 3) is one to two orders of magnitude less than the competition's.

### 2.3 MLpart

MLpart [CAM00] is another successful multiscale hypergraph partitioner. As with hMetis, it has a recursive coarsening phase, generation of an initial solu-

hypergraph	$ V $	$ E $	CDIP- LA3	CLIP- PROP	PARA- BOLI	GFM	GMetis	hMetis
s15850	10470	10383	59	56	91	63	53	42
industry2	12637	13419	182	192	193	211	177	168
industry3	15406	21923	243	243	267	241	243	241
s35932	18148	17828	73	42	62	41	57	42
s38584	20995	20717	47	51	55	47	53	48
avq.small	21918	22124	139	144	224	na	144	127
s38417	23849	23843	74	65	49	81	69	50
avq.large	25178	25384	137	143	139	na	145	127
golem3	103048	144949	na	na	1629	na	2111	1424

**Table 3.** Comparison of hMetis with other state-of-the-art partitioning algorithms in 1997. For each given hypergraph,  $|V|$  denotes the number of vertices, and  $|E|$  denotes the number of hyperedges. Overall, the cutsizes produced by hMetis is 4% to 23% better than that by previous partitioning algorithms [KAKS97].

tion, and interleaved interpolation and iterative refinement from the coarsest level back to the finest level.

### 2.3.1 Coarsening

MLpart has several distinct features in its coarsening scheme. One is that the netlist is continuously updated as the coarsening progresses. When a pair of vertices is chosen for clustering, the clustering effect is immediately visible to all the remaining unclustered vertices. The vertex-to-vertex connection-weight calculation of MLpart is also the sum-of-weights contribution by the hyperedges connecting a pair of vertices. However, in MLpart, the weight contribution of 2-vertex hyperedges is set to 2, whereas the contribution of hyperedges with more than 2 vertices is set to 1. The intuition is that the weight should correspond to the potential reduction of pins from the hypergraph. (When the vertices of a 2-pin net are clustered together, that net becomes a singleton at all coarser levels and can thus be removed at those levels.) To discourage the formation of large clusters, the weight is divided by the sum of the cluster areas. An area-balance limit set to  $4.5\times$  the average cluster size on each clustering level is imposed, so that clusters with area beyond the limit will not be formed. Clustering stops when the total number of clusters drops below 200.

### 2.3.2 Initial Partition Generation

The initial partitioning at the coarsest level is randomly generated by assigning clusters in decreasing order of area with biased probability. Before all the partition blocks reach a minimum area threshold, the probability of a cluster being assigned to a particular partition block is proportional to the area slack of that partition *after* the cluster is assigned. After the threshold is reached,

the probability is proportional to the maximum allowed area of each cluster. Following the random initial generation, FM-based refinement is applied to the initial solution. Similar to hMetis, multiple initial solutions are generated, but only the best solution is kept and propagated to finer levels.

### 2.3.3 Iterative refinement

The refinement of MLpart is also based on FM. The authors also note that strictly satisfying the area constraint at coarser levels does not lead to the smallest possible cutsize at the finest level. Therefore, the acceptance criteria of a cluster movement is relaxed so that a move is accepted as long as it does not increase the violation of balance constraints. It is empirically observed that, although the area-balance criterion is relaxed, the refinement can usually reach a legal solution with smaller cutsize after the first pass.

### 2.3.4 Comparison with hMetis

hypergraph	#vertices	#hyperedge	hMetis	MLPart
ibm01	12506	14111	255	238
ibm02	19342	19584	320	348
ibm03	22853	27401	779	802
ibm04	27220	31970	506	535
ibm05	28146	28446	1740	1726
ibm06	32332	34826	374	394
ibm07	45639	48117	809	790
ibm08	51023	50513	1166	1195
ibm09	53110	60902	540	560
ibm10	68685	75196	779	938
ibm11	70152	81454	744	779
ibm12	70439	77240	2326	2349
ibm13	83709	99666	1045	1095
ibm14	147088	152777	1894	1759
ibm15	161187	186608	2032	2029
ibm16	182980	190048	1721	1714
ibm17	184752	189581	2417	2316
ibm18	210341	201920	1632	1666

**Table 4.** Comparison of MLpart and hMetis. The two partitioners produce comparable results on 18 hypergraphs extracted from real circuits [CAM00].

Table 4 gives the comparison between MLPart and hMetis. Overall, MLpart produces comparable cutsize compared with hMetis on 18 hypergraphs extracted from real circuits [Alp98]. The average relative cutsize difference is within 1%. Run times reported by the authors are also comparable. A recent

optimality study of partitioning algorithms, however, suggests that MLpart may be considerably more robust than hMetis. On a certain class of synthetic bipartitioning benchmarks with known upper bounds on optimal cutsize, MLpart consistently obtains cuts attaining the upper bounds, while hMetis finds cuts up to 20% worse [JCX03, CCRX04].

### 3 Multiscale Placement

The spatial arrangement of circuit elements fundamentally constrains the layout of the circuit's interconnect and, therefore, the signal timing of the integrated circuit. Thus, as interconnect delay continues to increase relative to device delay, the importance of a good placement also increases. Rapid progress in placement has been made in the last few years independently across several different families of algorithms, with order-of-magnitude improvements in run-time and quality. Significant challenges remain, however, particularly with respect to the increasing size, complexity, and heterogeneity of integrated circuits and the constraint models associated with their design. Recent estimates of the gap between optimal and attainable placements (Section 1.5) strongly suggest that design improvements alone may produce the equivalent of at least a full technology generation's worth of improved performance [CCX03b].

#### 3.1 Problem Description

As described in Section 1.5, the *placement* problem is to assign coordinates  $(x_i, y_i) \in R$  to the  $v_i \in V$ . The wirelength-driven problem (2) is, however, only a generic representative of the true objectives and constraints, which include the following.

1. *Overlap.* No two cells may overlap. Expressed literally, this condition amounts to  $N(N - 1)/2$  nonsmooth, nonconvex constraints for the placement of  $N$  cells.
2. *Wirelength.* The estimated total length of all wires forming these connections must be either as small as possible (an objective) or below a prescribed limit (a constraint). Prior to routing (Section 4), estimates of the wirelength must be used.
3. *Timing Delay.* The maximum propagation time of a signal along any path in the IC should be either as small as possible (an objective) or below a specified limit (a constraint).
4. *Routing Congestion.* The estimated density of wires necessary to route the nets for the placed cells must not be so large that the spacing between adjacent wires falls below the limits of manufacturability.
5. *Additional Considerations.* Other conditions may involve total power consumption, maximum temperature, inductance, noise, and other complex manufacturability criteria.

Multiple mathematical formulations exist both for the data functions representing these conditions and the manner in which they are combined to formulate the placement problem. In practice, different formulations are often employed at different stages of design. For simplicity, the discussion in this section is limited primarily to wirelength-driven placement, however, it must be noted that the utility of any particular approach rests largely in its adaptability to more specific formulations which emphasize some subset of the above conditions over the others, e.g., timing-driven placement, routing-congestion-driven placement, temperature-driven placement, etc. A good placement supports a good routing, and ultimately, the quality of a placement can only be accurately judged after routing.

### 3.2 An Operative Definition of Multiscale Placement

A precise, universally accepted definition of multiscale placement has yet to emerge. Many long-standing [Bre77, QB79, SS95, Vyg97, KSJA91] and more recent [CKM00, YM01, WYS00b, BR03, CCK<sup>+</sup>03, CCPY02, KW04] algorithms for placement employ optimization at every level of a recursively constructed hierarchy of circuit models. None of these, however, includes all the core elements of the most successful multiscale solvers for PDEs [TOS00, Bra77, BHM00]. For example, no leading placement tool uses relaxation during recursive coarsening as a means of either improving the choice of coarse-level variables or reducing the error in the coarse-level solution. It is not yet clear whether active research will bring placement algorithms closer to the “standard” multiscale metaheuristics [Bra86, Bra01, BR02].

The coverage here follows the operative definition of multiscale optimization given in Section 1.7. In particular, each variable at any given coarser level must represent a subset of variables at the adjacent finer level. This requirement distinguishes multiscale algorithms from so called “hierarchical” methods, as explained below.

#### Comparison with Traditional Hierarchical Approaches

Within the placement community, the term “hierarchical” is generally used as a synonym for top-down recursive-partitioning based approaches, in which spatial constraints on cell movement are added recursively by top-down subregion-and-subnetlist refinement, but the variables at every step remain essentially identical to the variables for the problem as initially given. Initially, cells are partitioned into two almost-equal-area subsets such that the total weight of all nets containing cells in both subsets (the “cutsizes”) is minimized (Section 2). The placement region is divided by a straight-line cut into two subregions, and each of the cell subsets is assigned to one of these subregions. Connections to fixed input-output (I/O) pads, if present, can be used as a guide in selecting the cut orientation and position as well as the cell-subset-to-placement-subregion assignment. The cutsizes-driven area

bipartitioning continues recursively on the subregions until these are small enough that simple discrete branch-and-bound search heuristics can be efficiently used. At each step of partitioning, connections between blocks are modeled by *terminal propagation*. When the cells in block  $B_i$  are to be bipartitioned, cells in external blocks  $B_j$  are modeled as fixed points along the boundary of  $B_i$ , when they belong to nets also containing cells in  $B_i$ . That is, a net containing cells in  $B_i$  and cells in  $B_j$  is modeled simply as a net containing the same movable cells in  $B_i$  and, for each cell in  $B_j$ , a fixed point on the boundary of  $B_i$ .

An important variation on the top-down recursive-bisection based framework combines the partitioning with analytical minimization of smoothed wirelength in subtle ways. In this system, an initial layout is calculated by unconstrained minimization, typically of a weighted quadratic wirelength model, without regard to overlap.<sup>(3)</sup> This initial solution tends to knot cells together in an extremely dense subregion near the center of the placement region, but the presence of I/O pads along the chip boundary gives the cells a nontrivial relative ordering. This ordering then guides the subsequent recursive partitioning of cells toward a final layout. In Gordian [KSJA91], cells are initially partitioned consistently with the given relative ordering. Cutsizes-driven iterative improvement of the partitioning is then used not to displace cells but rather to define a recursive sequence of center-of-mass positions for the cell subsets calculated by the partitionings. These center-of-mass positions form hierarchical grids and are iteratively added as equality constraints to the global, weighted quadratic optimization. Eventually, the accumulation of these constraints leads to a relatively uniform cell-area distribution achieved by global wirelength minimization at each step. In BonnPlace [KSJA91, Vyg97, BR03], cell displacement from the analytically determined layout is minimized rather than cutsizes. No center-of-mass or other constraints are explicitly used. Instead, the wirelength objective is directly modified to incorporate the partitioning results; cell displacements to their assigned subregions are minimized at subsequent iterations.

The multiscale framework departs significantly from the traditional top-down hierarchical placement techniques. The key distinction is that, in the multiscale setting, optimization at coarser levels is performed on aggregates of cells, while in the traditional formulation, optimization at coarser levels is still done on the individual cells. That is, the traditional flow employs a hierarchy of constraints but not a hierarchy of variables.

The distinction is blurred somewhat by the fact that state-of-the-art top-down placement algorithms based on recursive cutsizes-driven partitioning *all* employ multiscale partitioning at most if not all levels. It is not yet precisely

---

<sup>(3)</sup> The quadratic objective is preferred principally to support the speed and stability of the numerical solvers used. It can be iteratively reweighted to better approximate half-perimeter wirelength [SDJ91].

understood to what extent this use of multiscale partitioning at every level of the top-down flow matches or exceeds the capabilities of multiscale placement.

### Classification of Multiscale Placement Algorithms

Our survey of leading multiscale placement algorithms is organized as follows. Algorithms constructing their problem hierarchies from the bottom up by recursive aggregation are described in Section 3.3. Those constructing their problem hierarchies from top-down by recursive partitioning are described in Section 3.4. Each overview of each method follows the organization outlined in Section 1.7: (i) hierarchy construction, (ii) intralevel relaxation, (iii) interpolation, and (iv) multiscale iteration flow.

### 3.3 Clustering-based Methods

Among the known leading methods, clustering-based algorithms are perhaps the closest to traditional multiscale methods in scientific computation. Local connections among vertices at one level are used to define vertex clusters in various ways, some hypergraph-based, others graph-based. Assuming each vertex is assigned to some nontrivial cluster of vertices, the number of clusters is at most half the number of vertices. Recursively clustering clusters and transferring nets produces the requisite multiscale problem formulation. For clarity, we speak of *vertices* at one level and *clusters* of vertices at its adjacent, coarser level.

Although various alternatives exist for defining the vertex clusters, connectivity among the clusters at a given level is generally defined by directly transferring nets of vertices at the adjacent finer level to nets of clusters, as described in Section 2.2.1. If the vertices in the hyperedge  $e_i = \{v_{i_1}, v_{i_2}, \dots, v_{i_j}\}$  are assigned to the clusters  $\bar{v}_{i_1}, \bar{v}_{i_2}, \dots, \bar{v}_{i_k}$ , then net  $e_i$  becomes net  $\bar{e}_i = \{\bar{v}_{i_1}, \bar{v}_{i_2}, \dots, \bar{v}_{i_k}\}$  at the coarser level. In this process, nets may be eliminated in two ways. First, because tightly interconnected vertices at the finer level are often clustered together at the coarser level, many low-degree nets at the finer level become singleton nets at the coarser level, where they are then deleted. Second, two or more distinct nets at the finer level may become identical at the coarser level, where they can be merged.

Hyperedge degrees decrease or remain constant as hyperedges are propagated toward coarser levels, because  $k \leq j$ . However, the nets eliminated during coarsening are mainly of low degree. Thus, the average degree of nets at coarser levels is typically about the same as that at finer levels, with some decrease in the maximum net degree. Vertex degree, however, increases dramatically with coarsening. Under first-choice clustering on one standard benchmark (IBM/ISPD98 #18), the average number of nets to which each vertex belongs jumps from 3 at the finest level to 458 at the coarsest level.

Alternatively, the number of nets increases relative to the number of vertices significantly at coarser levels. In this sense, the hypergraph model at the coarsest level is quite different from the one at the finest level.

The accuracy of hypergraph coarsening remains, as far as we know, a heuristic and poorly understood notion. There is no generally accepted means of comparing two different clustering rules other than to attempt them both for the problem at hand, e.g., partitioning, placement, or routing, and observe whether one rule consistently produces better final outcomes. Leading methods generally attempt to merge vertices in a way that eliminates as many hyperedges at the coarser level as possible. A recent study [HMS03a, HMS04] explores vertex clustering on hypergraphs in some detail for the purpose of accelerating a leading top-down partitioning-based placer (Capo [CKM00]) by applying just one level of clustering.

### 3.3.1 Ultrafast VPR

Though limited use of clustering appears earlier in the placement literature [SS95, HL99], to our knowledge, Ultrafast VPR [SR99] is the first published work to recursively cluster a circuit model into a hierarchy of models for placement by multiscale optimization. Ultrafast VPR is used to accelerate the annealing-based VPR algorithm (“Versatile Packing, Placement and Routing” [BR97]) in order to reduce design times on field-programmable gate arrays (FPGAs) at some expense in placement quality. (FPGA placement quality in Ultrafast VPR is measured by the area used.) Ultrafast VPR is not seen as a general means of improving final placement quality over VPR, when both approaches are allowed to run to normal termination. The authors observe, however, that when the run-time for the two algorithms is explicitly limited to about 10 seconds, the multiscale approach does produce superior results.

#### Hierarchy construction

Designed for FPGAs, Ultrafast VPR exploits their regular geometry. It creates only uniform, square clusters; i.e., the number of vertices per cluster at each level is held fixed at a perfect square: 4, 9, 16, etc. To form a cluster, a seed vertex is randomly selected and denoted  $c$ . Each vertex  $b$  connected to the cluster is then ranked for merging with  $c$  by the value

$$w_b = A_{bc} + \sum_{\{e \in E \mid b, c \in e\}} \frac{1}{|e| - 1},$$

where  $A_{bc}$  denotes the number of nets containing both  $b$  and  $c$  that will be eliminated if  $b$  is merged with  $c$ . The terms in the sum indicate that a low-degree net containing vertices  $b$  and  $c$  connects them more tightly than a high-degree net. An efficient bucketing data structure supports fast updates of the rankings as vertices are added to the cluster.

## Relaxation

Intralevel improvement in Ultrafast VPR proceeds in two stages: constructive placement followed by adaptive simulated annealing.

In the constructive phase, clusters are considered in order of their connectivity with I/O pads. I.e., first the I/O pads themselves are placed, then clusters connected to output pads are placed, then clusters connected to input pads are placed, then clusters connected to already placed clusters, and so on. Each cluster at a given level is placed as close as possible to its “optimal location,” determined as the arithmetic mean of the already placed clusters with which it shares connections. The initial I/O pad placement at the coarsest level is random; at subsequent, finer levels, pad positions are interpolated and improved in the same way that other clusters are. At the coarsest level, only connections to already placed clusters are considered in the weighted average used to compute a given cluster’s ideal position. At all other levels, the mean of *all* clusters connected to the given cluster being placed is used, where clusters yet to be placed at the current level are temporarily given the position of their parent clusters. The solution to the constructive placement defines an initial configuration for subsequent fast annealing-based relaxation.

High temperature annealing is used at coarser levels, low temperature annealing at finer levels. In both VPR and Ultrafast VPR, the temperature  $T$  is decreased by an adaptively selected factor  $\alpha \in [0.5, 0.95]$  determined by the number of moves accepted at the preceding temperature,  $T \leftarrow \alpha T$ . In Ultrafast VPR, however,  $\alpha$  is squared to accelerate the search. The number of moves per temperature is set to  $n_{\text{moves}} = cN^{4/3}$ , where  $N$  is the number of movable objects, and the constant  $c \in [0.01, 10]$  may also be decreased by the factor  $\alpha$ .

The starting temperature  $T_0^{(\ell)}$ , number of moves  $n_{\text{moves}}$ , stopping temperature  $T_f^{(\ell)}$ , and decrease factor  $\alpha$  at each level are the key parameters characterizing the *annealing schedule* of Ultrafast VPR. The authors consider three different annealing schedules: (i) an aggressive, adaptive schedule in which the parameters are dynamically updated; (ii) a greedy “quench” in which no uphill (wirelength-increasing) moves are permitted; (iii) a fixed choice of  $T_0^{(\ell)}$ ,  $T_f^{(\ell)}$ , and  $\alpha$ . The quality/run-time trade-offs of these schedules are reported in experiments for Ultrafast VPR. Overall, schedule (ii) performs best for the shortest allowed run-times (1 second or less), schedule (iii) performs best for intermediate run-times of 1–100 seconds, and schedules (i) and (iii) perform comparably after 100 seconds, on average. In general, the average results over 20 circuits are not very sensitive to the choice of annealing schedule.

During the annealing, component clusters are allowed to migrate across the boundaries of their parent clusters; the reported experiments indicate that this freedom of movement significantly improves final placement quality, allowing finer-level moves to essentially correct errors in clustering not visible at coarser levels.

### Interpolation

Initially, each vertex inherits the position of its parent. That is, the components of each cluster are simply placed concentrically at the cluster's center.

### Iteration Flow

Only one pass from the coarsest to the finest level is used. The burden of iterative improvement is placed entirely on the annealing process; multiple V-cycles are not attempted or even mentioned. There are no citations in the original report on Ultrafast VPR to any other work on multiscale algorithms. It appears the authors arrived at their algorithm without any awareness of existing multiscale techniques used in other areas.

#### 3.3.2 mPL

mPL [CCKS00, CCKS03, CCK<sup>+</sup>03] is the first clustering-based multiscale placement algorithm for standard-cell circuits. It evolved from an effort to apply recent advances in numerical algorithms for nonlinear programming and particle systems to circuit placement. The initial goal was the development of a scalable nonlinear-programming formulation, possibly making use of multiscale preconditioning for large Newton-based linear systems of equations. Experiments showed that for problem sizes above roughly 1000, steplength restrictions slowed progress to a crawl, as the intricacy of the  $\mathcal{O}(N^2)$  constraints limited the utility of pointwise approximations to tiny neighborhoods of the evaluation points. Multiscale optimization was seen as a means of escaping the complexity trap. Early numerical experiments demonstrated superb scalability at some loss in quality. Subsequent improvements have brought mPL's quality and scalability to a level comparable to the best available academic tools.

### Hierarchy construction

mPL uses first-choice clustering (FC, Section 2.2.1). The mPL-FC affinity that vertex  $i$  has for vertex  $j$  is

$$r_{ij} = \sum_{\{e \in E \mid i, j \in e\}} \frac{w(e)}{(|e| - 1)\text{area}(e)}, \quad (3)$$

where  $w(e)$  is the weight assigned to hyperedge (net)  $e$ ,  $\text{area}(e)$  denotes the sum of the areas of the vertices in  $e$ , and  $|e|$  denotes the number of vertices in hyperedge  $e$ . Dividing by the net degree promotes the elimination of small hyperedges at coarser levels, making the coarse-level hypergraph netlists sparser and hence easier to place [Kar99, SR99, HMS03b]. The area factor in the denominator gives greater affinity to smaller cells and thus promotes a more

uniform distribution of areas at coarser levels; this property supports the nonlinear programming and slot assignment modules discussed below. For each vertex  $i$  at the finer level, the vertex  $j$  assigned to it for clustering is not necessarily of maximal mPL-FC affinity but is instead of least hyperedge degree among those vertices within 10% of  $i$ 's maximum FC affinity. When this choice is not unique, a least-area vertex is selected from the least-degree candidates. Hyperedges are transferred from vertices to clusters in the usual fashion, as described at the beginning of Section 3.3.

Initially, edge-separability clustering (ESC) [CL00] was used in mPL to define clusters based on fast, global min-cut estimates. The first-choice strategy improves overall quality of results by about 3% over ESC [CCK<sup>+</sup>03].

### Relaxation

No relaxation is used in the recursive coarsening; the initial cluster hierarchy is determined completely by netlist connectivity. Nonlinear programming (NLP) is used at the coarsest level to obtain an initial solution, and local refinements on subsets are used at all other levels. Immediately after nonlinear programming at the coarsest level or interpolation to finer levels, the area distribution of the placement is evened out by recursive bisection and linear assignment. Subsequent subset relaxation is accompanied by area-congestion control; these area-control steps ultimately enable by local perturbations the complete removal of all cell overlap at the finest level prior to detailed placement.

By default, clustering stops when the number of clusters reaches 500 or fewer. At the coarsest level, vertices  $v_i$  and  $v_j$  are modeled as disks, and their pairwise nonoverlap constraint  $c_{ij}(X, Y)$  is directly expressed in terms of their radii  $\rho_i$  and  $\rho_j$ :

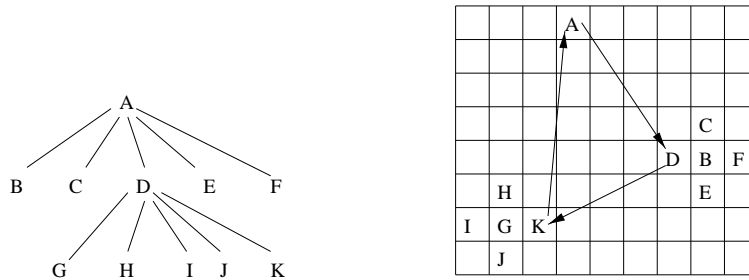
$$c_{ij}(X, Y) = (x_i - x_j)^2 + (y_i - y_j)^2 - (\rho_i + \rho_j) \geq 0 \quad \text{for all } i < j.$$

Quadratic wirelength is minimized subject to the pairwise nonoverlap constraints by a customized interior-point method with a slack variable added to the objective and the nonoverlap constraints to gradually remove overlap. Interestingly, experiments suggest that area variations among the disks can be ignored without loss in solution quality. That is, the radius of each disk can be set to the average over all the disks:  $\rho_i = \rho_j = \bar{\rho} = (1/N) \sum_1^N \rho_k$ . After nonlinear programming, larger-than-average cells are chopped into average-size fragments, and an overlap-free configuration is obtained by linear assignment on the cells and cell fragments. Fragments of the same cell are then reunited, the area overflow incurred being removed by ripple-move cell propagation described below. Discrete Goto-based swaps are then employed as described below to further reduce wirelength prior to interpolation to the next level. Relaxation at each level therefore starts from a reasonably uniform area-density distribution of vertices.

A uniform bin grid is used to monitor the area-density distribution. The first four versions of mPL rely on two sweeps of relaxations on local subsets at all levels except the coarsest. These local-subset relaxations are described in the next two paragraphs.

The first of these sweeps allows vertices to move continuously and is called quadratic relaxation on subsets (QRS). It orders the vertices by a simple depth-first search (DFS) on the netlist and selects movable vertices from the DFS ordering in small batches, one batch at a time. For each batch the quadratic wirelength of all nets containing at least one of the movable vertices is minimized, and the vertices in the batch are relocated. Typically, the relocation introduces additional area congestion. In order to maintain a consistent area-density distribution, a “ripple-move” algorithm [HL00] is applied to any overfull bins after QRS on each batch. Ripple-move computes a maximum-gain monotone path of vertex swaps along a chain of bins leading from an overfull bin to an underfull bin. Keeping the QRS batches small facilitates the area-congestion control; the batch size is set to three in the reported experiments.

After the entire sweep of QRS+ripple-move, a second sweep of Goto-style permutations [Got81] further improves the wirelength. In this scheme, vertices are visited one at a time in netlist order. Each vertex’s optimal “Goto” location is computed by holding all its vertex neighbors fixed and minimizing the sum of the bounding-box lengths of all nets containing it. If that location is occupied by  $b$ , say, then  $b$ ’s optimal Goto location is similarly computed along with the optimal Goto locations of all of  $b$ ’s nearest neighbors. The computations are repeated at each of these target locations and their nearest neighbors up to a predetermined limit (3–5). Chains of swaps are examined by moving  $a$  to some location in the Manhattan unit-disk centered at  $b$ , and moving the vertex at that location to some location in the Manhattan unit disk centered at its Goto location, and so on. The last vertex in the chain is then forced into  $a$ ’s original location. If the best such chain of swaps reduces wirelength, it is accepted; otherwise, the search begins anew at another vertex. See Figure 14.



**Fig. 14.** Goto-based discrete relaxation in mPL.

In the most recent implementation of mPL [CCS05], global relaxations, in which all movable objects are simultaneously displaced, have been scalably incorporated at every level of hierarchy. The redistribution of smoothed area density is formulated as a Helmholtz equation subject to Neumann boundary conditions, the bins defining area-density constraints serving as a discretization. A log-sum-exp smoothing of half-perimeter wirelength defined in Section 3.4.2 below is the objective. Given an initial unconstrained solution at the coarsest level or an interpolated solution at finer levels, an Uzawa method is used to iteratively improve the configuration.

### Interpolation

AMG-based weighted aggregation [BHM00], in which each vertex may be fractionally assigned to several generalized aggregates rather than to just one cluster, has yet to be successfully applied in the hypergraph context. The obstacle is that it is not known how to transfer the finer-level hyperedges, objectives, and constraints accurately to the coarser level in this case. AMG-based weighted *disaggregation* is simpler, however, it has been successfully applied to placement in mPL.

For each cluster at the coarser level, a C-point representative is selected from it as the vertex largest in area among those of maximal weighted hyperedge degree. C-points simply inherit their parent clusters' positions and serve as fixed anchors. The remaining vertices, called F-points, are ordered by non-increasing weighted hyperedge degree and placed at the weighted average of their strong C-point neighbors and strong, already-placed F-point neighbors. This F-point repositioning is iterated a few times, but the C-points are held fixed all the while.

### Iteration Flow

Two backtracking V-cycles are used (Figure 8). The first follows the connectivity-based FC clustering hierarchy described above. The second follows a similar FC-cluster hierarchy in which both connectivity and proximity are used to calculate vertex affinities:

$$r_{ij} = \sum_{\{e \in E \mid i, j \in e\}} \frac{w(e)}{(|e| - 1)\text{area}(e) \|(x_i, y_i) - (x_j, y_j)\|}.$$

During this second aggregation, positions are preserved by placing clusters at the weighted average of their component vertices' positions. No nonlinear programming is used in the second cycle, because it alters the initial placement too much and degrades the final result.

### 3.3.3 mPG

As described in Section 4, the wires implementing netlist connections are placed not only in the same region containing circuit cells, but also in a set of 3–12 routing layers directly above the placement region. The *bottom* layers closest to the cells are used for the shortest connections. The top layers are used for global connections. These can be made faster by increasing wire widths and wire spacing. As circuit sizes continue to increase, so do both the number of layers and the competition for wiring paths at the top layers. Over half the wires in a recent microprocessor design are over 0.5 mm in length, while only 4.1% are below 0.5 mm in length [CCPY02].

While many simple, statistical methods for estimating routing congestion during placement exist (*topology-free* congestion estimation [LTKS02]) it is generally believed that, for a placement algorithm to consistently produce routable results, some form of approximate routing topology must be explicitly constructed during placement as a guide. The principal goal of mPG [CCPY02] is to incorporate fast, constructive routing-congestion estimates, including layer assignment, into a wirelength-driven, simulated-annealing based multiscale placement engine. Compared to Gordian-L [SDJ91], mPG is 4–6.7 times faster and generates slightly better wirelength for test circuits with more than 100,000 cells. In congestion-driven mode, mPG reduces wiring overflow estimates by 45%–74%, with a 5% increase in wirelength compared to wirelength-driven mode, but 3–7% less wirelength after global routing. The results of the mPG experiments show that the multiscale placement framework is readily adapted to incorporate complex routability constraints effectively.

#### Hierarchy construction

mPG uses connectivity-driven, recursive first-choice clustering (FC, Section 2.2.1) to build its placement hierarchy on the netlist. The vertex affinity used to define clusters is similar to that used by mPL, as defined in (3). However, instead of matching a given vertex to its highest-affinity neighbor, mPG selects a neighbor at random from the those in the top 10% affinity. Moreover, the mPG vertex affinity does not consider vertex area. Experiments for mPG show that imposing explicit constraints on cluster areas in order to limit the cluster-area variation increases run time without significantly improving placement quality. The strategy in mPG is instead to allow unlimited variation in cluster areas, thereby reducing the number of cluster levels and allowing more computation time at each level.

Large variations in cluster sizes are manageable in mPG due to its hierarchical area-density model. Optimization at each level of the netlist-cluster hierarchy is performed over the *exact same* set of regular, uniform, nested bin-density grids. By gradually reducing the area overflow in bins at all scales from the size of the smallest cells up to 1/4 the placement region, a sufficiently

uniform distribution of cell areas is obtained for detailed placement. The same grid hierarchy is also used to perform fast incremental, global routing, including fast layer assignment, and to estimate routing congestion in each bin. In wirelength mode, the mPG objective is simply bounding-box wirelength, as in (1). In (routability) congestion mode, the objective is equivalent to a weighted-wirelength version of (1), in which the weight of a net is proportional to the sum of the estimated wire usages of the bins used by that net's rectilinear Steiner-tree routing. The congestion-based objective is used only at finer levels of the cluster hierarchy.

### Relaxation

Relaxation in mPG is by simulated annealing. Throughout the process, vertices are positioned only at bin centers. All vertex moves are discrete, from one bin center to another. At each step, a cluster is randomly selected. A target location for the cluster is then selected either (a) randomly within some range limit or (b) to minimize the total bounding-box wirelength of the nets containing it. The probability of selecting the target randomly is set to  $\max\{\alpha, 0.6\}$ , where  $\alpha$  is the "acceptance ratio." The probability  $p$  of accepting a move with cost change  $\Delta C$  is one if  $\Delta C \leq 0$  and  $\exp\{-\Delta C/T\}$  if  $\Delta C > 0$ , where  $T$  denotes the temperature. At the coarsest level  $k$ , the starting temperature is set to approximately 20 times the standard deviation of the cost changes of  $n_k$  random moves, where  $n_i$  denotes the number of clusters at level  $i$ . At other levels, binary search is used to estimate a temperature for which the expected cost change is zero. These approximate "equilibrium temperatures" are used as starting temperatures for those levels.

When accepting a move to a target location results in a density-constraint violation, an alternative location near the target can be found efficiently, if it exists, by means of the hierarchical bin-density structure.

Annealing proceeds at a given temperature in sweeps of  $n_i$  vertex moves as long as the given objective can be decreased. After a certain number of consecutive sweeps with net increase in the objective, the temperature is decreased by a factor  $\mu = \mu(\alpha)$ , in a manner similar to that used by Ultrafast VPR. The default stopping temperature is taken to be  $0.005C/|E|$ , where  $C$  is the objective value and  $|E|$  is the number of nets at the current level.

### Interpolation

Cluster components are placed concentrically at the cluster center.

### Iteration Flow

One V-cycle is used: recursive netlist coarsening followed by one recursive interpolation pass from the coarsest to the finest level. Relaxation is used at each level only in the interpolation pass.

### 3.4 Partitioning-based Methods

An apparent deficiency with clustering-based hierarchies is that, while they are observed to perform well on circuits that can be placed with mostly short, local connections, they may be less effective for circuits that necessarily contain a high proportion of relative long, global connections. A bottom-up approach to placement may not work well on a design formed from the top down.

A placement hierarchy can also be constructed from the top down in an effort to better capture the global interconnections of a design. An aggregate need not be defined by recursive clustering; recursive circuit partitioning (Section 2) can be used instead. In this approach, the coarsest level is defined first as just two (or perhaps four) partition blocks. Each placement level is obtained from its coarser neighboring level by partitioning the partition blocks at that coarser level.

Although partitioning's use as a means of defining a hierarchy for multi-scale placement is relatively new, placement by recursive partitioning has a long tradition. The average number of interconnections between subregions is obviously correlated with total wirelength. A good placement can therefore be viewed as one requiring as few interconnections as possible between subregions.

Minimizing cutsize is generally acknowledged as easier than placement, and, since the arrival of multiscale hypergraph partitioners hMetis and MLpart (Sections 2.2 and 2.3), little or no progress in partitioning tools has been made. Partitioning tools are thus generally seen as more mature than placement tools, whose development continues to progress rapidly. Placement algorithms based on partitioning gain some leverage from the superior performance of the state-of-the-art partitioning tools.

#### 3.4.1 Dragon

Since its introduction in 2000 [WYS00b], Dragon has become a standard for comparison among academic placers for the low wirelength and high routability of the placements it generates. Dragon combines a partitioning-based cutsize-driven optimization with wirelength-driven simulated annealing on partition blocks to produce placements at each level. Like mPG and Ultrafast VPR, Dragon relies on simulated annealing as its principal means of intralevel iterative improvement. Unlike these algorithms, Dragon's hierarchy is ultimately defined by top-down recursive partitioning rather than recursive clustering. Heavy reliance on annealing slows Dragon's performance relative to other techniques and may diminish its scalability somewhat (Section 1.5). The flexibility, simplicity, and power of the annealing-based approach, however, makes Dragon adaptable to a variety of problem formulations [SWY02, XWCS03].

### Hierarchy construction

Dragon's placement hierarchy is built from the top down. Initially, a cutsizes-driven quadrisection of the circuit is computed by hMetis (Section 2.2). Each of the four partition blocks is then viewed as an aggregate. The aggregate is given an area in proportion to its cell content, and the cells within each such aggregate are placed at the aggregate's center. Half-perimeter wirelength-driven annealing on these aggregates is then used to determine their relative locations in the placement region. Cutsizes-driven quadrisection is then applied to each of the aggregates, producing  $16 = 4 \times 4$  aggregates at the next level. Wirelength-driven annealing then determines positions of these new, smaller aggregates within some limited distance of their parent aggregates' locations. This sequence of cutsizes-driven subregion quadrisection followed by wirelength-driven positioning continues until the number of cells in each partitioning block is approximately 7. At that point, greedy heuristics are used to obtain a final, overlap free placement.

When a given partition block  $B_i$  is quadrisectioned, the manner in which its connections to other partition blocks  $B_j$  at the same level are modeled may have considerable impact. In standard non-multiscale approaches, various forms of terminal propagation (Section 3.2) are the most effective known technique. Experiments reported by Dragon's authors indicate, however, that terminal propagation is inappropriate in the multiscale setting, where entire partition blocks are subsequently moved. Among a variety of attempted strategies, the one ultimately selected for Dragon is simply to replace any net containing cells in both  $B_i$  and other partition blocks by the cells in  $B_i$  contained by that net. Thus, during quadrisection, all connections within  $B_i$  are preserved, but connections to external blocks are ignored. Connections between blocks are accurately modeled only during the relaxation phase described below.

Although hMetis is a multiscale partitioner and therefore generates its own hierarchy by recursive clustering, Dragon makes no explicit use of hMetis's clustering hierarchy in its top-down phase. Instead, Dragon uses the final result of the partitioner as a means of defining a new hierarchy from the top down. It is this top-down, partitioning-based hierarchy which defines the placement problems to which Dragon's wirelength-driven relaxations are applied.

### Relaxation

Low-temperature wirelength-driven simulated annealing is used to perform pairwise swaps of nearby partition blocks. These blocks are not required to remain within the boundaries of their parent blocks. Thus, relaxation at finer levels can to some extent correct premature decisions made at earlier, coarser levels. However, to control the run time, the range of moves that can be considered must be sharply limited.

After the final series of quadrisections, four stages proceed to a final placement. First, annealing based on swapping cells between partition blocks is performed. Second, area-density balancing linear programming [CXWS03] is used. Third, cells are spread out to remove all overlap. Finally, small permutations of cells are greedily considered separately along horizontal and vertical directions from randomly selected locations.

### Interpolation

The top-down hierarchy construction completely determines the manner in which a coarse-level solution is converted to a solution at the adjacent finer level. First, the cutsizes-driven netlist quadrisection performed by hMetis divides a partition-block aggregate into four equal-area subblocks. Second, the annealing-based relaxation is used to assign each subblock to a subregion. The initial assignment of subregions to subblocks is unspecified, but, due to the distance-limited annealing that follows, it may be taken simply as a collection of randomly selected 4-way assignments, each of these made locally within each parent aggregate’s subregion. As stated above, the ultimate subregion selected for a subblock by the annealing need not belong to the region associated with its parent aggregate.

### Iteration Flow

Dragon’s flow proceeds top-down directly from the coarsest level to the finest level. Because the multiscale hierarchy is constructed from the top down, there is no explicit bottom-up phase and thus no notion of V-cycle etc.

#### 3.4.2 Aplace

In VLSICAD, the word “analytical” is generally used to describe optimization techniques relying on smooth approximations. Among these methods, *force-directed* algorithms [QB79, EJ98] model the hypergraph netlist as a generalized spring system and introduce a scalar potential field for area density. A smooth approximation to half-perimeter wirelength (1) is minimized subject to implicit bound constraints on area-density. Regions of high cell-area density are sources of cell-displacement force gradients, regions of low cell-area density are sinks. The placement problem becomes a search for equilibrium states, in which the tension in the spring system is balanced by the area-displacement forces.

Aplace [KW04] (the ‘A’ stands for “analytic”) is essentially a multiscale implementation of this approach. The wirelength model for a net  $t$  consisting of pin locations<sup>(4)</sup>  $t = \{(x_i, y_i) \mid i = 1, \dots, \deg(t)\}$  follows a well-known log-sum-exp approximation,

<sup>(4)</sup> A *pin* is a point on a cell where a net is connected to the cell.

$$\ell_{\text{exp}}(t) = \alpha \cdot \left( \ln\left(\sum e^{x_i/\alpha}\right) + \ln\left(\sum e^{-x_i/\alpha}\right) + \ln\left(\sum e^{y_i/\alpha}\right) + \ln\left(\sum e^{-y_i/\alpha}\right) \right), \quad (4)$$

where  $\alpha$  is a smoothing parameter. To estimate area densities, uniform grids are used. Each subregion of a grid  $G$  is called a bin (the word “cell” is reserved for the movable subcircuits being placed). The scalar potential field  $\phi(x, y)$  used to generate area-density-balancing forces is defined as a sum over cells and bins as follows. For a single cell  $v$  at position  $(x_v, y_v)$  overlapping with a single bin  $b$  centered at  $(x_b, y_b)$ , the potential is the bell-shaped function

$$\phi_v(b) = \alpha(v)p(|x_v - x_b|)p(|y_v - y_b|),$$

where  $\alpha(v)$  is selected so that  $\sum_{b \in G} \phi_v(b) = \text{area}(v)$ , and

$$p(d) \equiv \begin{cases} 1 - 2d^2/r^2 & \text{if } 0 \leq d \leq r/2 \\ 2(d - r)^2/r^2 & \text{if } r/2 \leq d \leq r, \end{cases} \quad (5)$$

and  $r$  is the *radius* of the potential. The potential  $\phi$  at any bin  $b$  is then defined as the sum of the potentials  $\phi_v(b)$  for the individual cells overlapping with that bin. Let  $(X, Y)$  denote all positions of all cells in the placement region  $R$ . Let  $|G|$  denote the total number of bins in grid  $G$ . Then the target potential for each bin is simply  $\bar{\phi} = \sum_{v \in V} \text{area}(v)/|G|$ , and the area-density penalty term for a current placement  $(X, Y)$  on grid  $G$  is defined as

$$\psi_G(X, Y) = \sum_{b \in G} (\phi(b) - \bar{\phi})^2.$$

For the given area density grid  $G$ , Aplace then formulates placement as the unconstrained minimization problem

$$\min_{v \in V} \rho_\ell \left( \sum_{e \in E} \ell_{\text{exp}}(e) \right) + \rho_\psi \psi_G(X, Y)$$

for appropriate, grid-dependent scalar weights  $\rho_\ell$  and  $\rho_\psi$ . This formulation has been successfully augmented in Aplace to model routing congestion, movable I/O pads, and symmetry constraints on placed objects.

### Hierarchy construction

The aggregates are defined from the top down by recursive cutsizes-driven area bipartitioning on the hypergraph netlist. The partitioning engine used is MLpart (Section 2.3). As in Dragon, Aplace makes no attempt to reuse MLpart’s clustering hierarchy, and Aplace’s entire hierarchy of aggregates is defined before any relaxation begins. Thus, at the start of relaxation on the coarsest level, all the aggregates at all levels have been defined, but none has been assigned a position.

### Relaxation

Optimization at each level of Aplace proceeds by the Polak-Ribiere variant of nonlinear conjugate gradients [NS96] with Golden-Section linesearch [GMW81]. A hard iteration limit of 100 is imposed. The grid size  $|G|$ , objective weights  $\rho_\ell$  and  $\rho_\psi$ , wirelength smoothing parameter  $\alpha$  (4), and area-density potential radius  $r$  (5) are selected and adjusted at each level to guide the convergence. Bin size and  $\alpha$  are taken proportional to the average aggregate size at the current level. The potential radius  $r$  is set to 2 on most grids but is increased to 4 at the finest grid in order to prevent oscillations in the maximum cell-area density of any bin. The potential weight  $\rho_\psi$  is fixed at one. The wirelength weight  $\rho_\ell$  is initially set rather large and is subsequently decreased by 0.5 to escape from local minima with too much overlap. As iterations proceed, the relative weight of the area-density penalty increases, and a relatively uniform cell-area distribution is obtained.

### Interpolation

When the conjugate-gradient iterations converge, simple declustering is used. The components of each aggregate are placed concentrically at the location of the aggregate's center. This configuration is used as the starting point for conjugate-gradient iterations at the adjacent, finer level.

### Iteration Flow

As in Dragon, there is just one pass from the coarsest to the finest level. There is no bottom-up aggregation, and there are no V-cycles. After conjugate gradients at the finest level, a simple Tetris-based legalizer [Hil02] is used to remove overlap in a way that tends to preserve cells' relative orderings.

## 3.5 Numerical Comparison

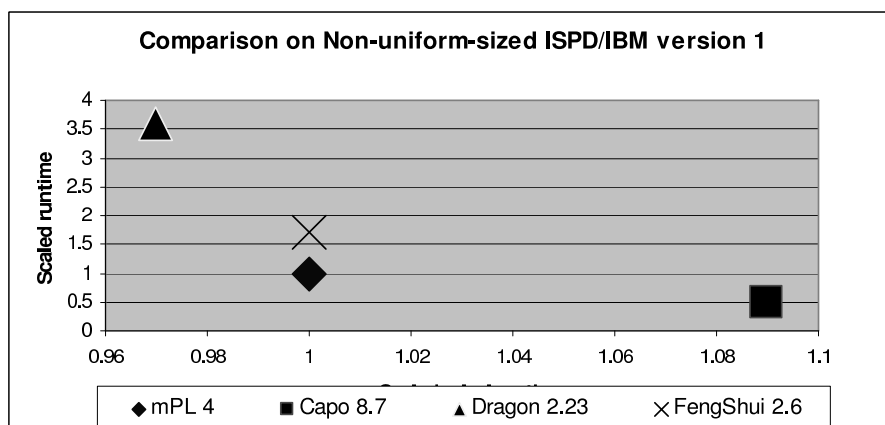
The results of Dragon and mPL on the IBM/ISPD98 benchmarks [Alp98] are compared<sup>(5)</sup> in Figure 15 to those of two leading representatives of the top-down partitioning-based paradigm described in Section 3.2: Capo and Feng-Shui. To illustrate the time/quality trade-offs, the figure shows scaled run time vs. scaled total wirelength. All values have been scaled relative to those for mPL 4. From the figure, the following conclusions can be drawn.

(i) Dragon produces the best wirelength, by 4%, but requires the most run time, by about 2.2 $\times$ .

(ii) Capo requires the least run time (by 3 $\times$ ) but produces the longest wirelength (by about 9%);

(iii) mPL and Feng-Shui produce the best wirelength-to-run-time tradeoff; both are competitive with Dragon in wirelength and with Capo in run time.

<sup>(5)</sup> As this chapter goes to press, no executable for Aplace is yet publicly available.



**Fig. 15.** Comparison of 4 state-of-the-art placement algorithms: mPL4, Dragon 2.23, Capo 8.7, and Feng-Shui 2.2. All values are scaled relative to those for mPL4.

## 4 Multiscale Routing

After placement, the positions and the sizes of the cells and the large *intellectual-property* (IP) blocks are determined. In the following step, routing, the placed cells and IP blocks are connected by metal wires according to the netlist and subject to the constraints of design rules, timing performance, noise, etc. As VLSI technology reaches deep-submicron feature size and gigahertz clock frequencies, the interconnect has become the dominant factor in the performance, power, reliability, and cost of the overall system.

The difficulty of VLSI routing has increased with the development of IC technology for the following reasons.

- The current trend of component-based design requires a flexible, full-chip based (i.e., not reduced by partitioning), area-routing algorithm that can simultaneously handle both the interconnects between large macro blocks and the interconnects within individual blocks.
- The numbers of gates, nets and routing layers in IC designs keep increasing rapidly, and the size of the routing problem grows correspondingly. According to the International Technology Roadmap for Semiconductors [itr], a router may have to handle designs of over 100 million transistors and 8–9 metal layers in the near future, assuming the full-chip based routing approach.
- Many optimization techniques, including high-performance tree structures such as A-tree, BA-tree, etc.; buffer insertion; wire sizing; and wire order-

ing and spacing; have been proposed for high-performance circuit designs. These make the routing problem even more complicated.

In this section, we review recent efforts to apply multiscale optimization to VLSI routing. We start with a brief introduction to the routing problem and basic techniques.

#### 4.1 Introduction to VLSI Routing

*Routing* is the process of finding the geometric layouts of all the nets. The input of a routing problem includes the following.

**routing area**— the dimensions of the rectangular routing region and the number of available routing layers.

**netlist**— required interconnects as sets of connection points (“pins”) on placed objects.

**design rules**— specifications of the minimum and/or maximum values allowed for wire width, wire spacing, via width, via spacing, etc.

**pin locations**— as determined by the placement or the floorplanning process.

**obstacles**— objects which wires cannot pass through, including placed cells, IP blocks, pads, pre-routed wires, etc.

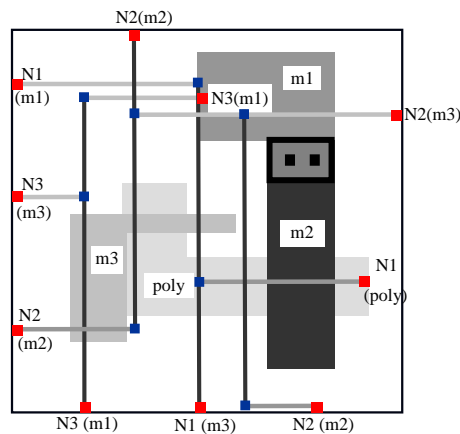
**constraint-related parameters**— such as the electrical parameters (e.g., RC constants) for performance and signal integrity constraints; thermal parameters for the thermal constraints, etc.

The constraints of a general routing process usually include *connection rules* and *design rules*. Connection rules stipulate that (i) to prevent open circuits, wires of the same net must be connected together; and (ii) to avoid short circuits, wires of different nets must not overlap with each other at the same layer. Design rules specify the sizes of the wires and vias (connections between layers) and the minimum spacing between them according to the available manufacturing technology. The objective of the routing problem depends on the circuit design. For general purpose chips, the objective is usually the total wire length of all nets. For high-performance chips, the delay of the circuit is minimized. Figure 16 shows a very simple routing example with three nets.

#### 4.2 Basic Techniques for Single Two-Pin Net Routing

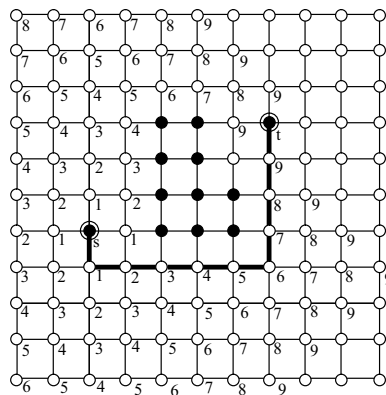
In single two-pin-net routing, a path for one single net with two terminals is sought. Single two-pin-net routing is a necessary component for every routing algorithm. Basic techniques for it include maze-routing algorithms and line-probe algorithms.

The first maze-routing algorithm was proposed in 1961 by Lee [Ake67], and later on, the algorithm was improved in both time and space complexity [Sou78, Had75]. Lee’s basic maze-routing algorithm searches for a path



**Fig. 16.** A routing example of four layers — poly, m1, m2, and m3 — and three nets — N1, N2, and N3.

from source  $s$  to target  $t$  in a breadth-first fashion. The routing area is represented by a grid map, with each bin in the grid either empty or occupied by an obstacle. The algorithm starts the search process from the  $s$  like a propagating wave. Source point  $s$  is labeled '0', and all the unblocked neighbors are labeled as '1'. All empty neighbors of all '1'-labeled nodes are then visited and labeled as '2', and so on until the target  $t$  is finally reached. An example is shown in Figure 17. The shortest-path solution is guaranteed as long as a solution exists.



**Fig. 17.** A net routed by a maze-routing algorithm.

Line-probe algorithms were proposed independently by Mikami and Tabuchi in 1968 [MT68] and by Hightower in 1969 [Hig69]. The basic difference between

line-probe and maze-routing algorithms is that the line-probe algorithms use line segments instead of grid points to search for a path. Search lines are generated from both the source  $s$  and the target  $t$ . At the next iteration, new line segments are generated from *escape* points selected along the existing line segments. The generated lines cannot go through an obstacle or the area boundary. The main difference between the Mikami algorithm and the Hightower algorithm lies in the selection of escape points. In the Mikami algorithm, every point on the search line is an escape point; hence, the optimal solution is guaranteed, if a path exists. In the Hightower algorithm, only one escape point will be generated for each line segment, the choice of location determined by heuristics designed to support navigation around obstacles. However, the shortest path is not guaranteed, and a path may not be found, even when one exists. Figure 18 shows an example of routing a net using the Hightower line-probe algorithm.

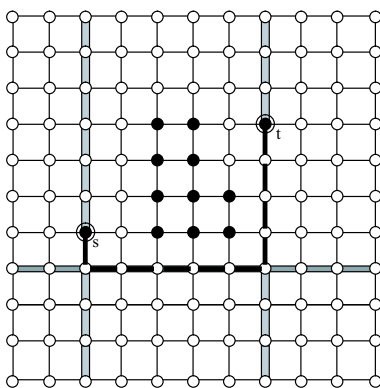


Fig. 18. A net routed by a line-search algorithm.

### 4.3 Traditional Two-Level Routing Flow

The traditional routing system is usually divided into two stages: global routing and detailed routing, as shown in Figure 19. During global routing, the entire routing region is partitioned into tiles or channels. A rough route for each net is determined among these tiles to optimize the overall congestion and performance. After global routing, detailed routing is performed within each tile or channel, and the exact geometric layout of all nets is determined.

There are two kinds of global routing algorithms, sequential and iterative. Sequential methods route the nets one by one in a predetermined order, using either maze routing or line probing. However, the solution quality is often affected by the net ordering. Iterative methods try to overcome the net ordering problem by performing multiple iterations. A negotiation-based iterative global routing scheme was proposed [Nai87] and later used in FPGA

routing [ME95]. In flow-based iterative methods [CLC96, Alb00], the global routing problem is relaxed to a linear programming formulation and solved by an approximate multi-terminal, multicommodity flow algorithm. A combination of maze routing and iterative deletion has recently been used for performance-driven global routing [CM98].

There are two types of detailed routing approaches, grid-based and gridless. In grid-based detailed routing, routing grids with fixed spacings from the design rules are defined before routing begins. The path of each net is restricted to connect grid points rectilinearly. As the grids are uniform, the layout representation in grid-based routing is quite simple, usually just a 3-dimensional array. Variable widths and spacings may be used for delay and noise minimization (e.g. [CHKM96, CC00]). A gridless detailed router allows arbitrary widths and spacings for different nets; these can be used to optimize performance and reduce noise. However, the design size that a gridless router can handle is usually limited, due to its higher complexity.

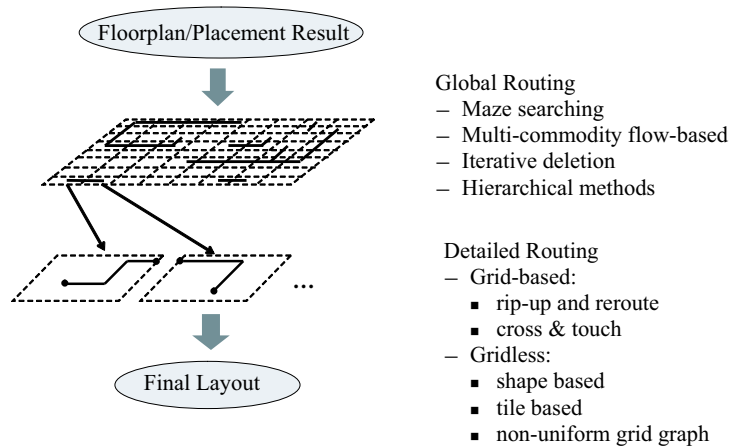


Fig. 19. Traditional two-level routing flow.

#### 4.4 More Scalable Approaches

Because routing tiles are confined to discrete routing layers, the global routing problem is sometimes described as “2.5-dimensional.” In this setting, flat two-level (global routing + detailed routing) routing approaches have two limitations in current and future VLSI designs. First, future designs may integrate over several hundreds of millions of transistors in a single chip. The traditional two-level design flow may not scale well to handle problems of such size. For example, a  $2.5 \times 2.5$  cm<sup>2</sup> chip using a  $0.07\mu\text{m}$  processing technology may have over 360,000 horizontal and vertical routing tracks at the full-chip

level [itr]. That translates to about  $600 \times 600$  routing tiles, if the problem size is balanced between global-routing and detailed-routing stages, and presents a significant challenge to the efficiencies of both stages. To handle the problem, several routing approaches have been proposed to scale to large circuit designs.

Top-down hierarchical approaches have also been used to handle large routing problems. The flow is shown in Figure 20. The first hierarchical method was proposed for channel routing by Burstein [BP83]. Heisterman and Lengauer proposed a hierarchical integer programming-based algorithm for global routing [HL91]. Wang and Kuh proposed a hierarchical  $(\alpha, \beta)^*$  algorithm [WK97] for multi-chip module (MCM) global routing. The main problems with the hierarchical approaches are (1) the higher-level solutions over-constrain the lower level solutions, and (2) the lack of detailed routing information at coarser levels makes it difficult to make good decisions there. Therefore, when an unwise decision is made at some point, it is very costly, through rip up and reroute, to revise it later at a finer level. To overcome the disadvantages of hierarchical methods, hybrid routing systems have been proposed. Lin, Hsu, and Tsai proposed a combined top-down hierarchical maze-routing method [LHT90]. Parameter-controlled expansion instead of strictly confined expansion is used to overcome the first disadvantage, but there is still no way to learn finer-level routing information at coarser levels. Hayashi and Tsukiyama proposed a combination of a top-down and a bottom-up hierarchical approach [HT95], aiming to resolve the second problem of the original hierarchical approach, but the fine-level planning results are still fixed once they are generated, causing a net-ordering problem.

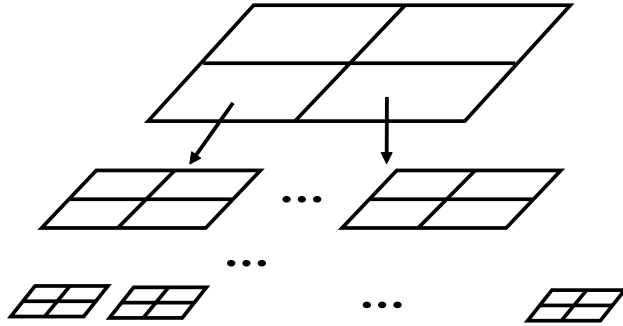


Fig. 20. Hierarchical routing flow.

A 3-level routing scheme with an additional wire-planning phase between performance-driven global routing and detailed routing has also been proposed [CFK00] and is illustrated in Figure 21. The additional planning phase improves both the completion rate and the runtime. However, for large designs, even with the three-level routing system, the problem size at each level

may still be very large. For the previous example of a  $2.5 \times 2.5 \text{ cm}^2$  chip using a  $0.07\mu\text{m}$  processing technology, the routing region has to be partitioned into over  $100 \times 100$  tiles on both the top-level global routing and the intermediate-level wire planning stage (assuming the final tile for detailed routing is about  $30 \times 30$  tracks for the efficiency of the gridless router). Therefore, as the designs grow, more levels of routing are needed. Rather than a predetermined, manual partition of levels, which may have discontinuities between levels, an automated flow is needed to enable seamless transitions between levels.

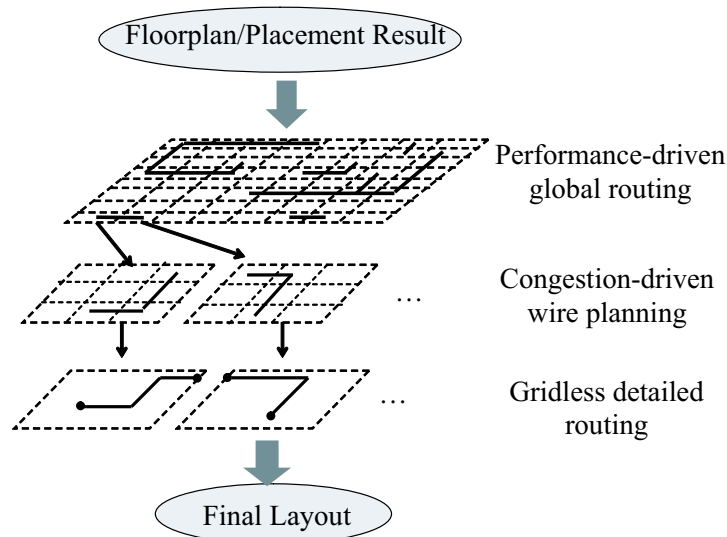
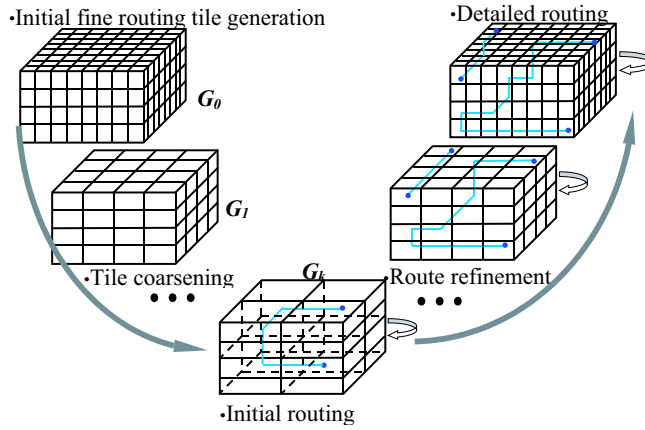


Fig. 21. 3-level routing flow.

#### 4.5 MARS – A Multiscale Routing System

The success of multiscale algorithms in VLSI partitioning and placement led naturally to the investigation of multiscale methods for large-scale VLSI routing. A novel multiscale router, MRS [CFZ01], was proposed for gridless routing. Soon afterward, an enhanced, more effective version of MRS called MARS (*Multiscale Advanced Routing System*) was proposed [CXZ02]. Independently of MARS, the MR system [CL04] was also developed based on the multiscale-routing framework of MRS.

The MARS algorithm is reviewed in this section. Section 4.5.1 provides an overview. Section 4.5.2 describes the tile-partitioning and resource-estimation algorithms employed during coarsening. Section 4.5.3 describes the multi-commodity flow algorithm used to compute the initial routing solution at the coarsest level. The history-based iterative refinement process is explained



**Fig. 22.** Multiscale Routing Flow.

in Section 4.5.4. Results of experiments with MARS are presented in Section 4.5.7.

#### 4.5.1 Overview of the MARS Routing Flow

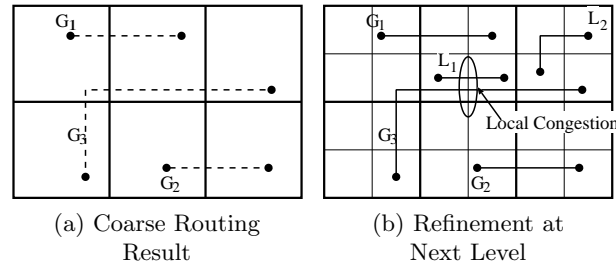
Figure 22 illustrates the V-cycle flow for MARS. The routing area is partitioned into routing tiles. The algorithm employs a multiscale planning procedure to find a tile-to-tile path for each net. In contrast, most traditional global-routing algorithms [CLC96, Alb00, CM98] try to find routing solutions on the finest tiles directly. For large designs, the number of tiles may be too large for the flat algorithms to handle.

The multiscale approach first estimates routing resources using a line-sweeping algorithm on the finest tile level. A recursive coarsening process is then employed to merge the tiles, to reserve the routing resources for local nets, and to build a hierarchy of coarser representations. At each coarsening stage, the resources of each tile are calculated from the previously considered finer-level tiles which form it. Also, part of the routing resources are assigned to nets local to that level.

Once the coarsening process has reduced the number of tiles below a certain limit, the initial routing is computed using a multicommodity flow-based algorithm. A congestion-driven Steiner-tree generator gradually decomposes multi-pin nets into two-pin nets. During recursive interpolation, the initial routing result is refined at each level by a maze-search algorithm. When the final tile-to-tile paths are found at the finest level of tiles for all the nets, a gridless detailed routing algorithm [CFK99] is applied to find the exact path for each net.

The V-cycle multiscale flow has clear advantages over the top-down hierarchical approach. The multiscale subproblems are closer to the original

problem, because at each level, all nets, including those local to the current level, are considered during resource reservation. In the interpolation pass, the finer-level router may alter the coarser level result according to its more detailed information about local resources and congestion. The coarse-level solution is used as a *guide* to the finer-level solver, but not as a *constraint*. Compared to the traditional top-down hierarchical approach, this feature leads the multiscale method to better solutions with higher efficiency.



**Fig. 23.** Limitation of top-down hierarchical approaches. In a top-down hierarchical approach, coarse-level decisions are based on limited information (a). Finer-level refinement cannot change the coarse-level solution, even when it leads to local congestion at the finer level (b).

#### 4.5.2 Coarsening Process and Resource Reservation

The hierarchy of coarse-grain routing problems is built by recursive aggregation of routing subregions. Initially, at the finest level, all routing layers are partitioned the same way into regular 2-D arrays of identical rectangular tiles. The finest level is denoted Level 0. A three-dimensional routing graph  $G_0$  is built there, each node in  $G_0$  representing a tile in some routing layer at Level 0. Every two neighboring nodes in  $G_0$  that have a direct routing path between them are connected by an edge. The edge capacity represents the routing resources at the common boundary of the two tiles. The ultimate objective of the multiscale routing algorithm is to find, for each net, a tile-to-tile path in  $G_0$ . These paths are then used to guide the detailed router in searching for actual connections for the nets.

The multiscale router first accurately estimates the routing resources at the finest level, then recursively coarsens the representations into multiple levels. The number of routing layers does not change during the coarsening. All layout objects such as obstacles, pins and pre-routed wires are counted in the calculation. Because the detailed router is gridless, actual obstacle dimensions are used to compute the routing-resource estimates. Three kinds of edge capacities are computed: *wiring capacity*, *interlayer capacity* and *through capacity*.

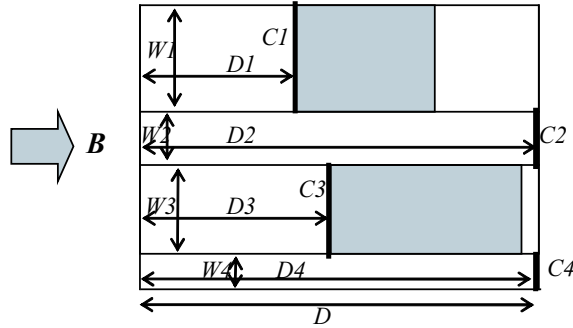
A line-sweeping resource-estimation algorithm [CFK00] is used in MARS. This method *slices* each routing layer along lines parallel to the layer's wiring direction, horizontal or vertical. The *depth* of a slice relative to boundary  $B$  is the distance along the slice from  $B$  to the nearest obstacle in the slice or, if there is no obstacle, to the end of the tile. For the example in Figure 24, the wiring capacity at boundary  $B$  can be computed as a weighted sum of slice areas reachable from  $B$  along lines in the slice direction,

$$C = \sum_i (W_i \times D_i / D), \quad (6)$$

where  $W_i$  and  $D_i$  are the width and depth of each slice  $S_i$ , and  $D$  is the tile depth. To calculate  $W_i$  and  $D_i$ , we maintain a *contour list* of  $B$ , defined as a sorted list of the boundaries of all the rectangular obstacles that can be seen from  $B$ . In Figure 24, the *contour list* of  $B$  is  $C_1, C_2, C_3, C_4$ .

The interlayer edge capacity, which corresponds to the resources used by vias, is calculated as the sum of the areas of all empty spaces in the tile. The through capacity, which corresponds to the paths that go straight through a routing tile, is the sum of the boundary-capacity contributions of those empty rectangles that span the whole tile. The through capacity at the horizontal direction of the tile in Figure 24 is  $C_{th} = W_2 + W_4$ .

All three capacities contribute to the path costs. For the example in Figure 25, the total path cost  $C_{path} = c_{1,right} + c_{2,left} + c_{2,right} + c_{2,through} + c_{3,left} + c_{3,up} + c_{4,down} + c_{4,right} + c_{5,left}$ , where  $c_{1,right}, c_{2,left}, c_{2,right}, c_{3,left}, c_{4,right}$  and  $c_{5,left}$  are the costs related to the wiring capacities of tiles 1,2,3,4 and 5;  $c_{2,through}$  is the cost corresponds to the through capacity of tile 2; and  $c_{3,up}, c_{4,down}$  are the via costs related to interlayer capacities.



**Fig. 24.** Resource-estimation model for one layer. Shaded rectangles represent obstacles.

Given the accurate routing-capacity estimation at the finest level and the grid graph  $G_0$ , the coarsening process generates a series of reduced graphs

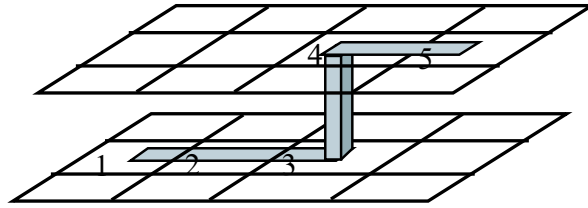


Fig. 25. Path cost.

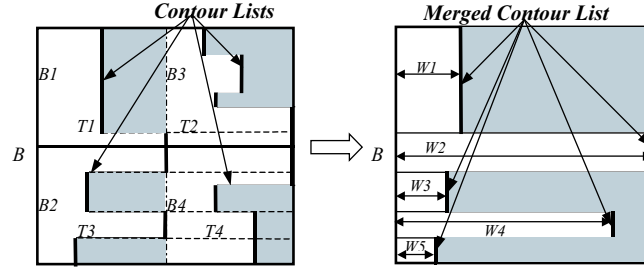
$G_i$ , each representing a coarsened Level- $i$  routing problem,  $P_i$ , with a different resolution. At a coarser Level  $i + 1$ , the tiles are built simply by merging neighboring tiles on the adjacent, finer Level  $i$ . The coarser-level graph,  $G_{i+1}$ , which represents the routing resources available at the coarse tiles, can also be derived from the finer-level graph  $G_i$  directly. The routing graphs are iteratively coarsened until the size of the graph falls below a predetermined limit.

### Resource Merging

Every move from a finer Level  $i$  to a coarser Level  $i + 1$  requires merging a certain number of component tiles —  $2 \times 2$  in our implementation. A new contour list for a resulting Level- $i + 1$  tile is obtained by merging the contour lists of the component Level- $i$  tiles. The wiring capacities of the new tile can also be derived from (6). Figure 26 illustrates the merging process. Level- $i$  tiles  $T_1, T_2, T_3$  and  $T_4$ , whose left boundaries are  $B_1, B_2, B_3$  and  $B_4$ , respectively, are to be merged. The contour lists of  $B_1, B_2, B_3$  and  $B_4$  are retrieved and merged into the contour list of the new edge  $B$ . Since the contour lists are sorted, the merging process can be accomplished in  $O(n)$  time, where  $n$  is the number of segments in the new contour list. With the contour list of  $B$ , it is straightforward to derive the rectangles abutting  $B$  and then calculate the wiring capacity of  $B$ . The interlayer capacity of the new tile is calculated as the sum of the interlayer capacities of the component tiles. The through capacity of the new tile is calculated as the sum of the heights of the empty slices that span the whole tile.

### Resource Reservation

The estimation computed by the above procedure still cannot precisely model available routing capacities at coarser levels. When the planning engine moves from Level  $i$  to coarser Level  $i + 1$ , a subset of the nets at Level  $i$  become completely local to individual Level- $i + 1$  tiles. Such nets are therefore invisible at that level and all coarser levels. If the number of such local nets is large, a solution to the coarse-level problem may not be aware of locally congested areas and may therefore generate poor planning results.



**Fig. 26.** Merging contour lists.

Figure 27(a) shows an example of the effect of local nets. Nets 1 and 2 are local to Level 1, and appear at Level 0. Net 3 is local to Level 2, and Net 4 is global to the coarsest level (level 2). Each net is planned without any information about the nets local to the planning level. Net 3 and Net 4 will be planned as in Figure 27(a). Both Net 3 and Net 4 have to be changed in Level-0 planning to minimize the local congestion. This adjustment not only places a heavier burden on the refinements at finer levels but also wastes the effort spent at coarser levels.

In order to correctly model the effect of local nets, the resources that will be used by nets local to each level are predicted and reserved during coarsening. This process is called *resource reservation*. More specifically, suppose the coarsening process goes through Level 0, Level 1, ..., Level  $k$ , with Level 0 being the finest level. Let  $c_{i,j}$  denote the initial capacity of edge  $e_{i,j}$  in routing graph  $G_i$ , and the capacity vector  $C_i = [c_{i,1}, c_{i,2}, \dots, c_{i,m}]$  represents all routing capacities at Level  $i$ . Let  $T_{n,i} = \{\text{the minimal set of tiles in which the pins of net } n \text{ are located on Level } i\}$ , which is called the *spanning tile set* of net  $n$  on Level  $i$ . The level of net  $n$ ,  $\text{level}(n)$ , is defined as the level above which a net becomes contained within the boundary of one tile. It is calculated as follows:

$$\text{level}(n) = \begin{cases} k & \text{if } |T_{n,k}| > 1 \\ -1 & \text{if } |T_{n,0}| = 1 \\ \min\{i \mid |T_{n,i}| > 1 \text{ and } |T_{n,i+1}| = 1\} & \text{otherwise.} \end{cases} \quad (7)$$

Let  $L_i = \{n \mid \text{level}(n) = i\}$ ;  $L_i$  is called the *local net set* on level  $i$ . Let  $M_i = \{n \mid \text{level}(n) > i\}$ ;  $M_i$  is called the *global net set* on level  $i$ . To better estimate the local nets, we use a maze-routing engine to find a path for each net in  $L_i$  before going from Level  $i$  to Level  $i+1$ . Then we deduct the routing capacity taken by these local nets in resource reservation. Figure 28 shows an example of the calculation of the reservation for edges  $CD$ ,  $AC$  and  $BD$  of a Level  $i+1$  tile. An L-shaped path connects pins  $s$  and  $t$  in a horizontal layer. The wiring capacities on  $CD$ ,  $AC$  and  $BD$  are first calculated from (6). After the horizontal wire is added, one segment in the contour list of  $CD$  will be pushed right by  $h$ . Therefore, the reserved capacity is  $r = w \cdot h_1/h$ , where

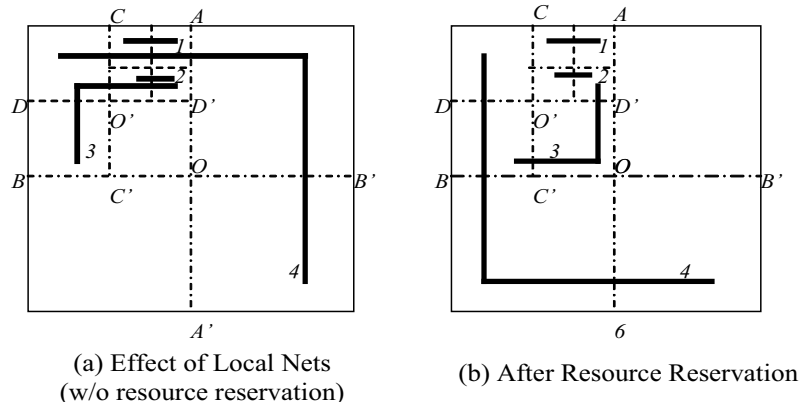


Fig. 27. Effect of resource reservation.

$w$  is the wire width. Similarly, the vertical resource reservations on  $AC$  and  $BD$  are  $w \cdot v_1/v$  and  $w \cdot v_2/v$ , respectively. However, since pins are treated as obstacles in contour-list generation, the capacity reservation on  $AB$  remains zero. A vector  $R_{i+1} = [r_{i+1,1}, r_{i+1,2}, \dots, r_{i+1,j}]$  can be obtained by repeating this process; each element corresponds to the reservation on  $e_{i+1,j}$  in  $G_{i+1}$ . The routing capacity of edges in  $G_{i+1}$  is updated as  $C_{i+1} = C_{i+1} - R_{i+1} = [c_{i+1,1} - r_{i+1,1}, c_{i+1,2} - r_{i+1,2}, \dots, c_{i+1,j} - r_{i+1,j}]$ .

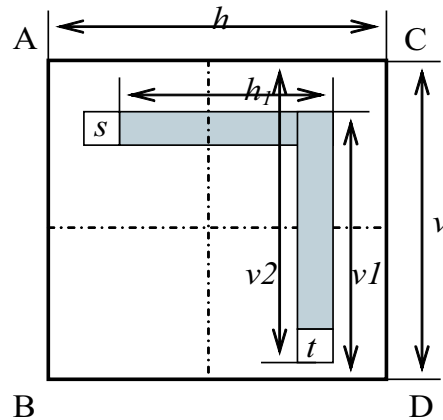


Fig. 28. Reservation calculation.

Figure 27(b) shows the planning result with the resource-reservation approach. Net 1 and Net 2 are routed at Level 0, and the reservations are made for them on  $CO'$  and  $AD'$ . At Level 1, Net 3 will take a route different from

that of Figure 27(a), due to the resource reservation for local nets made at Level 0. For the same reason, Net 4 is routed through the less congested tiles at Level 2.

Because the local nets at a given level occupy a small number of adjacent or nearly adjacent tiles, the maze-routing engine will route them very quickly. Hence, the reservation procedure is very fast. The routes during this phase are usually short and straight, so the reservation amount is close to the lower bound of the resources actually needed by the nets. Furthermore, the reserved routes are not fixed and can be changed during the refinement process when necessary.

### 4.5.3 Initial Routing

The coarsening process stops once the size of the tile grid falls below a certain user-specified value; the default maximum coarsest-level size is  $20 \times 20$  on each routing layer. A set of tile-to-tile paths is then computed at the coarsest level for the nets which cross tile boundaries at that level. This process is called the *initial routing*. It is important to the final result of multiscale routing for two main reasons. First, long interconnects that span more tiles are among the nets that appear during the initial routing. Normally, these long interconnects are timing-critical and may also suffer noise problems due to coupling along the paths. Initial routing algorithms should be capable of handling these performance issues. Second, the initial routing result will be carried all the way down to the finest tiles through the refinement process in the multiscale scheme. Although a multiscale framework allows finer-level designs to change coarser-level solutions, a bad initial routing solution will slow down the refinement process and is likely to degrade the final solution.

In MARS, a multicommodity flow-based algorithm is used to compute the initial routing solution at the coarsest level,  $k$ . The flow-based algorithm is chosen for several reasons. First, the flow-based algorithm is fast for the relatively small grid size at the coarsest level. Also, the flow-based algorithm considers all the nets at the same time and thus avoids the net-ordering problem of net-by-net approaches. Last, the flow-based algorithm can be integrated with other optimization algorithms to consider special requirements of certain critical nets. For example, high-performance tree structures such as A-Tree [CKL99], or buffered tree [CY00] can be taken instead of the Steiner tree as the candidate tree structure in the flow-based initial routing.

The objective of the initial routing is to minimize congestion on the routing graph. A set of candidate trees is computed for each net on the coarsest level routing graph  $G_k$ . For a given net  $i$ , let  $P_i = \{P_{i,1}, \dots, P_{i,l_i}\}$  be the set of possible trees. In the current version of MARS, graph-based Steiner trees are used as candidates for each net. Assume the capacity of each edge on the routing graph is  $c(e)$ , and  $w_{i,e}$  is the cost for net  $i$  to go through edge  $e$ . Let  $x_{i,j}$  be an integer variable with possible values 1 or 0 indicating if tree  $P_{i,j}$  is chosen or not ( $1 \leq j \leq l_i$ ). Then, the initial routing problem can be

formulated as a mixed integer linear-programming problem as follows:

$$\begin{aligned}
 & \min \lambda \\
 & \text{subject to} \\
 & \sum_{i,j:e \in P_{i,j}} w_{i,e} x_{i,j} \leq \lambda c(e) \text{ for } e \in E \\
 & \sum_{j=1}^{l_i} x_{i,j} = 1 \text{ for } i = 1, \dots, n_k \\
 & x_{i,j} \in \{0, 1\} \text{ for } i = 1, \dots, n_k;
 \end{aligned} \tag{8}$$

where  $n_k$  is the number of nets to be routed at level  $k$ . Normally, this mixed integer linear-programming problem is relaxed to a linear-programming problem by replacing the last constraint in Equation 8 by

$$\begin{aligned}
 x_{i,j} & \geq 0 \text{ for } i = 1, \dots, n_k; \\
 & j = 1, \dots, n_k;
 \end{aligned} \tag{9}$$

After relaxation, a maximum-flow approximation algorithm can be used to compute the value of  $x_{i,j} \in [0, 1]$  for each net in the above linear-programming formulation. The algorithm in (Figure 29) is implemented based on an approximation method [Alb00]. Garg and Konemann [GK98] have proved the optimality of this method and have given a detailed explanation of its application to multicommodity flow computation.

After the fractional results for each path are computed, a randomized rounding algorithm is used to convert the fractional values into 0 or 1 values for candidate paths of each net so that one path is chosen for each net. Error bounds have been estimated for the randomized rounding approach to global routing [KLR<sup>+</sup>87].

#### 4.5.4 History-Based Incremental Refinement

One major difference between the hierarchical routing and multiscale routing approaches is that a multiscale framework allows the finer-level relaxations to change coarser-level routing solutions. In the interpolation phase of the multiscale framework, paths computed by the initial flow-based algorithm are refined at every level.

In MARS, a congestion-driven minimum Steiner tree is used to decompose a multipin net. The Steiner tree is generated using a point-to-path  $A^*$ -search algorithm in both the initial routing and the relaxation at finer levels. Steiner-tree-based decomposition of a multipin net achieves better wire length and routability than a minimum-spanning-tree-based decomposition.

#### 4.5.5 The Path-Search Algorithm

Two types of nets must be handled at each level of the refinement. One type is “new” nets that appear just at the current level, as shown by solid lines in Figure 30(b). These nets are relatively short and do not cross coarser tile

```

parameter initialization;
for each iteration {
  for each net {
    if((there is no candidate topology for this net) ||
      (the cost of the net's last topology in current
       graph increases too much))
      generate a new topology T for this net;
    else
      keep the last topology T;
      assign a unit flow to T;
      update the routing graph edges;
    }
  }
  pick one topology for each net by randomized
  rounding according to the assigned flow volume;

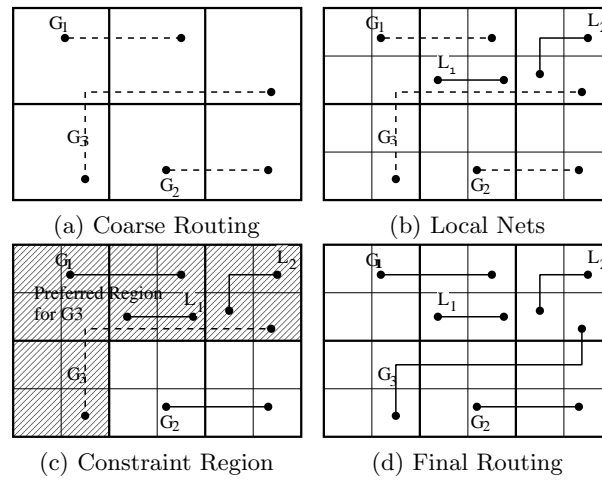
```

Fig. 29. Approximate multicommodity-flow algorithm.

boundaries, so they are not modeled at coarser levels. These nets are called *local nets*. New paths must be created for local nets at the current level. Another set of nets, *global nets*, are those carried over from the previous coarser-level routing, whose paths must be refined at the current level. During each refinement stage, local nets are routed prior to global nets. Finding paths for the local nets is relatively easy, as each local net crosses at most two tiles. Furthermore, routing local nets before any refinement provides a more accurate estimation of local resources.

The major part of the refinement work comes from refining the global nets. In general, the amount of work needed for refinement depends on the quality of the coarse-level solution. At one extreme, the choice of paths at the coarser level is also optimal at the current level. The paths need only be refined within the regions defined by the paths in coarse tiles. With this restriction, the multiscale algorithm can be viewed as a hierarchical algorithm. At the other extreme, when the coarser solution is totally useless or misleading, optimization at the current level must discard the coarser level solution and search for a new path for each coarse-level net among all finer tiles at this level. In this case, whole-region search is required at the current level. The reality lies between these two extreme cases. A good coarse-level routing provides hints as to where the best solution might be. However, if the search space is restricted to be totally within the coarse level tiles as in a hierarchical approach, the flexibility to correct the mistakes made by coarser levels will be lost.

In order to keep the coarser-level guide while still maintaining the flexibility to search all finer tiles, a net-by-net refinement algorithm using a modified maze-search algorithm is implemented. Figure 30 shows an example. The path on coarser tiles guides the search for a path at the current level. A *preferred region* is defined as the set of tiles that the coarse-level path goes through. Weights and penalties associated with each routing-graph edge are computed based on the capacities and usage by routed nets. Additional penalties are assigned to graph edges linking to and going between the graph nodes corresponding to tiles that are not located within the preferred region, as shown in Figure 30(c). Dijkstra’s shortest-path algorithm [Dij59] is used to find a weighted shortest path for each net, considering wire length, congestion, and the coarser-level planning results. In general, Dijkstra’s algorithm may be slow in searching for a path in a large graph. However, by putting penalties to non-preferred regions, we can guide the path to search within the preferred regions first. The penalty is chosen so that it does not prevent the router from finding a better solution that does not fall into the preferred region.



**Fig. 30.** Constrained maze refinement. Previous routing at coarse grid is shown in (a). Local nets at current level are routed first. Preferred regions are defined by a path at the coarser tile grid, as shown in (c). However, the modified maze algorithm is not restricted to higher-level results and can leave the preferred regions according to local congestion, thereby obtaining better solutions (d).

#### 4.5.6 History-based Iterative Refinement

Limiting to only one round of refinement may not be enough to guarantee satisfactory results. In MARS, a form of history-based iterative-refinement [ME95]

is applied at each level. Edge routing costs are iteratively updated with the consideration of historical congestion information and all nets are re-routed based on the new edge costs. The cost of edge  $e$  during the  $i$ th iteration is calculated by:

$$Cost(e, i) = \alpha \cdot congestion(e, i) + \beta \cdot history(e, i) \quad (10)$$

$$history(e, i) = history(e, i - 1) + \gamma \cdot congestion(e, i - 1) \quad (11)$$

where  $congestion(e, i)$  is a three-tiered slope function of the congestion on  $e$ ,  $history(e, i)$  is the history cost, indicating how congested that edge was during previous iterations, and  $\alpha, \beta, \gamma$  are scaling parameters. Explicit use of congestion history is observed to prevent oscillations and smooth the convergence of the iterative refinement.

The congestion estimates of the routing edges are updated every time a path of a net is routed. After each iteration, the history cost of each edge is increased according to (11). Then the congestion estimates of all edges are scanned to determine whether another iteration is necessary. If so, all edge usages are reset to zero and the refinement process at the same level is restarted.

#### 4.5.7 Experiments and Results

MARS has been implemented in C++. The multiscale-routing results are finalized using an efficient multilayer gridless routing engine [CFK99]. MARS has been tested on a wide range of benchmarks, including MCM examples and several standard cell examples, as shown in Table 5 [CXZ02]. Mcc1 and Mcc2 are MCM examples, where Mcc1 has 6 modules and Mcc2 has 56 modules [CFZ01]. The results are collected on a Sun Ultra-5 440Mhz with 384MB of memory.

MARS is compared with the three-level routing flow recently presented at ISPD 2000 [CFK00]. The three-level flow features a performance-driven global router [CM98], a noise-constrained wire-spacing and track-assignment algorithm [CC00], and finally a gridless detailed-routing algorithm with wire planning [CFK00]. In this experiment, the global router partitions each example into  $16 \times 16$  routing tiles. Nets crossing the tile boundaries are partitioned into subnets within each tile. After the pin assignment, the gridless detailed routing algorithm routes each tile one by one. “#Total Sub-nets” are the total two-pin nets seen by the detailed router. Since long two-pin nets are segmented to shorter subnets, this number not only depends on the number of multiple-pin nets, but also depends on the net planning results. From the results in Table 6, the multiscale routing algorithm helps to eliminate failed nets and reduces the runtime by  $11.7\times$ .

The difference between a multiscale router and a hierarchical router has been discussed in Section 4.5.1. For comparison, the MARS implementation has been modified and transformed into a hierarchical implementation. Table 7

**Table 5.** Examples used for multiscale routing.

Circuit	Size ( $\mu\text{m}$ )	# Layers	#2-Pin Nets	# Cells	# Pins	# Levels	Division
Mcc1	39000 $\times$ 45000	4	1694	MCM	3101	2	$25 \times 22$
Mcc2	152400 $\times$ 152400	4	7541	MCM	25024	3	$43 \times 43$
Struct	4903 $\times$ 4904	3	3551	n/a	5471	5	$273 \times 273$
Primary1	7552 $\times$ 4988	3	2037	n/a	2941	3	$53 \times 35$
Primary2	10438 $\times$ 6468	3	8197	n/a	11226	6	$73 \times 46$
S5378	4330 $\times$ 2370	3	3124	1659	4734	3	$61 \times 34$
S9234	4020 $\times$ 2230	3	2774	1450	4185	3	$57 \times 32$
S13207	6590 $\times$ 3640	3	6995	3719	10562	4	$92 \times 51$
S15850	7040 $\times$ 3880	3	8321	4395	12566	4	$98 \times 55$
S38417	11430 $\times$ 6180	3	21035	11281	32210	3	$159 \times 86$
S38584	12940 $\times$ 6710	3	28177	14716	42589	4	$180 \times 94$

compares the routing results of such a hierarchical approach with those of the multiscale approach. Although the hierarchical approach gains a little bit in runtime in some cases, by constraining the search space during the refinement process, it loses to the multiscale routing in terms of completion rate. This trend holds true especially in designs with many global nets, such as Mcc1 and Mcc2. This result indicates that the multiscale method can generate planning results with better quality.

## 5 Conclusion

Multiscale algorithms have captured a large part of the state of the art in VLSI physical design. However, experiments on synthetic benchmarks suggest that a significant gap between attainable and optimal still exists, and other kinds of algorithms remain competitive in most areas other than partitioning. Therefore, a deeper understanding of the underlying principles most relevant to multiscale optimization in physical design is widely sought. Current efforts to improve multiscale methods for placement and routing include attempts to answer the following questions.

**Table 6.** Comparison of 3-level and multiscale routing.

Circuit	3-level Routing		Multiscale Routing	
	#Failed Nets (#Total sub-nets)	Run-time(s)	#Failed Nets	Run-time(s)
S5378	517(3124)	430.2	0	30
S9234	307(2774)	355.2	0	22.8
S13207	877(6995)	1099.5	0	85.2
S15850	978(8321)	1469.1	0	107.1
S38417	1945(21035)	3560.9	0	250.9
S38584	2535(28177)	7086.5	0	466.1
Struct	21 (3551)	406.2	0	31.6
Primary1	19 (2037)	239.1	0	33.5
Primary2	88 (8197)	1311	0	162.7
Mcc1	195 (1694)	933.2	0	105.9
Mcc2	2090 (7541)	12333.6	0	1916.9
Avg.		11.7		1

**Table 7.** Comparison of hierarchical routing and multiscale routing.

Circuit	Hierarchical Routing		Multiscale Routing	
	#Failed Nets (#Total sub-nets)	Run-time(s)	#Failed Nets	Run-time(s)
S5378	1(3390)	22.6	0	30
S9234	0	15.7	0	22.8
S13207	1(7986)	60.9	0	85.2
S15850	1(9587)	75.8	0	107.1
S38417	0	223.2	0	250.9
S38584	3(31871)	334.6	0	466.1
Struct	0	21.6	0	31.6
Primary1	0	34.6	0	33.5
Primary2	0	164.8	0	162.7
Mcc1	377(13338)	205.3	0	105.9
Mcc2	7409(96030)	4433.0	0	1916.9
Avg.		1.04		1

How should an optimization hierarchy be defined? Is it important to maintain a hypergraph model at coarser levels, or will a graph model suffice? Can relaxation itself be used as a means of selecting coarse-level variables [BR02]? How should the accuracy and quality of hypergraph coarsenings and coarse-level data functions and placements be quantified? Is it possible to simultaneously employ multiple hierarchies constructed by different means and perhaps targeting different objectives? Can optimization on the dual hypergraph hierarchy be coordinated with optimization on the primal? Given multiple candidate solutions at a given level, is there a general prescription for combining them into a single, superior solution?

Current approaches directly optimize a coarse-level formulation of the fine-level problem. Would it be more effective at the coarser level to focus instead on the error or displacement in the given, adjacent finer-level configuration?

Which methods of relaxation are the most effective at each level? What should be the relative roles of continuous and discrete formulations? How should they be combined? At what point in the flow should constraints be strictly satisfied and subsequently enforced? Are relaxations best restricted to sequences of small subsets? How should these subsets be selected? When should all variables be updated at every step?

Which is preferable: a small number of V-cycles with relatively expensive relaxations, or a large number of V-cycles (or other hierarchy traversal) with relatively inexpensive relaxations? Is there some underlying convergence theory for multiscale optimization, either to local or global solutions, that can be used to guide the practical design of the coarsening, relaxation, interpolation, and iteration flow?

As the final, physical barriers to the continuation of Moore's Law in fabrication technology emerge, these questions pose both a challenge and an opportunity for significant advances in VLSI physical design.

## 6 Acknowledgments

Funding for this work comes from Semiconductor Research Consortium Contracts 2003-TJ-1091 and 99-TJ-686 and National Science Foundation Grants CCR-0096383 and CCF-0430077. The authors thank these agencies for their support.

## References

- [AHK97] C. Alpert, J.-H. Huang, and A.B. Kahng. Multilevel circuit partitioning. In *Proc. 34th IEEE/ACM Design Automation Conf.*, 1997.
- [AK96] C. Alpert and A. Kahng. A hybrid multilevel/genetic approach for circuit partitioning. In *In Proceedings of the Fifth ACM/SIGDA Physical Design Workshop*, pages 100–105, 1996.
- [Ake67] S.B. Akers. A modification of Lee’s path connection algorithm. *IEEE Trans. on Computers*, EC-16:97–98, Feb. 1967.
- [Alb00] Christoph Albrecht. Provably good global routing by a new approximation algorithm for multicommodity flow. In *Proc. International Symposium on Physical Design*, pages 19–25, Mar. 2000.
- [Alp98] C.J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. Intl Symposium on Physical Design*, pages 80–85, 1998.
- [BHM00] W.L. Briggs, V.E. Henson, and S.F. McCormick. *A Multigrid Tutorial*. SIAM, Philadelphia, second edition, 2000.
- [BP83] M. Burstein and R. Pelavin. Hierarchical channel router. *Proc. of 20th Design Automation Conference*, pages 519–597, 1983.
- [BR97] V. Betz and J. Rose. VPR: A new packing, placement, and routing tool for FPGA research. In *Proc. Intl. Workshop on FPL*, pages 213–222, 1997.
- [BR02] A. Brandt and D. Ron. *Multigrid Solvers and Multilevel Optimization Strategies*, chapter 1 of *Multilevel Optimization and VLSICAD*. Kluwer Academic Publishers, Boston, 2002.
- [BR03] U. Brenner and A. Rohe. An effective congestion-driven placement framework. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(4):387–394, April 2003.
- [Bra77] A. Brandt. Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation*, 31(138):333–390, 1977.
- [Bra86] A. Brandt. Algebraic multigrid theory: The symmetric case. *Appl. Math. Comp.*, 19:23–56, 1986.
- [Bra01] A. Brandt. Multiscale scientific computation: Review 2001. In T. Barth, R. Haimes, and T. Chan, editors, *Multiscale and Multiresolution Methods*. Springer Verlag, 2001.
- [Bre77] M.A. Breuer. Min-cut placement. *J. Design Automation and Fault Tolerant Comp.*, 1(4):343–362, Oct 1977.
- [Cad99] Cadence Design Systems Inc. Envisia ultra placer reference. In <http://www.cadence.com>, QPlace version 5.1.55, compiled on 10/25/1999.
- [CAM00] A.E. Caldwell, A.B.Kahng, and I.L. Markov. Improved algorithms for hypergraph partitioning. In *Proc. IEEE/ACM Asia South Pacific Design Automation Conf.*, 2000.
- [CC00] C. Chang and J. Cong. Pseudo pin assignment with crosstalk noise control. In *Proc. International Symposium on Physical Design*, Apr 2000.
- [CCK<sup>+</sup>03] T.F. Chan, J. Cong, T. Kong, J. Shinnerl, and K. Sze. An enhanced multilevel algorithm for circuit placement. In *Proc. IEEE International Conference on Computer Aided Design*, San Jose, CA, Nov 2003.
- [CCKS00] T.F. Chan, J. Cong, T. Kong, and J. Shinnerl. Multilevel optimization for large-scale circuit placement. In *Proc. IEEE International Conference on Computer Aided Design*, pages 171–176, San Jose, CA, Nov 2000.

- [CCKS03] T.F. Chan, J. Cong, T. Kong, and J. Shinnerl. *Multilevel Circuit Placement*, chapter 4 of *Multilevel Optimization in VLSICAD*. Kluwer Academic Publishers, Boston, 2003.
- [CCPY02] C.C. Chang, J. Cong, Z. Pan, and X. Yuan. Physical hierarchy generation with routing congestion control. In *Proc. ACM International Symposium on Physical Design*, pages 36–41, San Diego, CA, Apr 2002.
- [CCRX04] C. Chang, J. Cong, M. Romesis, and M. Xie. Optimality and scalability study of existing placement algorithms. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 537–549, 2004.
- [CCS05] T.F. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. Int'l Symp. on Phys. Design*, pages 185–192, 2005.
- [CCX03a] C-C. Chang, J. Cong, and M. Xie. Optimality and scalability study of existing placement algorithms. In *Proc. Asia South Pacific Design Automation Conference*, pages 621–627, 2003.
- [CCX03b] C.C. Chang, J. Cong, and M. Xie. Optimality and scalability study of existing placement algorithms. In *Asia South Pacific Design Automation Conference*, pages 325–330, Kitakyushu, Japan, Jan 2003.
- [CFK99] J. Cong, J. Fang, and K.Y. Khoo. An implicit connection graph maze routing algorithm for ECO routing. In *Proc. International Conference on Computer Aided Design*, pages 163–167, Nov. 1999.
- [CFK00] J. Cong, J. Fang, and K.Y. Khoo. DUNE: A multi-layer gridless routing system with wire planning. In *Proc. International Symposium on Physical Design*, pages 12–18, Apr. 2000.
- [CFZ01] J. Cong, J. Fang, and Y. Zhang. Multilevel approach to full-chip gridless routing. *Proc. IEEE International Conference on Computer Aided Design*, pages 396–403, 2001.
- [CHKM96] J. Cong, Lei He, C.-K. Koh, and P. Madden. Performance optimization of VLSI interconnect layout. *Integration, the VLSI Journal*, 21(1-2):1–94, 1996.
- [CKL99] J. Cong, A.B. Kahng, and K.S. Leung. Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design. *IEEE Trans. on Computer-Aided Design*, 17(1):24–39, Jan. 1999.
- [CKM00] A.E. Caldwell, A.B. Kahng, and I.L. Markov. Can recursive bisection produce routable placements? In *Proc. 37th IEEE/ACM Design Automation Conf.*, pages 477–482, 2000.
- [CL00] J. Cong and S.K. Lim. Edge separability based circuit clustering with application to circuit partitioning. In *Asia South Pacific Design Automation Conference, Yokohama Japan*, pages 429–434, 2000.
- [CL04] Y. Chang and S. Lin. Mr: A new framework for multilevel full-chip routing. *IEEE Trans. on Computer Aided Design*, 23(5), May 2004.
- [CLC96] R.C. Carden, J. Li, and C.K. Cheng. A global router with a theoretical bound on the optimal solution. *IEEE Trans. Computer-Aided Design*, 15(2):208–216, Feb. 1996.
- [CLW99] J. Cong, H. Li, and C. Wu. Simultaneous circuit partitioning/clustering with retiming for performance optimization. *Proc. 36th ACM/IEEE Design Automation Conf.*, pages 460–465, Jun 1999.

- [CLW00] J. Cong, S.K. Lim, and C. Wu. Performance-driven multi-level and multiway partitioning with retiming. In *Proceedings of Design Automation Conference*, pages 274–279, Los Angeles, California, Jun 2000.
- [CM98] J. Cong and P. Madden. Performance driven multi-layer general area routing for PCB/MCM designs. In *Proc. 35th Design Automation Conference*, pages 356–361, Jun 1998.
- [CP68] H.R. Charney and D.L. Plato. Efficient partitioning of components. In *In Proc. of the 5th Annual Design Automation Workshop*, pages 16–0 – 16–21, 1968.
- [CS93] J. Cong and M. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi designs. In *Proc. Design Automation Conference*, pages 755–760, San Jose, CA, 1993.
- [CS03] J. Cong and J.R. Shinnerl, editors. *Multilevel Optimization in VLSICAD*. Kluwer Academic Publishers, Boston, 2003.
- [CW02] J. Cong and C. Wu. Global clustering-based performance-driven circuit partitioning. In *Proc. Int. Symp. on Physical Design*, pages 149–154, 2002.
- [CXWS03] B. Choi, H. Xu, M. Wang, and M. Sarrafzadeh. Flow-based cell moving algorithm for desired cell distribution. *Proc. IEEE International Conference on Computer Design*, pages 218–225, Oct 2003.
- [CXZ02] J. Cong, M. Xie, and Y. Zhang. An enhanced multilevel routing system. *IEEE International Conference on Computer Aided Design*, pages 51–58, 2002.
- [CY00] J. Cong and X. Yuan. Routing tree construction under fixed buffer locations. In *Proc. 37th Design Automation Conference*, pages 379–384, Jun. 2000.
- [DD97] Shantanu Dutt and Wenyong Deng. Vlsi circuit partitioning by cluster-removal using iterative improvement techniques. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 194 – 200, 1997.
- [Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [DiM94] G. DiMicheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.
- [Don88] W. E. Donath. Logic partitioning. *Physical Design Automation in VLSI systems*, 1988.
- [EJ98] H. Eisenmann and F.M. Johannes. Generic global placement and floorplanning. In *Proc. 35th ACM/IEEE Design Automation Conference*, pages 269–274, 1998.
- [FM82] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. Design Automation Conference*, pages 175–181, 1982.
- [GK98] N. Garg and J. Konemann. Faster and simpler algorithms for multi-commodity flow and other fractional packing problems. In *Proc. Annual Symposium on Foundations of Computer Science*, pages 300–309, Nov. 1998.
- [GMW81] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, London and New York, 1981. ISBN 0-12-283952-8.
- [Goe03a] R. Goering. FPGA placement performs poorly, study says. *EE Times*, 2003. <http://www.eedesign.com/story/OEG20031113S0048>.

- [Goe03b] R. Goering. IC placement benchmarks needed, researchers say. *EE Times*, 2003. <http://www.eedesign.com/story/OEG20030410S0029>.
- [Goe03c] R. Goering. Placement tools criticized for hampering IC designs. *EE Times*, 2003. <http://www.eedesign.com/story/OEG20030205S0014>.
- [Got81] S. Goto. An efficient algorithm for the two-dimensional placement problem in electrical circuit layout. *IEEE Trans. on Circuits and Systems*, 28(1):12–18, January 1981.
- [Had75] F. Hadlock. Finding a maximum out of a planar graph in polynomial time. *SIAM Journal of Computing*, 4(3):221–225, Sep. 1975.
- [Hig69] D.W. Hightower. A solution to line routing problems on the continuous plane. In *Proc. IEEE 6th Design Automation Workshop*, pages 1–24, 1969.
- [Hil02] D. Hill. Method and system for high speed detailed placement of cells within an integrated circuit design. In *US Patent 6370673*, Apr 2002.
- [HK72] M. Hannan and J.M. Kurtzberg. A review of the placement and quadratic assignment problems. *SIMA*, 14, 1972.
- [HL91] J. Heisterman and T. Lengauer. The efficient solution of integer programs for hierarchical global routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(6):748–753, Jun. 1991.
- [HL99] S.-W. Hur and J. Lillis. 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, Jun 1999.
- [HL00] S.-W. Hur and J. Lillis. Mongrel: Hybrid techniques for standard-cell placement. In *Proc. IEEE International Conference on Computer Aided Design*, pages 165–170, San Jose, CA, Nov 2000.
- [HMS03a] B. Hu and M. Marek-Sadowska. Fine granularity clustering for large-scale placement problems. In *Proc. Int'l Symp. on Physical Design*, Apr. 2003.
- [HMS03b] B. Hu and M. Marek-Sadowska. Wire length prediction based clustering and its application in placement. In *Proc. Design Automation Conference*, Jun. 2003.
- [HMS04] B. Hu and M. Marek-Sadowska. Fine granularity clustering based placement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Apr. 2004.
- [HT95] M. Hayashi and S. Tsukiyama. A hybrid hierarchical approach for multi-layer global routing. *Proceedings of the 1995 European conference on Design and Test*, pages 492–496, Mar. 1995.
- [itr] *International Technology Roadmap for Semiconductors*. <http://public.itrs.net/> .
- [JCX03] M. Romesis J. Cong and M. Xie. Optimality, scalability and stability study of partitioning and placement algorithms. In *Proc. International Symposium on Physical Design*, 2003.
- [KAKS97] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 526–529, 1997.
- [Kar99] G. Karypis. Multilevel algorithms for multi-constraint hypergraph partitioning. Technical Report 99-034, Department of Computer Science, University of Minnesota, Minneapolis, 1999.

- [Kar02] G. Karypis. *Multilevel Hypergraph Partitioning*, chapter 3 of *Multilevel Optimization and VLSICAD*. Kluwer Academic Publishers, Boston, 2002.
- [KLR<sup>+</sup>87] R.M. Karp, F.T. Leighton, R.L. Rivest, C.D. Thompson, U.V. Vazirani, and V. V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 2:113–129, 1987.
- [KSJA91] J.M. Kleinhans, G. Sigl, F.M. Johannes, and K.J. Antreich. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. on Computer-Aided Design*, 10:356–365, 1991.
- [KW04] A.B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. In *Proc. Int’l Symp. on Physical Design*, pages 18–25, 2004.
- [LHT90] Y. Lin, Y. Hsu, and F. Tsai. Hybrid routing. *IEEE Transactions on Computer-Aided Design*, 9(2):151–157, Feb. 1990.
- [LLC95] J. Li, J. Lillis, and C. Cheng. Linear decomposition algorithm for vlsi design applications. In *Proc. Int’l Conf. on Computer-Aided Design*, pages 223–228, 1995.
- [LTKS02] J. Lou, S. Thakur, S. Krishnamoorthy, and H. Sheng. Estimating routing congestion using probabilistic analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(1):32–41, January 2002.
- [ME95] L. McMurchie and C. Ebeling. Pathfinder: a negotiation-based performance-driven router for FPGAs. In *Proc. of ACM Symposium on Field-Programmable Gate Array*, pages 111–117, Feb. 1995.
- [MT68] K. Mikami and K. Tabuchi. A computer program for optimal routing of printed circuit connectors. *IFIPS Proc*, H-47:1475–1478, 1968.
- [Nai87] R. Nair. A simple yet effective technique for global wiring. *IEEE Trans. on Computer-Aided Design*, 6(2), 1987.
- [NS96] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw Hill, New York, 1996.
- [QB79] N. Quinn and M. Breuer. A force-directed component placement procedure for printed circuit boards. *IEEE Trans. on Circuits and Systems CAS*, CAS-26:377–388, 1979.
- [RDJ94] Bernhard M. Riess, Konrad Doll, and Frank M. Johannes. Partitioning very large circuits using analytical placement techniques. In *Proc. Design Automation Conference*, pages 646 – 651, 1994.
- [SDJ91] G. Sigl, K. Doll, and F.M. Johannes. Analytical placement: A linear or a quadratic objective function? In *Proc. 28th ACM/IEEE Design Automation Conference*, pages 427–432, 1991.
- [She99] Naveed Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Boston, Dordrecht, London, third edition, 1999.
- [Sou78] J. Soukup. Fast maze router. In *Proc. 15th Design Automation Conference*, pages 100–102, 1978.
- [SR99] Y. Sankar and J. Rose. Trading quality for compile time: Ultra-fast placement for FPGAs. In *FPGA ‘99, ACM Symp. on FPGAs*, pages 157–166, 1999.
- [SS95] W.-J. Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Trans. on Computer-Aided Design*, pages 349–359, Mar 1995.
- [SWY02] M. Sarrafzadeh, M. Wang, and X. Yang. *Modern Placement Techniques*. Kluwer Academic Publishers, Boston, 2002.

- [TOS00] U. Trottenberg, C.W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, London, 2000.
- [Vyg97] Jens Vygen. Algorithms for large-scale flat placement. In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 746–751, 1997.
- [WK97] Dongsheng Wang and E.S Kuh. A new timing-driven multilayer mcm/ic routing algorithm. In *Proc. IEEE Multi-Chip Module Conference*, pages 89–94, Feb. 1997.
- [WYS00a] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proc. International Conference on Computer-Aided Design*, pages 260–264, 2000.
- [WYS00b] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large circuits. *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 260–263, Apr 2000.
- [XWCS03] H. Xu, M. Wang, B. Choi, and M. Sarrafzadeh. A trade-off oriented placement tool. *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 467–471, Apr 2003.
- [YM01] M.C. Yildiz and P.H. Madden. Improved cut sequences for partitioning-based placement. In *Proc. Design Automation Conference*, pages 776–779, 2001.