Large-Scale Circuit Placement

JASON CONG, JOSEPH R. SHINNERL, and MIN XIE
UCLA Computer Science
TIM KONG
Magma Design Automation
and
XIN YUAN
IBM Corporation, Microelectronics Division

Placement is one of the most important steps in the RTL-to-GDSII synthesis process, as it directly defines the interconnects, which have become the bottleneck in circuit and system performance in deep submicron technologies. The placement problem has been studied extensively in the past 30 years. However, recent studies show that existing placement solutions are surprisingly far from optimal. The first part of this tutorial summarizes results from recent optimality and scalability studies of existing placement tools. These studies show that the results of leading placement tools from both industry and academia may be up to 50% to 150% away from optimal in total wirelength. If such a gap can be closed, the corresponding performance improvement will be equivalent to several technology-generation advancements. The second part of the tutorial highlights the recent progress on large-scale circuit placement, including techniques for wirelength minimization, routability optimization, and performance optimization.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and routing*; G.4 [**Mathematical Software**]: *Algorithm design and analysis*; J.6 [**Computer-Aided Engineering**]: *Computer-aided design (CAD)*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Placement, optimality, scalability, large-scale optimization

1. INTRODUCTION

The exponential growth of on-chip complexity has dramatically increased the demand for scalable optimization algorithms for large-scale physical design.

This work was funded by Semiconductor Research Consortium, Contracts 2003-TJ-1091 and 99-TJ-686; National Science Foundation, Grants CCR-0096383 and CCF-0430077; and Magma Corporation and Xilinx Corporation under the California MICRO Program.
Authors' addresses: J. Cong, J. R. Shinnerl, and M. Xie, UCLA Computer Science Department, Campus Mailcode 159610, Los Angeles, CA 90095-1596; email: {cong,shinnerl,xie}@ca.ucla.edu; T. Kong, Magma Design Automation, 12100 Wishire Blvd., Suit, 480, Los Angeles, CA 90025; email kongtm@magma-da.com; Xin Yuan, IBM Corporation, Microelectronics Division, 1000 River Street, Mail Stop 862F, Essex Junction, VT 05452; email: xinyuan@us.ibm.com.

Although complex logic functions can be composed in a hierarchical fashion following the logical hierarchy, recent studies [Cong 2001] show the importance of building a good physical hierarchy from a flattened or nearly flattened logical netlist for performance optimization. Because a logical hierarchy is usually conceived with little or no consideration of the layout and interconnect information, it may not map well to a two-dimensional layout solution. Therefore, large-scale global placement on a nearly flattened netlist is needed for physical hierarchy generation to achieve the best performance. This approach is even more important in today's nanometer designs, where the interconnect has become the performance bottleneck.

This tutorial highlights state-of-the-art placement optimization techniques. Section 2 presents recent studies on the quality and scalability of existing placement algorithms on a set of benchmarks with known optimal solutions. Section 3 reviews scalable paradigms for large-scale wirelength minimization. Timing optimization and routability optimization are discussed in Sections 4 and 5, respectively. Conclusions are given in Section 6.

2. GAP ANALYSIS OF EXISTING PLACEMENT ALGORITHMS

Placement algorithms have been actively studied for the past 30 years. However, there is little understanding of how far 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 using cleverly constructed placement examples with known optimal wirelength [Hagen et al. 1995; Chang et al. 2003b]. In this section, we summarize the results of these studies.

2.1 Placement Examples with Known Optima

Recently, four suites of placement examples with known optimal wirelength (PEKO) were constructed [Hagen et al. 1995; Chang et al. 2003b]. The construction method takes as input an integer n and a *net-profile* vector of integers D. It then generates a placement example $\mathcal{P}$ with n placeable modules such that (i) the number of nets of degree i equals $D(i)$, and (ii) $\mathcal{P}$ has a known globally optimal half-perimeter wirelength. The values of n and D used to construct PEKO either were directly extracted from the netlists of the ISPD98 suite originally from IBM [Alpert 1998] or were taken as those values scaled by a factor of 10. The PEKO suite is given in both GSRC BookShelf format and LEF/DEF format and is available online [Cong et al. 2004].

All the nets in PEKO are local, that is, the wirelength of every net has the minimum possible value. However, in real circuits, there may also be global connections that span a significant portion of the chip, even when they are optimally placed. Additional benchmark circuits were therefore constructed to study the impact of global nets [Cong et al. 2003b]. Circuits in the G-PEKU suite consist only of global nets connecting either an entire row or an entire column. For such circuits, an obvious upper bound on optimal wirelength is the sum of the lengths of the rows and columns. Circuits in the PEKU suite (Placement

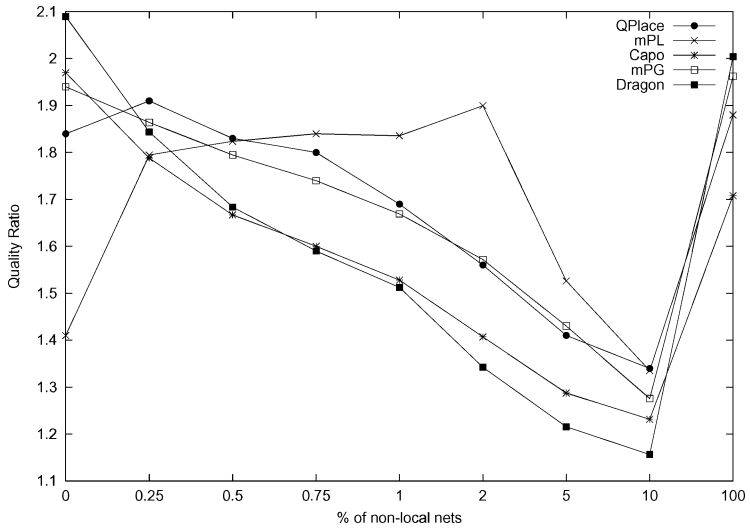


Fig. 1. Average solution quality vs percentage of non-local nets, from PEKO (0% non-local nets) through PEKU (0.25% to 10% of nonlocal nets) to G-PEKU (100% non-local nets). Each data point is an average quality ratio for a given placer over all circuits in the given suite.

Examples with Known Upper bounds on wirelength) consist of both PEKO-style local nets and additional, randomly generated nonlocal nets. An upper bound on the optimal wirelength is derived simply by adding the wirelengths of nonlocal nets to the known total wirelength of the local nets. In the study [Cong et al. 2003b], the percentage of nonlocal nets was gradually increased from 0.25% to 10%. The G-PEKU and PEKU suites are also available online [Cong et al. 2004].

2.2 Gap Analysis Results

Four state-of-the-art placers from academia and one industrial placer were studied for optimality and scalability: Dragon v.2.20 [Wang et al. 2000], Capo v.8.5 [Caldwell et al. 2000b], mPL v.2.0 [Chan et al. 2003b], mPG v.1.0 [Chang et al. 2003a], and QPlace v.5.1.55 [Cadence Design Systems, Inc. 1999]. Experiments with Dragon, mPL, mPG and QPlace were performed on a SUN Blade 750 MHz running SunOs 5.8 with 4GB of memory. The experiments with Capo were performed on a Pentium IV 2.20GHz running RedHat 8.0 with 2GB of memory. To measure how close the placement results are to optimal, the ratio of a placement's wirelength to the optimal wirelength (on PEKO) or its upper bound (on G-PEKU and PEKU) was computed. This ratio is called the "quality ratio." An upper limit of 24 hours was placed on the run time; any process exceeding this limit was terminated.

The results are summarized in Figure 1 and Figure 2. Figure 1 shows how the average quality ratios of these tools change with the percentage of nonlocal nets. Figure 2 shows how the run times of these tools changes with increase in

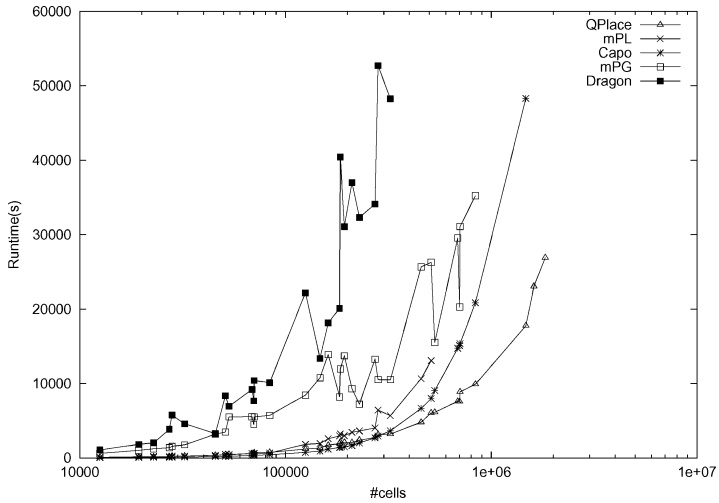


Fig. 2. Run time vs. cell number for several algorithms on the PEKO suite.

cell number. We make the following observations:

(i) None of the placers from the 2003 study achieves a quality ratio close to one.[1] On PEKO, the wirelengths produced by these tools range from 1.41 to 2.09 times the optimal on average (see Figure 1) and 1.66 to 2.50 times the optimal in the worst case (not shown). On G-PEKU, the gap between their solutions and the upper bound varies between 79% and 102% in the worst case. Some placers may try to improve routability by sacrificing wirelength. However, given the gap between their wirelengths and the optimal value, there remains significant room for improvement in existing placement algorithms.

(ii) The quality ratio from the same placer can vary significantly for designs of similar sizes but different characteristics. None of them produces consistently better results than another. On PEKO, mPL gives the shortest wirelength. However, its quality ratio shows an increase of more than 40% with a small increase of nonlocal nets. On G-PEKU, Capo gives the closest solution to the upper bound in most cases. On PEKU, Dragon's wirelength gradually becomes the closest to the upper bound. This seems to suggest that more scalable and stable hybrid techniques may be needed for future generations of placement tools.

(iii) Different placers displayed different scalability in run time and solution quality. None of them can successfully finish all the circuits of PEKO, because of either the run-time limit (e.g., Dragon), or memory consumption (e.g., Capo, mPL, mPG, QPlace). For those circuits they successfully placed, an average solution quality deterioration from 4% (on QPlace) to 25% (on mPL) can be observed when the problem size is increased by a factor of 10.

[1]However, recently improved versions of mPL and Feng Shui (not included in the 2003 study) have consistently obtained average quality ratios below 1.3, and other placers have also reported significant gains.

It is not known whether the gaps on real circuits are similar to those observed on the benchmarks discussed above. A recent study [Wang et al. 2005] computes lower bounds for the optimal half-perimeter wire lengths of some widely used FGPA benchmarks and observes ratio gaps between 1.14 and 4.09 for placements computed by VPR [Betz and Rose 1997]. The construction of placement examples that resemble real circuits more closely, including examples optimized for timing [Cong et al. 2003a] or routability, is an active area of research.

3. SCALABLE PARADIGMS

Scalability typically derives from some hierarchical form of computation. The use of hierarchy may be subtle or indirect but is rarely completely absent. In this article, we use the word "scalability" in the practical, operational sense and therefore consider not just $\mathcal{O}(N)$ algorithms but rather any framework likely to have applicability lasting for several technology generations and circuit-size ranges.

Wirelength, performance, power consumption, and routability are the typical objectives of VLSI placement. Of these, weighted total wirelength is a useful single representative, as (i) it can be optimized efficiently, and (ii) strategic, iterative net reweighting can be used to optimize other objectives, such as performance and routability.

Our discussion is centered on methods for wirelength-driven *global placement*. The goal here is only an approximately uniform distribution of cells with as little total wirelength as possible. The problem of transforming a global placement to an overlap-free configuration is left to the *detailed* placement phase.

The most promising large-scale approaches to wirelength-driven global placement can be broadly categorized by (i) the manner in which their hierarchies are constructed and traversed, (ii) the kinds of intralevel optimizations used and the manner in which they are incorporated into the hierarchy and coordinated with each other. At the highest level, we classify algorithms as top-down, multilevel, or flat; in practice, however, these categories overlap in interesting ways. Top-down algorithms (Section 3.1) use variants of recursive partitioning. Multilevel approaches (Section 3.2) compute placements of aggregates at several distinct levels of aggregation. These levels are most commonly formed by recursive clustering but may be defined instead by top-down partitioning. Flat approaches (Section 3.3), if scalable, use hierarchy for internal iterative computation while maintaining a consistent non-hierarchical view of the placement problem.

3.1 Recursive Top-Down Partitioning

Among academic placement tools, all the leading top-down methods rely on variants of recursive circuit partitioning in some way. Seminal work on partitioning-based placement was done by Breuer [1977] and Dunlop and Kernighan [1985]. Most contemporary methods, including Capo [Caldwell et al. 2000b] and Feng Shui [Yildiz and Madden 2001a], have exploited further advances in fast algorithms for hypergraph partitioning to push these frameworks beyond their original capabilities. Fast, high-quality $\mathcal{O}(N)$ partitioning algorithms give

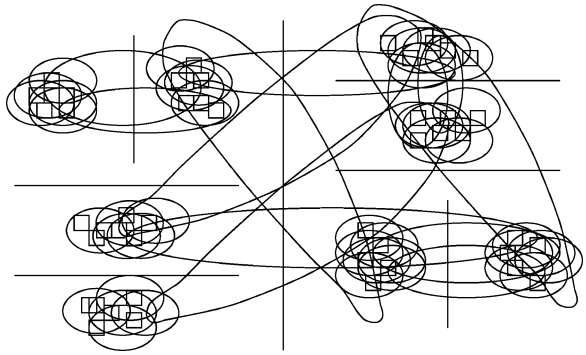


Fig. 3. Cutsize-driven partitioning-based placement. Rectangles represent movable cells, line segments represent cutlines, and ellipses and other closed curves represent nets. The recursive bipartitioning attempts to minimize the number of nets containing cells in more than one subregion.

top-down partitioning attractive $\mathcal{O}(N \log N)$ scalability overall. The asymptotic is $\mathcal{O}(N \log N)$ and not $\mathcal{O}(N)$, because partitioning is always applied to cells, not to aggregates.

3.1.1 *Cutsize Minimization.* Simple and traditional recursive bisection with a cutsize objective can be used quite effectively with simple Fiduccia-Matheysses-style iterations. At a given level, each region is considered separately from the others in some arbitrary order. A spatial cutline for the region, either horizontal or vertical, can be carefully chosen. Given some initial partition, subsets of cells are moved across the cutline in a way that reduces the total weight of hyperedges cut without violating a given area-balance constraint. This constraint can be set loosely initially and then gradually tightened. As the recursion proceeds, cell subsets become smaller, and the cell-area distribution over the placement region becomes more uniform. Base cases of the bipartitioning recursion are reached when cell subsets become small enough that special end-case placers can be applied [Caldwell et al. 2000d]. A small example is illustrated after 3 levels of bipartitioning in Figure 3.

Connections between subregions can be modeled by *terminal propagation* [Dunlop and Kernighan 1985], in which the usual cutsize objective is augmented by terms incorporating the effect of connections to external subregions. Other techniques for organizing local partitioning subproblems use Rent's rule to relate cutsize to wirelength estimation [Wang et al. 2000; Yildiz and Madden 2001b]. Careful consideration of the order and manner in which subregions are selected for partitioning can be significant. For example, a dynamic-programming approach to cutline selection can improve overall results by 5% or more [Yildiz and Madden 2001b]. In the *multiway partitioning* framework, intermediate results from the partitioning of each subregion are used to influence the final partitioning of others. Explicit use of multiway partitioning at each stage can in some cases bring the configuration closer to a global optimum than is possible by recursive bisection alone [Yildiz and Madden 2001a]. Cell replication and iterative deletion have been used for this purpose. Rather than attempt

to find the best subregion in which to place a cell, we can replicate the cell enough times to place it in once in every subregion, then iteratively delete only the worst choices. These iterations may continue until only one choice remains, or they may be terminated earlier, allowing a small pool of candidates to be propagated to and replicated at finer levels, By postponing further deletion decisions until better information becomes available, spurious effects from locally optimal subregion partitions can be diminished and the global result improved.

Example: CAPO. To provide a concrete example of an implementation of fixed-die placement by top-down, cutsize-driven recursive bipartitioning, we briefly describe the CAPO package [Caldwell et al. 2000b] and a few of its extensions. The stated goals of CAPO are simplicity and automatic routability. In addition, the top-down flow and the use of leading multilevel hypergraph partitioner MLpart [Caldwell et al. 2000a] give CAPO superior speed and scalability. For simplicity, no explicit congestion management is used. Instead, decisions affecting the flow of the top-down recursive bipartitioning are carefully considered for their impact on routability.

In CAPO, recursive cutsize-driven hypergraph-netlist bipartitioning is enhanced to support its ultimate goal of wirelength-driven circuit placement. Key considerations include nonuniformity of vertex weights, assignment of partition blocks to rectangular placement subregions, efficient solution of partitioning subproblems with small balance tolerances, and connections between cells in the subregion currently being partitioned and other, "external" subregions (terminal propagation). Movable cells in the circuit correspond to vertices in the partitioning instance. Vertex weights are determined by corresponding cell areas.

Given a hypergraph netlist and a placement region or subregion, the multilevel partitioner MLpart [Caldwell et al. 2000a] is used to divide the set of movable cells into two subsets of nearly equal total areas, when the number of cells exceeds 200. For fewer than 200 cells, enhanced Fiduccia-Matheysses (FM) partitioning [Fiduccia and Mattheyses 1982; Caldwell et al. 2000c] is used directly (MLpart is also based on this enhanced version of FM). The region must then be split, either vertically or horizontally, so that the resulting subregions hold the partition blocks, and whitespace is distributed as evenly as possible. CAPO uses a horizontal cut if the number of standard-cell rows contained in the subregion exceeds $M/15$, where M is the number of movable cells in the subregion. Otherwise, it chooses cut direction in order to make the aspect ratios of the resulting subregions as close to one as possible. For a vertical cut, whitespace can be distributed perfectly evenly, but horizontal cuts are constrained to lie between uniform-height rows of the standard-cell layout. Respecting standard cell row boundaries in this fashion greatly facilitates legalization of the final global placement. However, a recent study shows that this restriction occasionally overconstrains end cases and increases wirelength [Agnihotri et al. 2003]. The authors of this study show that Feng Shui's "fractional-cut" relaxation of row boundaries during the partitioning can considerably improve results, when it is followed by careful displacement-minimizing legalization, such as dynamic-programming based row-assignment.

Once the number of cells to be placed in any subregion decreases below 35, CAPO employs time-limited branch-and-bound heuristics to obtain an optimal or nearly optimal end-case solution [Caldwell et al. 2000d]. Finally, greedy refinement of cell orientations further improves wirelength and routability.

Much of CAPO's performance derives from its placement-driven enhancements to its core FM partitioner. These enhancements are concerned mainly with (i) avoiding the "corking effect," as described below, caused by improper handling of large variations in movable cell areas and/or tight area-balance constraints, and (ii) terminal propagation.

Given any initial partition, FM considers sequences of single, maximum-gain cell moves from one partition block to the other. It maintains a list of "buckets" for each partition block, where the kth bucket in each list holds the vertices which, when moved to the opposite block, will reduce the total number of nets cut by k. However, a cell will not be moved if the move violates the vertex-weight balance constraint. The original version of FM [Fiduccia and Mattheyses 1982] is focused primarily on hypergraph instances with all vertex weights equal. It does not specify any ordering for vertex weights are not ordered within buckets. According to the partitioning studies on which Capo is based [Caldwell et al. 2000c], many leading implementations of FM-based partitioning, in order to reduce run time, simply terminate searches in gain buckets when the first vertex in the kth bucket cannot be moved without violating the vertex-weight balance constraint. This short-cut has dire consequences, however, when cell sizes vary widely. If a cell too large to be moved occurs at the front of a bucket, it prevents any of the other moves in the bucket from being examined.

By avoiding this "corking" effect, CAPO significantly improves its results [Hagen et al. 1997]. CAPO prevents corking in two different ways. First, gain buckets are maintained as last-in, first-out (LIFO) stacks. Second and most importantly, it starts each bipartitioning subproblem with a relaxed area-balance constraint and gradually tightens the constraint as partitioning iterations proceed. Initially, the maximum allowed area imbalance is set to 20% of total area, or three times the area of the largest movable cell, whichever is greater. As the balance tolerance decreases below the area of any cell, that cell is locked in its current partition block. The final subproblem balance tolerance is selected so that, given an initial whitespace budget, enough relative whitespace in endcase subproblems is ensured that overlap-free configurations can typically be found.

In CAPO's original implementation, terminal propagation is simple. A given subproblem consists of (i) a pair of adjacent rectangular subregions, say, A and B, (ii) a collection of movable cells S to be divided between A and B, and (iii) a list of nets, each of which contains cells in S and, possibly, other *external* cells in other subregions as well. A hyperedge which contains some external cells in subregions closer to A and also other external cells in subregions closer to B will necessarily be cut and is therefore ignored. A hyperedge whose external cells (those not in S) are all closer to A than to B is treated as if its external cells are all fixed at A's center. Similarly, a hyperedge whose external cells are all closer to B than to A is treated as if its external cells are all fixed at B's center. Although this simple strategy is effective, recent studies [Yildiz and

Madden 2001a; Kahng and Reda 2004] demonstrate significant improvement by iterative refinement. Initially many nets contain *ambiguous* terminals, that is, external cells whose locations are not yet known precisely enough to determine which of subregions A and B they are closer to. However, after a complete sweep of bipartitioning, many of these ambiguous terminals become unambiguous relative to given subproblems. Repartitioning such subproblems with the improved terminal propagation leads to improved average results—for example, 5% cutsize reduction after 3 complete sweeps at each level [Kahng and Reda 2004].

3.1.2 *Incorporating Advances in Floorplanning.* Recent improvements to Capo include the incorporation of fixed outline floorplanning to improve handling of large macro blocks in mixed-size placement [Adya et al. 2004]. Min-cut placement proceeds as described above until certain ad-hoc tests suggest that legalization of a subset of blocks and cells within their assigned subregion may be difficult. At that point, the cells in that subregion are aggregated into soft clusters, and annealing-based fixed outline floorplanning is applied to the given subproblem [Adya and Markov 2002]. If it succeeds, the macro locations in its solution are fixed. If it fails, it must be merged with its sibling subproblem, and the merged parent subproblem must then be floorplanned. This step therefore initiates a recursive backtracking through ever larger ancestor subproblems. The backtracking terminates as soon as one of these ancestor subproblems is successfully floorplanned. The ad-hoc tests are chosen to prevent long backtracking sequences on most test cases, as the floorplanner does not scale well to large subproblems. Adya et al. [2004] observe that it is typically possible to define the ad-hoc tests so that the transition from min-cut partitioning to fixed-outline floorplanning does not impair scalability. However, as the algorithm cannot ensure the legalizability of the subproblems it generates by min-cut partitioning, it cannot prevent the possibility of a long backtracking sequence or a failure, especially on difficult low-whitespace instances.

The general challenge of ensuring the legalizability of subproblems within a min-cut-partitioning-based floorplanning or placement has been addressed by Patoma [Cong et al. 2005a, 2005b]. Beginning with the given instance itself, Patoma employs fast and scalable area-driven floorplanning before cutsize-driven partitioning in order to confirm that the problem can be legalized as given. This area-driven "pre-legalization" ignores wirelength but serves as a guarantor of the legalizability of subsequent steps. It is extremely robust; no failure on any known public-domain benchmark circuit has been observed. Given the guarantor legalization at a given level, cutsize-driven partitioning proceeds at that level. The flow then proceeds recursively on the subproblems generated by the cutsize-driven partitioning, each subproblem being legalized before it is solved. When prelegalization fails, the failed subproblem is merged with its sibling, and the previously computed legal guarantor solution to this parent subproblem is improved to reduce wirelength. The flow thus guarantees the computation of a legal placement or floorplan, under the very modest assumption that the initial attempt to prelegalize the given instance succeeds.

Experiments with this flow demonstrate significantly more robust performance on mixed-size benchmarks with white space between 1 and 10%.

3.1.3 *Partitions Guided by Analytical Placements.* An oft-cited disadvantage of recursive bisection is its alleged tendency to ignore the global objective as it pursues locally optimal partitions. Approximating wirelength by cutsize in the objective may also degrade the quality of the final placement. A radically different approach, first introduced in Proud [Cheng and Kuh 1984; Tsay et al. 1988] and subsequently refined by Gordian [Kleinhans et al. 1991], is to use continuous, iteratively-constrained quadratic star-model wirelength minimization over the entire circuit to guide partitioning decisions. The choice of a quadratic-wirelength objective helps avoid long wires and facilitates the construction of efficient numerical linear-system solvers for the optimality conditions, for example, preconditioned conjugate gradients. I/O pads prevent the cells from simply collapsing to a single point. Linear wirelength can still be asymptotically approximated by iterative adjustments to the net weights [Sigl et al. 1991]. Following this "analytical" placement, each region is then quadrisected, and cells are partitioned to subregions in order to further reduce overlap and area congestion. In Gordian, carefully chosen cutlines and FM-based cutsize-driven partitioning and repartitioning are used. Cell-to-subregion assignments are loosely enforced by imposing and maintaining a single center-of-mass equality constraint for each subregion. As constraints accumulate geometrically, degrees of freedom in cell movement are eliminated, and the quadratic minimization at each step moves cells less and less.

Example: BonnPlace. BonnPlace [Vygen 1997; Brenner and Rohe 2002] is the leading contemporary variation of placement by top-down recursive partitioning guided by analytical quadratic wirelength minimization. Global unconstrained minimization of quadratic wirelength (cf. (2) below) determines a starting configuration; the presence of fixed I/O pads, typically along the circuit boundary, prevents the movable cells from collapsing to a single point. The cells are then partitioned into four disjoint subregions, in linear time, in a manner that essentially minimizes the sum of their rectilinear displacements from their starting positions [Vygen 2000]. BonnPlace does not explicitly impose equality constraints into the subseqent analytical minimization to preserve these partitioning assignments, as Gordian does. Instead, it directly alters the quadratic-wirelength objective to minimize the sum of all cells' displacements from their assigned subregions. The following four steps are then repeated until subregions become small enough that legalization and detailed placement by local perturbations can be applied [Brenner et al. 2004].

(1) For each pair of connected cells (clique model) not in the same subregion, express their contributions to the global objective as squared Manhattan distances to their respective subregion boundaries, in the direction of the segment joining them.
(2) Minimize this quadratic objective over the *entire chip* simultaneously. (Don't minimize over subregions separately in sequence.) By moving all cells in all regions at the same time, a higher quality solution can be obtained.

(3) Quadrisect each subregion by displacement-minimizing partitioning.
(4) Suppose there are now $n^2 = n \times n$ subregions.
FOR EACH of the $(n-1)^2$ (overlapping) 2×2 windows of subregions,
(4.1) Perform quadratic WL minimization separately over just the cells in the current 2×2 window, allowing these cells to move anywhere within this window but not into other windows.
(4.2) IF the center of mass of the result of 4.1 can be maintained by some overlap-free placement of cells within the current window
THEN Repartition the cells in the current window into the four contained subregions.
ELSE Redo 4.1 subject to a center-of-mass equality constraint, then repartition.
(4.3) Do QP minimization over the current window with an updated objective respecting the partitioning assignments from 4.2.
END FOR

3.1.4 *Iterative Refinement.* Following the initial partitioning at a given level, various means of further improving the result at that level can be used. In BonnPlace (Section 3.1.3), unconstrained quadratic wirelength minimization over 2×2 windows of subregions is followed by a repartitioning of the cells in these windows. Windows can be selected based on routing-congestion estimates. Capo [Caldwell et al. 2000b] greedily selects cell orientations in order to reduce wirelength and improve routability. Feng Shui [Yildiz and Madden 2001a] follows k-way partitioning by localized repartitioning of each subregion. Some leading partitioning-based placers also employ time-limited branch-and-bound-based enumeration at the finest levels [Caldwell et al. 2000d].

In Dragon [Wang et al. 2000; Sarrafzadeh et al. 2002], an initial cutsize-minimizing quadrisection is followed by a bin-swapping-based refinement, in which entire partition blocks at that level are interchanged in an effort to reduce total wirelength. At all levels except the last, low-temperature simulated annealing is used; at the finest level, a more detailed and greedy strategy is employed. Because the refinement is performed on aggregates of cells rather than on cells from the original netlist, Dragon may also be grouped with the multilevel methods discussed next.

3.2 Multilevel Methods

Placement algorithms in the multilevel paradigm have only recently drawn attention [Sankar and Rose 1999; Chan et al. 2000; Chang et al. 2003a; Chan et al. 2003a, 2003b]. These methods are based on coarsening, relaxation, and interpolation, defined as follows.

(i) *Coarsening.* Hierarchies are built either from the bottom up by recursive aggregation or from the top down by recursive partitioning.
(ii) *Relaxation.* Localized optimizations are performed at every aggregation level.
(iii) *Interpolation.* Intermediate solutions are transferred from each aggregation level to its adjacent finer level.

Additionally, the order in which the various problems at the various levels are solved can be important. The simplest and most common approach is simply to proceed top down, from the coarsest to the finest level, once the aggregation hierarchy has been constructed [Sankar and Rose 1999; Chan et al. 2000; Sarrafzadeh et al. 2002; Kahng and Wang 2004]. When the hierarchy is defined by recursive bottom-up clustering, the combined flow of recursive clustering followed by recursive top-down optimization and interpolation is traditionally referred to as a "V-cycle" (Figure 5; the bottom of the 'V' corresponds to the coarsest or "top" level of the hierarchy). However, studies show considerable improvement is possible by repeated traversals and reconstructions of the hierarchy in various orderings [Brandt and Ron 2002; Chan et al. 2003b], as in traditional multiscale methods for PDEs [Briggs et al. 2000]. We refer to the organization of these traversals as *iteration flow*.

The scalability of the multilevel approach is straightforward to obtain and understand. Provided relaxation at each level has order linear in the number N_a of aggregates at that level, and the number of aggregates per level decreases by factor $r < 1$ at each level of coarsening, say $N_a(i) = r^i N$ at level i, the total order of a multilevel method is at most $cN(1+r+r^2+\cdots) = cN/(1-r)$. Higher-order (nonlinear) relaxations can still be used, if their use is limited to subsets of bounded size, e.g., by sweeps over overlapping windows of contiguous clusters at the current aggregation level.

3.2.1 *Coarsening.* A hierarchy of problem formulations can be defined either from the bottom up by recursive aggregation or from the top down by recursive aggregation. Traditional multiscale algorithms form their hierarchies by recursive clustering or generalizations thereof. However, the importance of limiting cutsize makes partitioning attractive in the placement context [Sarrafzadeh et al. 2002; Kahng and Wang 2004].

Typically, clustering algorithms merge tightly connected cells in a way that eliminates as many nets at the adjacent coarser level as possible while respecting some area-balance constraints. Experiments to date suggest that relatively simple, graph-based greedy strategies like First-Choice vertex matching [Karypis 1999, 2003] may be more effective than more sophisticated ideas like edge-separability clustering (ESC) [Cong and Lim 2000] that attempt to incorporate estimates of global connectivity information. How best to define coarse-level hyperedges without explosive growth in the number and degree of coarsened hyperedges relative to coarsened vertices remains an important open question [Hu and Marek-Sadowska 2004].

First-Choice clustering is the method currently used by mPL [Chan et al. 2000, 2003a, 2003b] and mPG [Chang et al. 2003a]. A graph is defined on the netlist vertices with each edge weighted by the "affinity" of the given two vertices. The affinity may represent some weighted combination of complex objectives, such as hypergraph connectivity, spatial proximity, timing delay, area balance, coarse-level hyperedge elimination, etc. Each vertex is paired with some other vertex for which it has its highest affinity. This maximum-affinity pairing is not symmetric and is independent of the order in which vertices

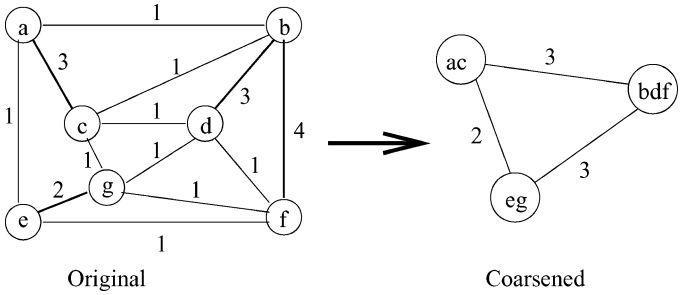


Fig. 4. First-Choice Clustering on an affinity graph. Darkened edges in the original graph are of maximal weight for at least one of their vertices. Note that vertex d has maximal affinity for vertex b, but vertex b has maximal affinity for vertex f.

are considered (see Figure 4). The corresponding maximum-affinity edges are marked and define a subgraph of the affinity graph; connected components of this subgraph are clustered and thus define vertices at the next coarser level.

A common objection to clustering is that its associations may be incorrect and therefore lead subsequent iterations to the wrong part of the solution space. To reduce the likelihood of poorly chosen clusters, the notion of a cluster can be generalized by *weighted aggregation*. Rather than assign each cell to just one cluster, we can break it into a small number of weighted fragments and assign the fragments to different coarse-level vertices; these are no longer simple clusters and are instead called aggregates. During interpolation, a cell's initial, inherited position is then typically determined by that of several aggregates as a weighted average [Chan et al. 2003b]. Clustering, also called strict aggregation, is a special case of weighted aggregation. Both are associated with *algebraic multigrid* (AMG) methods [Brandt 1986; Briggs et al. 2000] for the hierarchical, numerical solution of PDE's over unstructured discretizations.

3.2.2 *Initial Placement at Coarsest Level.* A placement at the coarsest aggregate level may be derived in various ways. Because the initial placement may have a large influence at subsequent iterations, and because the coarsest-level problem is relatively small, the placement at this level is typically performed with great care, to the highest quality possible. mPL [Chan et al. 2000, 2003a, 2003b] uses nonlinear programming (Section 3.2 below); mPG uses simulated annealing [Chang et al. 2003a]. How to judge the coarse-level placement quality is not necesssarily obvious, however, as the coarse-level objective may not correlate strictly with the ultimate fine-level objectives. For this reason, multiple iterations over the entire hierarchical flow are important [Brandt and Ron 2002; Chan et al. 2003b].

3.2.3 *Relaxations.* Relaxations at a given level are fast and relatively localized. The global view comes from the multilevel hierarchy, not from the intralevel relaxations. Almost any algorithm can be used, provided that it can support (i) incorporation of complex constraints (ii) restriction to subsets of movable objects. Relaxation in mPG and Ultrafast VPR is by fast annealing.

The mPG framework employs a fixed set of hierarchical bin-density constraints to monitor area and routing congestion. In mPL, relaxation at intermediate levels proceeds both by (i) quadratic wirelength minimization on small subsets followed by path-based area-congestion relief [Hur and Lillis 2000] and (ii) randomized, greedy, and discrete Goto-based cell swapping [Goto 1981].

3.2.4 *Interpolation.* Simple declustering and linear assignment can be effective [Chan et al. 2000]. With this approach, each component cluster is initially placed at the center of its parent's location. If an overlap-free configuration is needed, a uniform bin grid can be laid down, and clusters can be assigned to nearby bins or sets of bins. The complexity of this assignment can be reduced by first partitioning clusters into smaller windows, for example, of 500 clusters each. If clusters can be assumed to have uniform size, then fast linear assignment can be used. Otherwise, approximation heuristics are needed.

Under AMG-style weighted disaggregation, interpolation proceeds by weighted averaging: each finer-level cluster is initially placed at the weighted average of the positions of all coarser-level clusters with which its connection is sufficiently strong [Chan et al. 2003b]. Finer-level connections can also be used: once a finer-level cluster is placed, it can be treated as a fixed, coarser-level cluster for the purpose of placing subsequent finer-level clusters. Weighted disaggregation is described further in Section 3.2 below.

A constructive approach, as in Ultrafast VPR [Sankar and Rose 1999], can also lead to extremely fast and scalable algorithms. At each level, clusters are initially placed in the following sequence: (i) clusters directly connected to output pads, (ii) clusters directly connected to input pads, (iii) other clusters.

Example: mPL. To provide a concrete example of an implementation of placement by multilevel optimization, we briefly describe the mPL package [Chan et al. 2000, 2003a, 2003b]. mPL began as an attempt at scalable placement by efficient nonlinear-programming. Early experiments, however, showed that requiring descent at each iteration forced step sizes to be prohibitively small on problems with more than a few hundred variables. The complexity of the $\mathcal{O}(N^2)$ nonconvex nonoverlap constraints rendered pointwise approximations useful only in microscopic neighborhoods of their evaluation points. Multiscale optimization was used to overcome this complexity barrier. An early fast and scalable formulation was produced relatively easily, at some cost in wirelength compared to leading methods. Subsequent work has made mPL competitive in both run-time and quality of result, without loss of scalability.

Coarsening. mPL builds its hierarchy of problem scales by recursive first-choice clustering [Karypis 1999]. The mPL-FC affinity that vertex i has for vertex j is

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

where $w(e)$ is the weight assigned to hyperedge e, area(e) denotes the sum of the areas of the vertices in e, and $|e|$ denotes the number of vertices in hyperedge e, viz., the degree of e. Dividing by $|e|$ helps eliminate small hyperedges at coarser

levels, making coarse-level netlists sparser and hence easier to place [Karypis 1999; Sankar and Rose 1999; Hu and Marek-Sadowska 2003]. Dividing by area(e) gives greater affinity to smaller cells and thus helps keep cluster areas commensurate. A relatively uniform cluster-area distribution improves the performance of the nonlinear-programming and slot-assignment modules discussed below. Given a vertex i at the finer level, the vertex j assigned to it has 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 defined at the coarser level simply by replacing the elements of the finer-level hyperedge $e = \{e_1, \ldots, e_k\}$ by their corresponding clusters: $\bar{e} = \{c(e_1), \ldots, c(e_k)\}$, where duplicate clusters are of course removed. Hence, hyperedge degrees decrease during coarsening, and many hyperedges eventually become singletons at some level, where they are ignored.

Relaxation. A customized nonlinear-programming solver is used at mPL's coarsest level—500 cells or fewer, by default—to obtain an initial solution. Relaxation at all other levels is restricted to sweeps of local refinements on subsets. All relaxations are combined or alternated with techniques for spreading cells out to obtain a sufficiently uniform cell-area distribution at each level.

The nonlinear-programming (NLP) formulation employs simplified, smoothed objective and constraint functions. At the coarsest level, clusters v_i and v_j are modeled as disks. Let x and y denote vectors containing the respective x and y coordinates of all cells and pads (assume pin locations are at cell centers). Pairwise nonoverlap constraints $c_{ij}(x, y)$ are directly expressed in terms of disk radii ρ_i and ρ_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 over a clique-model graph netlist approximation,

$$q(x, y) = \sum_{i,j} \gamma_{ij}((x_i - x_j)^2 + (y_i - y_j)^2), \tag{2}$$

is minimized subject to the pairwise nonoverlap constraints (for efficiency, large nets are modeled as chains rather than cliques). The NLP solver is a customized interior-point method. In order that overlap can be removed gradually, a slack variable ξ is added to both the objective and the constraints, as follows:

$$\begin{aligned} \min_{x,y,\xi} \quad & f(x, y) + \alpha\xi \\ \text{subject to} \quad & c_{ij}(x, y) + \xi \geq 0. \end{aligned}$$

The penalty weight factor α is gradually increased to remove overlap.

After nonlinear programming and the QRS local-relaxation sweeps described below, bin-area densities are balanced by displacement-minimizing linear assignment of clusters to bin locations. Discrete Goto-based swaps are then employed as described below to further reduce wirelength prior to interpolation to the next level.

For scalability, global relaxations (in which all variables are simultaneously modified) are avoided at all levels except the coarsest. Instead, two separate

sweeps of iterative improvement over small subsets of vertices are performed. A uniform grid is used to monitor the area-density distribution.

The first of these local-relaxation sweeps is called quadratic relaxation on subsets (QRS). Traversing the netlist in simple depth-first search (DFS) order, it selects movable vertices in small batches. For each batch, the quadratic wirelength of all nets containing at least one movable vertex, viewed as a continuous function of the movable vertex locations, is minimized without regard to overlap. Each such relocation typically introduces additional area congestion and is therefore followed directly by a clean-up step to keep the area-density distribution consistent. For this purpose, the "ripple-move" algorithm [Hur and Lillis 2000] is applied to any overfull bins after QRS on each batch. Ripple-move computes a max-gain monotone path of vertex swaps along a chain of bins leading from an overfull bin to an underfull bin. To facilitate the area-congestion control, only very small batches of movable cells are used in QRS; the batch size is set to three in the reported experiments.

After the entire sweep of QRS+ripple-move, a sweep of discrete, Goto-style permutations [Goto 1981] further reduces total wirelength. Vertices are visited one at a time in netlist order. The optimal "Goto" location of a given vertex a is computed by minimizing the sum of the bounding-box lengths of all nets containing a while holding all neighbors of a fixed. If a's Goto 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. To prevent corruption of a given cell-area distribution, swapping a large cell with a much smaller cell is explicitly disallowed.

Smoothing the distribution of cell area in the presence of widely varying cell or cluster sizes has particular importance. Interestingly, mPL *ignores* area variations among clusters at all coarser levels. That is, at every level except the finest, each cluster's area is set to the average of all cluster areas at that level. The reasons for this assumption's effectiveness are not completely understood. It is *not* used at the finest level, however, where larger-than-average cells are chopped into average-size fragments. After linear assignment of the cells and cell fragments to finest-level bins, fragments of the same cell are explicitly reunited. Any resulting area overflow is then removed by ripple-move cell propagation as described above.

Interpolation. mPL employs AMG-based weighted disaggregation in interpolating solutions from level to level. For each cluster at the coarser level, a single, "C-point" representative component is selected from it for use as a fixed anchor. C-points are selected for maximal netlist degree and large area (area is used only if the maximum-degree vertex is not unique). C-points are locked in place at their parent clusters' centers. The remaining "F-point" vertices in the

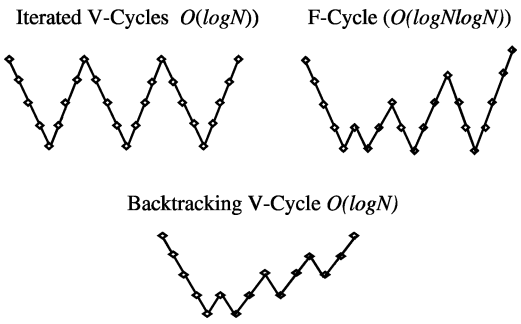


Fig. 5. Some iteration flows for multilevel optimization.

cluster are ordered by nonincreasing weighted hyperedge degree. Following this order, they are then placed one by one at the weighted averages of their strong C-point neighbors and strong F-point neighbors already placed. This F-point repositioning is iterated a few times, but the C-points are held fixed throughout.

Iteration Flow. The idea behind the F-cycle shown in Figure 5 is that the accuracy of a coarse-level solution can be enhanced by recursively applying the multilevel flow to it before interpolation. Although the F-cycle flow does not compromise scalabilty, assuming linear-order relaxations, it may increase run time considerably. In mPL, two backtracking V-cycles are used as a compromise—instead of descending all the way back to the coarsest level after each interpolation, mPL backtracks just one level before continuing toward finer levels. The first backtracking V-cycle follows the connectivity-based FC clustering hierarchy described in the coarsening section above. The second backtracking V-cycle follows a different 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 pass of clustering, vertex positions calculated in the first backtracking V-cycle are preserved by placing clusters at the weighted averages of their component vertices' positions. In the interpolation phase, however, the new clustering supports exploration of new territory in the solution space, enabling the flow to improve the result.

3.3 Embedded Multilevel Optimization

Leading algorithms owe their performance not only to their basic, outer structure but also to sophisticated and hierarchical iterative internal calculation. In particular, all leading contemporary partitioning-based methods as described in Section 3.1.1, including Capo [Caldwell et al. 2000b], Dragon [Sarrafzadeh et al. 2002], and Feng Shui [Yildiz and Madden 2001a; Agnihotri et al. 2003; Khatkhate et al. 2004], rely heavily on multilevel algorithms for netlist partitioning. Although each partitioning is, as a component of the placement algorithm, performed on individual cells rather than on aggregates of cells, the partitioning algorithm itself is multilevel. That is, a hierarchy of aggregates of cells

is formed by recursive clustering, each of the resulting levels is partitioned by FM, starting with the coarsest level, proceeding in sequence to the finest level, the solution at each level defining a starting point for iterative improvement at the next. Although no explicit use is made of the multilevel cluster hierarchy once partitioning is completed, it seems clear that the multilevel partitioners play an enabling role in these methods.

In fact, a placement problem of order 10^6 cells and nets can still be solved "flat," i.e., without any *explicit* aggregation or partitioning, provided that sufficiently fast and scalable numerical solvers are available for the given formulation. Two clear demonstrations of this approach are found in recent adaptations of force-directed methods [Quinn and Breuer 1979].

An AMG-Accelerated Force-Directed Method. Seminal work by Eisenmann and Johannes [1998] formulated placement as a sequence of unconstrained quadratic minimizations. The objective function

$$q(x, y) = \frac{1}{2}(x^T Q x + y^T Q y) + b_x^T x + b_y^T y + f_x^T x + f_y^T y$$

captures both netlist connectivity and area congestion by a graph approximation and force-field calculation, as follows. Cell-to-cell connections determine the off-diagonal entries and part of the diagonal entries in the fixed *graph Laplacian* matrix Q by means of a quadratic star-wirelength model [Kleinhans et al. 1991]. Cell-to-pad connections contribute to the diagonal elements of Q, rendering it positive definite, and determine the linear-term coefficients in the right-hand-side vector $b = (b_x, b_y)$. Viewing this vector b as external spring-like forces following Hooke's law, the circuit connectivity is represented by the (constant) symmetric-positive-definite matrix Q and the vector b. The perturbation vector $f = (f_x, f_y)$ represents global area-distribution forces analogous to electrostatic repulsion, with cell area playing the role of electric charge. At each iteration, vector f is recalculated from the current cell positions by means of a fast Poisson-equation solver. Since Q does not change from one iteration to the next unless nets are reweighted, a hierarchical set of approximations to Q can typically be reused over several iterations. We refer to this approach as *Poisson-based*.

Recently, a customized AMG-based linear-system solver was derived for iterated force-directed quadratic-wirelength minimization [Chen et al. 2003]. The motivation is simply to improve the scalability of the Poisson-based approach, the run time of which is dominated by linear-system solves. The AMG approach to linear systems proceeds by repeatedly applying a simple *relaxation* update to the approximate solution at each level of an aggregation hierarchy. The relaxation is typically componentwise (Jacobi, Gauss-Seidel, SOR, etc.): the kth equation of the system $\sum_j a_{ij} x_j = b_i$, $i = 1, \ldots, n$ is used to express the kth coordinate x_k of the solution in terms of the others, $x_i, i \neq k$, which are temporarily held fixed.[2] Coordinates are updated in turn, one at a time, in a *sweep* across the equations (for example, Golub and Van Loan [1989]). With a well

[2]Since the system matrix for the force-directed approach is positive definite, its diagonal elements are strictly positive, and the expression $x_k = (b_k - \sum_{j \neq k} a_{ij} x_j)/a_{kk}$ is well defined.

constructed hierarchy, solution of the optimality conditions

$$Qx = -(b_x + f_x) \quad \text{and} \quad Qy = -(b_y + f_y) \tag{3}$$

can be done in this way in linear or nearly linear amortized time. In the work of Chen et al. [2003], the hierarchy is derived by strict aggregation from four separate preliminary layouts. Several iterations of the SOR variant of coordinate-wise relaxation are applied to the flat, unconstrained (i.e., ignoring overlap) quadratic over four separate trials. In each trial, the cells are initially placed all at the same point: one of the four corners of the placement region. Clusters are selected according to cells' average final proximity over the results of all four trials. Although this iterative, empirical approach to clustering requires significant run time, it is a fixed initial cost that can be amortized over the cost of subsequent iterations. Numerical results confirm the scalability of this approach.

3.4 Advances in Analytical Placement

As the size and complexity of placement instances continue to increase, continuous approximation becomes ever more effective. The impact of algorithm enhancements also becomes more noticeable at larger scales. Several recent papers have introduced novel and intriguing variations of basic analytical frameworks that consistently improve performance. Three of them are summarized here.

3.4.1 *FastPlace.* Recent work of Chu and Viswanathan [2004] demonstrates that placements comparable in total wirelength to those of leading available academic tools can be computed in orders of magnitude less run time. Experiments comparing FastPlace to CAPO [Caldwell et al. 2000b] and Dragon [Sarrafzadeh et al. 2002] demonstrate speed-up factors of 20×—100× and relative differences in total wirelengths within 1–10%. The FastPlace flow combines simple, local and global heuristics in a way that supports fast computation and convergence. It repeats the following four steps until the movable cells are distributed evenly enough that legalization and detailed placement can be applied.

(1) Minimize unconstrained modified quadratic wirelength.
(2) Shift cells locally to relieve high-density regions.
(3) Move cells one by one to reduce linear half-perimeter wirelength (HPWL).
(4) Calculate displacement forces corresponding to the displacements in steps (2) and (3) and impose them cell by cell, by means of pseudo-nets connected to pseudo-pads, modifying the quadratic objective accordingly.

As the initial unconstrained quadratic tends to clump cells in the center of the region, cells flow generally from the center toward the boundary over the 20–30 iterations needed to converge.

Local cell shifting proceeds as follows: A uniform bin grid is laid over the region. The grid is sized so that the average number of cells in a bin is about 4. Bins are expanded or contracted independently in the horizontal and vertical

directions in proportion to their cell-area content (the resulting bins are no longer disjoint). Cells are moved with their bins during the bin resizing in proportion to the stretch or shrink. Local HPWL refinement proceeds greedily, cell by cell.

In contrast to the force-directed methods described in Section 3.3, FastPlace computes the forces needed to generate cell displacements already computed by other, local means. It imposes the forces only *after* making the displacements, incorporating them into the global quadratic objective as 2-pin pseudo-nets connecting cells directly to pseudo-pads placed along the chip boundary in the direction of the desired forces. These artificial forces prevent cells from collapsing back to their previous positions at the next iteration. The total number of iterations and the displacement per iteration are thus tightly controlled.

Interestingly, no hierarchy is used in FastPlace as originally published; all linear-systems are solved by preconditioned conjugate gradients (PCG) with generic ILU preconditioning [Saad 1996]. Technically this approach, as published, is not scalable, due to the larger number of iterations of PCG needed to solve (3) at the early placement iterations. As iterations proceed, however, the artificial-pad forces accelerate the convergence of the PCG iterations. The force associated with each cell displacement is imposed by connecting the cell to an artificial pad on the placement region boundary and weighting the connection to generate the displacement. Derivation of the linear system coefficients shows that this technique adds positive numbers to matrix diagonal elements. Because the quadratic wirelength model simulates a Hooke's-Law spring system, forces at later iterations must be stronger than at earlier iterations. Hence, the diagonal elements of the linear system (3) increase, making the system easier to accurately precondition, making the ILU preconditioner more effective, and reducing the number of iterations necessary for PCG to converge. Scalability can be obtained simply by applying an AMG-based solver at earlier iterations, as described above in the previous section.

3.4.2 *Grid Warping.* Following Proud [Tsay et al. 1988], most analytical placers, including Gordian [Kleinhans et al. 1991; Sigl et al. 1991] and FastPlace (Section 3.4.1 above), start from the premise that an unconstrained solution, that is, one obtained by minimizing wirelength without regard to nonoverlap or any other constraints, provides a high-quality relative ordering of the placeable objects. Although connections to fixed terminals generally make the optimal unconstrained cells locations distinct, the unconstrained solution is still, typically, extremely nonuniform and is not easily legalized in any way that approximately preserves the cells' relative orderings. What most distinguishes different analytical approaches is the manner in which they spread the cells from an initially highly nonuniform distribution to one uniform enough to be legalized without major perturbations. The novelty of grid warping [Xiu et al. 2004] is that, rather than directly move cells based on their distribution, it uses the cells to deform or *warp* the region in which they lie, in an analogy with gravity as described by Einstein's general relativity. The inverse of the deformation is then used to carry cells from their original locations to a more uniform distribution.

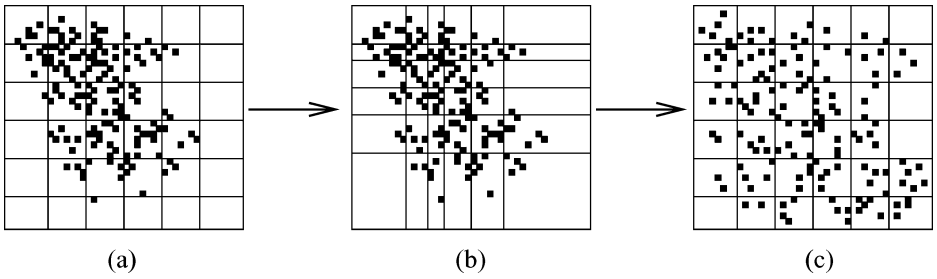


Fig. 6. Prewarping. The gridlines of a uniform bin grid (a) are translated so as to capture roughly equal numbers of cells in each row and column (b). The inverse translation is applied to cells as well as gridlines, giving a more even distribution of the cells over the region (c).

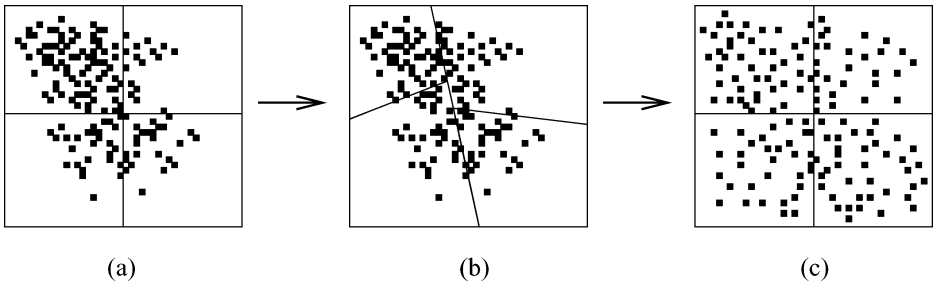


Fig. 7. Warping. Each bin of a uniform bin grid (a) is mapped to a corresponding quadrilateral in an oblique but slicing bin structure (b) so as to capture roughly equal numbers of cells in each quadrilateral. The inverse bin maps are applied to the cells in order to spread them out (c).

Two variants of the basic concept are shown in Figures 6 and 7. In Figure 6, a nonuniform rectilinear grid is defined so that each of its rows and columns contains approximately the same amount of total cell area. This simplified form of grid modification, in which all gridlines remain parallel to coordinate axes, is called "prewarping" by Xiu et al., because they find it useful as a fast, first step in the spreading process.

Figure 7 illustrates the more general approach. As shown, oblique grid lines are used, and although a slicing structure with alternating cutline directions and quadrilateral bins is maintained, gridlines not necessary to the slicing pattern are broken at points where they intersect other gridlines. This weakening of the grid structure allows close neighbors in the original unconstrained placement to be separated a relatively large distance by the warping. The warp is defined as a collection of bilinear maps, one from each of the bins in the original, uniform grid to the corresponding quadrilateral in the warped grid. To invert the warp and move the cells, the inverse of each such bilinear map is applied to the coordinates of all the cells in its quadrilateral bin. A block-scanline approximation algorithm is used for fast determination of which cells are contained in which quadrilateral.

The grid points of the warped grid are determined simultaneously by a derivative-free method of nonlinear optimization of Brent and Powell. The

required top-down slicing grid structure is maintained by (a) fixing the alternating cutline direction order a priori, by deciding whether to orient the first cut from top to bottom or from side to side, and (b) expressing each cutline after the first in terms of 2 variables, one for where it intersects its parent cutline, and another for where it intersects the opposite boundary or cutline. A penalty function f is used as the objective:

$$f = \text{wirelength} + \rho \cdot \sum_{\text{bins}} \beta_{ij},$$

where β_{ij} is approximately the square of the difference between the total cell area in bin (i, j) and the target cell area $\kappa = \kappa(i, j)$ for each bin. Wirelength is the total weighted half-perimeter wirelength obtained after the inverse warp. Although evaluating the objective is fairly costly, the number of variables in the optimization is low—only 6 for a 2×2 grid or 30 for a 4×4 grid—and convergence is fast.

To obtain an effective and scalable placement algorithm, this basic spreading operation must still be incorporated within some hierarchical framework, for example, top-down partitioning or multilevel optimization. Xiu et al. [2004] use top-down partitioning, with the partitions defined by grid warping. That is, each step of grid warping defines a partition of both cells and space. Cutsize-driven partitioning is used to separate cells lying on grid boundaries. Prewarping and warping recurs on the subregions defined by the grids. The overall flow is summarized below.

(1) Unconstrained quadratic-wirelength placement.
(2) Redistribute cells by prewarping (e.g., with an 8×8 grid).
(3) Redistribute cells by linear-wirelength optimizing grid warping (e.g., with a 4×4 grid).
(4) Assign cells to grid bins, using cut-size driven partitioning only on cells near bin boundaries.
(5) Propagating terminals to bin boundaries, recur prewarping, warping, and cell partitioning separately on the cells in each bin, until the overall cell distribution is even enough for legalization.

3.4.3 *Multilevel Generalized Force-Directed Placement.* While the force-directed framework described in Section 3.3 has broad appeal for its generality and scalability, considerable effort is needed to produce a fast and stable implementation [Vorwerk et al. 2004]. Recently, this framework has been generalized to a more rigorous mathematical formulation and adapted to a multilevel implementation in mPL5 [Chan et al. 2005]. An overview of the mPL5 formulation is given here.

Placement objectives and constraints at each level of a cluster hierarchy (Section 3.2) are approximated by smooth functions. A bounding-box weighted wirelength objective is approximated by the log-sum-exp model

[Naylor et al. 2001; Kahng and Wang 2004]

$$W(x, y) = \gamma \sum_{\text{nets } e \in E} (\log \sum_{\text{nodes } v_k \in e} \exp(x_k/\gamma) + \log \sum_{v_k \in e} \exp(-x_k/\gamma) + \log \sum_{v_k \in e} \exp(y_k/\gamma) + \log \sum_{v_k \in e} \exp(-y_k/\gamma)), \tag{4}$$

where $\mathbf{x}$ and $\mathbf{y}$ denote vectors of cell x- and y-coordinates. The smaller the parameter γ, the more accurate the approximation. Letting D_{ij} denote the cell-area density of bin B_{ij} and K the total cell area divided by the total placement area, the area-density constraints are initially expressed simply as $D_{ij} = K$ over all bins B_{ij}. Viewing the D_{ij} as a discretization of the smooth density function $d(x, y)$, these constraints are smoothed by approximating d by the solution ψ to the Helmholtz equation

$$\begin{cases} \Delta\psi(x, y) - \epsilon\psi(x, y) = d(x, y), & (x, y) \in R \\ \frac{\partial\psi}{\partial\nu} = 0, & (x, y) \in \partial R \end{cases} \tag{5}$$

where $\epsilon > 0$, ν is the outer unit normal, ∂R is the boundary of the placement region R, $d(x, y)$ is the continuous density function at a point $(x, y) \in R$, and Δ is the Laplacian operator $\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$. The smoothing operator $\Delta_\epsilon^{-1} d(x, y)$ defined by solving (5) is well defined, because (5) has a unique solution for any $\epsilon > 0$. Since the solution of (5) has two more derivatives [Evans 2002] than $d(x, y)$, ψ is a smoothed version of d. Discretized versions of (5) can be solved rapidly by fast numerical multilevel methods. Recasting the density constraints as a discretization of ψ gives the nonlinear programming problem

$$\begin{array}{ll} \min & W(\mathbf{x}, \mathbf{y}) \\ s.t. & \psi_{ij} = -K/\epsilon, \quad 1 \le i \le m, 1 \le j \le n, \end{array} \tag{6}$$

where the ψ_{ij} are obtained by solving (5) with the discretization defined by the given bin grid. Interpolation from the adjacent coarser level (Section 3.2) defines a starting point. This nonlinear-programming problem is solved by the Uzawa iterative algorithm [Arrow et al. 1958], which does not require second derivatives or large linear-system solves:

$$\begin{gathered} \nabla W(\mathbf{x}^{k+1}, \mathbf{y}^{k+1}) + \sum_{i,j} \lambda_{ij}^k \nabla\psi_{ij} = 0 \\ \lambda_{ij}^{k+1} = \lambda_{ij}^k + \alpha(\psi_{ij} + \bar{K}/\epsilon) \end{gathered} \tag{7}$$

where λ is the Lagrange multiplier, $\lambda^0 = 0$, α is a parameter to control the rate of convergence, and gradients of ψ_{ij} are approximated by simple forward finite differences $\nabla_{x_k}\psi_{ij} = \frac{\psi_{i,j+1} - \psi_{i,j}}{h_x}$, $\nabla_{y_k}\psi_{ij} = \frac{\psi_{i+1,j} - \psi_{i,j}}{h_y}$ when the center of cell v_k is inside B_{ij} and are set to zero otherwise. The nonlinear equation for $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})$ is recast as an ordinary differential equation and solved by an explicit Euler method [Morton and Mayers 1994].

Multiscaling the generalized force-directed algorithm renders it much more scalable. Enabling efficient, global analytical relaxations in the multilevel framework dramatically improves placement quality. Compared with other leading academic tools Dragon 3.01, Capo 8.8, Feng-Shui 2.6 and FastPlace 1.0 on standard IBM-ISPD98 benchmark circuits, mPL5's average wirelength

is the shortest—1% shorter than Dragon's, 10% shorter than Capo's, 4% shorter than Feng-Shui's, and 11% shorter than FastPlace's. The run time of mPL5 is also very competitive: 9× faster than Dragon's, about the same as Capo's, 2× faster than Feng-Shui's, and 8× slower than FastPlace's. A fast mode of mPL5 gives wirelength roughly between that of Feng-Shui and Capo with run time only 2× longer than that of FastPlace.

3.5 Legalization and Detailed Placement

While most research in placement is still directed at global placement, some recent work on the PEKO benchmarks (Section 2) suggests that existing global-placement algorithms already place nearly every cell within a few cell-widths of its globally optimal location [Ono and Madden 2005]. This observation has renewed interest in improved methods for legalization [Li and Koh 2003; Agnihotri et al. 2003; Khatkhate et al. 2004] and detailed placement [Brenner et al. 2004; Ramachandaran et al. 2005]. Nevertheless, progress in global placement still continues to be made. The search for the most effective means of legalizing and refining a good global placement plays a critical role in current efforts to further reduce the gap between achievable and optimal placements.

4. TIMING OPTIMIZATION

Extensive research on timing-driven placement has been done in the past two decades and continues today. The performance of a circuit is determined by its longest path delay, but timing constraints are extremely complex. The number of paths present grows exponentially with circuit size. Even a circuit of modest size can have a huge number of paths. For example, Chang et al. [1994] estimated the number of path constraints in a 5K-cell design to be around 245K, requiring roughly 243Mb memory space if stored explicitly. Moreover, users may have different requirements for different paths. For example, a circuit may have different tsu (input to register), tco (register to clock output), r2r (register to register) or i2o (input to output) requirements for individual nodes, or paths. The existence of multiple clock domains and multiple cycle paths makes the problem even more complicated.

Existing timing-driven placement algorithms can be broadly classified into two categories: path-based and net-based.

4.1 Path-based Algorithms

Path-based algorithms try to directly minimize the longest path delay. Popular approaches in this category include mathematical programming and iterative critical path estimation.

Formulation of the problem as a linear or nonlinear programming problem typically introduces auxiliary variables (i.e., *arrival times*) at circuit nodes [Jackson and Kuh 1989; Srinivasan et al. 1991; Hamada et al. 1993]. Different mathematical programming techniques can then be used to solve the problem. For example, in terms of arrival time $a(i)$ at pin i, timing constraints

may be expressed as follows.

$$\begin{array}{ll} a(j) \geq a(i) + d(i,j) & \forall (i,j) \in G \\ a(j) \leq T & \forall j \in PO \\ a(i) = 0 & \forall i \in PI, \end{array}$$

where

G	denotes the timing graph.
PI	denotes the set of starting points of any timing path, including primary input pins and output pins of memory elements.
PO	denotes the set of ending points of any timing path, including primary output pins and data input pins of memory elements.
$d(i,j)$	denotes the delay of timing arc (i,j), which is either a constant (for cell internal delay) or a function of cell locations.
T	denotes the longest path-delay target.

Here we assume that the arrival time at all PI pins is zero, and that all PO pins have the same delay targets. Simple changes can be made to the formula to accomodate more complex situations.

Explicitly minimizing the sum of the lengths of the paths in some set of critical paths is a popular approach. This set of critical paths can be precomputed in a static manner or dynamically adjusted from iteration to iteration. TimberWolf [Swartz and Sechen 1995] used simulated annealing to minimize a set of pre-specified timing-critical paths, while mathematical programming techniques [Burstein and Youssef 1985; Marek-Sadowska and Lin 1989] have also been employed.

The advantage of path-based algorithms is their accurate timing view during the optimization procedure. However, the drawback is that they usually require substantial computation resources due to the exponential number of paths which need to be simultaneously minimized. Moreover, in certain placement frameworks, for example, top-down partitioning, it is very difficult or infeasible to maintain an accurate global timing view.

4.2 Net-based Algorithms

Net-based algorithms [Dunlop et al. 1984; Nair et al. 1989; Tsay and Koehl 1991; Eisenmann and Johannes 1998], on the contrary, do not directly enforce path-based constraints. Instead, timing constraints or requirements on paths are transformed into either net-length constraints or net weights. This information is then fed to a weighted-wirelength-minimization-based placement engine to obtain a new placement with better timing. This new placement is then analyzed by a static analyzer, thus generating a new set of timing information to guide the next placement iteration. Usually this process must be repeated for a few iterations until no improvement can be made or until a certain iteration limit has been reached.

The process of generating net-length constraints or net-delay constraints is called *delay budgeting* Hauge et al. 1987; Gao et al. 1991; Luk 1991; Youssef et al. 1992; Tellez et al. 1996; Sarrafzadeh et al. 1997a, 1997b; Chen et al.

2000]. The main idea is to distribute slacks at the endpoints of each path (POs or inputs of memory elements) to constituent nets in the path such that a zero-slack solution is obtained [Nair et al. 1989; Youssef and Shragowitz 1990; Chen et al. 2000]. The original zero-slack algorithm (ZSA) [Nair et al. 1989] assigns slacks based mainly on fanout factors. Subsequently, researchers considered a more general framework [Sarrafzadeh et al. 1997b] in which delay budgeting is formulated as follows.[3]

$$\begin{array}{ll} \max & \sum_{(i,j)\in G} C_{ij}(s_{ij}) \\ \text{such that} & \\ & s_{ij} = a(j) - a(i) - d(i,j) \quad \forall (i,j) \in G \\ & a(j) \le T \qquad\qquad\qquad\quad \forall j \in PO \\ & a(i) = 0 \qquad\qquad\qquad\quad \forall i \in PI \end{array}$$

Here s_{ij} is slack for edge (i, j), and $C_{ij}(s_{ij})$ is a flexibility function[4] of edge (i, j). The intuition is that, since delay budgeting will generate a set of constraints for the placement, these constraints are stated as weakly as possible, in order to minimize their impact on solution quality.

A serious drawback of this class of algorithms is that delay budgeting is usually done in the circuit's structural domain, without consideration of physical placement feasibility. There is no generally conceived good flexibility function. As a result, it may severely overconstrain the placement problem. Recently, some attempts have been made to unify delay budgeting and placement [Sarrafzadeh et al. 1997a; Yang et al. 2002a; Halpin et al. 2001], where a complete or coarse [Yang et al. 2002a; Halpin et al. 2001] placement solution is used to guide the delay budgeting step. However, it is generally difficult to find an efficient or scalable algorithm for such unification.

To overcome these problems, approaches based on net weighting use different means. Instead of assigning a delay budget to each individual net or edge, net-weighting-based approaches assign weights to nets based on their timing criticality. Compared with delay-budgeting approaches, these methods will not suffer from the overconstraining problem. Net-weighting-based algorithms are generally very flexible. They can be integrated naturally into an existing wirelength-minimization-based placement framework. They also have a relatively low complexity. As circuit sizes continue to increase and practical timing constraints become increasingly complex, these advantages make the net-weighting-based approaches more and more attractive.

Consider the following simple example. Suppose one circuit contains only one timing path P, which consists of the following two-pin connections: $e_1, e_2, \ldots, e_n$. Let $d(e_i)$ denote the delay of edge e_i . Using a net-weighting approach, we assign the weight of each edge the same value, say 1 (as all edges are in the same path, they have the same criticality). We can transform the path minimization

[3]The formulation has been modified to fit the context here.
[4]This concept is not explicitly used in the original paper.

problem *exactly* into the following problem:

$$\min \sum_{i=1}^{n} d(e_i).$$

However, under a delay-budgeting approach, we first need to find a delay budget $b(e_i)$ for each edge, then solve the following constrained optimization problem:

$$\begin{array}{ll} \min & f(x) \\ \text{such that} & d(e_i) \leq b(e_i), \forall 1 \leq i \leq n \end{array}$$

The objective function $f(x)$ is usually wirelength; timing has been relegated to the constraints. It is obvious that this method suffers from the overconstraining problem: if we use a nonoptimal delay budget, there is no guarantee we can find a solution as good as that obtained under a net-weighting approach. An *optimal* delay budget is very difficult to compute without solving the placement problem. Even if an optimal delay budget can be obtained, there may exist many optimal delay budgets whose resulted timing constraint sets may differ wildly. Some may be very tight, others may be very loose, and predicting which are tight or loose may be very difficult.

Unfortunately, despite its advantages, net weighting is usually done in an ad-hoc, intuitive manner. The main principle used in most algorithms is that a timing-critical net should receive a heavy weight. For example, VPR [Marquardt et al. 2000] used the following formula to assign weight to an edge e:

$$w(e) = (1 - slack(e)/T)^{\alpha}$$

where T is the current longest path delay and α is a constant.

These methods ignored another important principle – *path sharing*. In general, an edge with many paths passing through it should receive a heavy weight as well. *Path counting* is a method developed to take path-sharing effects into consideration by computing the number of paths passing through each edge in the circuit. These numbers can then be used as edge weights. Unfortunately, the naive approach suffers from a severe drawback: it cannot distinguish timing-critical paths from non-critical paths. The variant ϵ-network path counting [Senn et al. 2002] suffers from the same problem [Kong 2002].

A nice solution has recently been proposed [Kong 2002]. The algorithm, named PATH, can properly scale the impact of all paths by their relative timing criticalities (measured by their slacks) respectively, instead of counting critical paths and non-critical paths with equal weight. It is shown [Kong 2002] that for certain *discount* functions, this method is equivalent to enumerating all the paths in the circuit, counting their weights, and then distributing the weights to all edges in the circuit. Such computation can be carried out very efficiently in linear time, and experimental results have confirmed its effectiveness. Compared with VPR [Marquardt et al. 2000] under the same placement framework, PATH reduces the longest path delay by 15.6% on average with no runtime overhead and only a 4.1% increase in total wirelength.

Like a standard static timing-analysis algorithm, the PATH algorithm works in two phases. In the first, forward phase, a forward partial counting is conducted, while in the second, backward phase, a backward partial counting is

Table I. The PATH Algorithm Accurately Computes the Impact of All Paths Passing Through Each Edge

```
1.   set F(p) = B(p) = 0 for each pin p;
2.   for each PI pin p, set F(p) = 1;
3.   traverse pins t in topological order
4.       for each input pin s of t
5.           F_s(s,t) = a(t) − a(s) − d(s,t);
6.           compute discount = D(F_s(s,t), T);
7.           F(t) += discount × F(s);
10.
11.  for each PO pin p, set B(p) = 1;
12.  traverse pins s in reverse topological order
13.      for each output pin t of s
14.          B_s(s,t) = r(t) − d(s,t) − r(s);
15.          compute discount = D(B_s(s,t), T);
16.          B(s) += discount × B(t);
19.
20.  for each edge (s,t), compute
21.      AP(s,t) = F(s) × B(t) × D(slack(s,t), T);
```

Here $r(t)$ represents the required arrival time at pin t, $\mathcal{D}(x)$ represents the weighting function (called the *discount function*).

performed. PATH maintains two counters for each pin p: $F(p)$, the forward partial counter, and $B(p)$, the backward partial counter. For each edge (s,t), PATH also maintains a counter $AP(s,t)$ for the total number of weighted paths passing through it. A brief description of the algorithm is listed in Table I.

It is interesting to note that if we use a trivial discount function $\mathcal{D}(x) = 1$, we will get the total number of paths passing through each edge in the timing graph.

A potential problem in net-based approaches is the so-called *oscillation* problem. Usually net weights or budgets are assigned by performing timing analysis for some given placement solution P^n at the nth iteration. More critical nets receive higher weights. Thus, in the next placement solution P^{n+1}, the lengths of critical nets in P^n will be reduced, while the lengths of other noncritical nets are potentially increased, resulting in changes in net criticalities, and, thus, in net weights. If a net alternates between critical and noncritical, its length may alternately increase and decrease, impeding convergence. Certain path-based approaches suffer from similar problems, for example, a need to dynamically adjust the set of paths being optimized [Swartz and Sechen 1995].

Two ways to eliminate the oscillation problem appear widely in the literature. The first approach is to perform timing analysis and recompute net weighting periodically. VPR [Marquardt et al. 2000] and PATH [Kong 2002] follow this approach. Based on simulated annealing, both methods perform timing analysis and net re-weighting once per temperature. The second approach is to make use of historic information [Eisenmann and Johannes 1998], that is, to combine weights in previous iterations with criticality information in the current placement to derive the current weights. Intuitively, if a net is always critical during all placement iterations, we want to gradually increase its weight; while if it is never critical, we will gradually decrease its weight. As a typical example,

we consider the approach used by Eisenmann and Johannes [1998]. At each iteration m, each net e has criticality $c_m(e)$, initialized to zero and updated as follows:

$$c_m(e) = \begin{cases} (c_{m-1}(e) + 1)/2 & \text{if } e \text{ is among 3\% most critical} \\ c_{m-1}(e)/2 & \text{otherwise.} \end{cases}$$

Weight $w_m(e)$ is initialized to one and updated as

$$w_m(e) = w_{m-1}(e)(1 + c_m(e)).$$

The general approaches described above can be viewed as iteratively targeting the worst negative slack (WNS). Rather than focus only on the most critical path, it is possible to consider the sum of all slacks over all paths. In this approach, different paths have different delay targets and thus different required arrival times. The timing constraints for a given path are satisfied at a point i if the slack at i is sufficiently positive. More recently, a sensitivity-guided net-weighting strategy based on minimizing total slack over all paths have been proposed [Ren et al. 2004]. In this approach, nets are targeted for their impact on the global objective and not necessarily for their criticality.

5. ROUTABILITY OPTIMIZATION

Routing congestion is one of the fundamental issues in VLSI physical design. Because an aggressive wirelength-driven placement may not be routable, routability is best considered directly during the placement phase. Routability-driven placement involves mainly (i) routability modeling and (ii) optimization techniques for routability control. Usually optimization for routability control is performed based on the estimated routing congestion of a placement configuration. We discuss these two issues in the following subsections.

5.1 Routability Modeling

Routability is usually modeled on an $X \times Y$ global-routing grid in the chip's core region. Routing supply and demand are modeled for each bin and/or each boundary of the routing grid structure. There are two major categories on routability modeling: topology-free (TP-free), where no explicit routing is done, and topology-based (TP-based), where routing trees are explicitly constructed on some routing grid.

5.1.1 *Topology-free Modeling.* TP-free modeling is faster in general. Examples of this class include bounding-box (BBOX)-based modeling [Cheng 1994], probabilistic analysis-based modeling [Lou et al. 2002; Westra et al. 2004], Rent's rule-based modeling [Yang et al. 2002b], and pin density-based modeling [Brenner and Rohe 2003].

In RISA modeling [Cheng 1994], the routing supply for each bin in the routing grid structure is modeled according to how the existing wiring of power or clock nets, regular cells, and macros (macros are referred to as "mega cells" [Cheng 1994]) are placed, and the routing demand of a net is modeled by its weighted BBOX length. Let T_v and T_h denote the total number of full tracks available in both vertical and horizontal directions over the core area including all metal

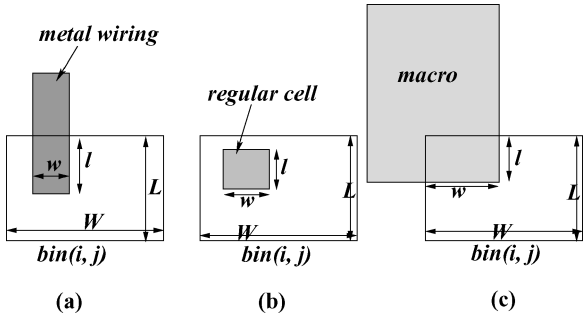


Fig. 8. RISA modeling for routing supply of each bin with existing wiring (a), regular cell (b), and macro(c).

layers. Given a global-routing bin structure of $X \times Y$ bins, the initial routing supply of a bin (i, j) is

$$S_{ij}^{v} = \frac{T_v}{X}, \quad S_{ij}^{h} = \frac{T_h}{Y}.$$

Existing wiring of power and clock nets, regular cells, and macros are considered to an obstacle to routing; thus, the routing resource supply decreases when any of these elements is found in a bin. Given a bin (i, j) of width W and length L, when existing wiring on layer m is found in it as shown in Figure 8(a), the routing supply of the bin decreases by $w \times l/L$, where w is the width of the wiring, expressed as a number of routing tracks. A regular cell normally creates a blockage as large as its outline at the first metal layer and a few smaller blockages in the second metal layer. Therefore, the routing supply decrease due to a regular cell of width w and length l in a bin, as shown in Figure 8(b), is

$$\begin{aligned} &T_1 \times \frac{l \times w}{L \times W} && \text{(metal layer 1—horizontal)} \\ &c/s \times T_2 \times \frac{l \times w}{L \times W}, && \text{(metal layer 2—vertical)} \end{aligned}$$

where T_1 and T_2 are numbers of tracks on the first and second metal layers of the bin, respectively, s is the cell width in units of the vertical tracks (assuming the routing tracks on the second metal layer are vertical), and c is the number of vertical tracks occupied by the blockages of the cell on the second metal layer as well as blockages caused by connection to pins in metal layer 1. In multilayer designs, macros usually fully block the first two metal layers over the macros' outlines. When a macro is placed over a bin as shown in Figure 8(c), the routing supply decreases by $(T_1,\ T_2) \times \frac{(l \times w)}{(L \times W)}$ in the respective horizontal and vertical directions, where w and l are the width and length of the rectangular intersection of the macro and the bin, respectively.

Although an optimal Steiner tree is not used for estimating the routing demand of a net, the probability of having a wire at location (x, y) within a net's BBOX is approximated by adding up and normalizing K optimal Steiner trees of K sets of M randomly located pins each (e.g., $K = 10,000$). A wiring distribution map (WDM) is obtained for each pin-count case in each direction. It is incorporated into the wirelength objective by adjustment of net weights. The

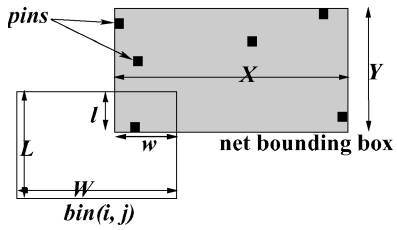


Fig. 9. RISA modeling for routing demand.

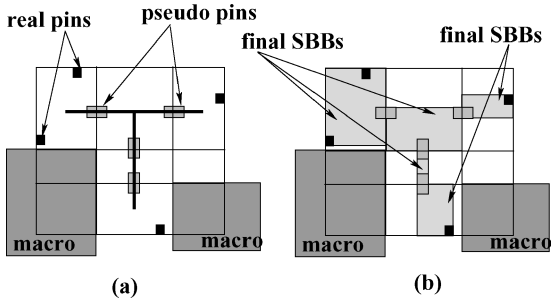


Fig. 10. Partition net BBOX (NBB) into sub-BBOX (SBB) to model the detour due to macro blockage.

new net weights are calculated by finding the mean values of track usage over the WDMs for each of the different pin counts. The net weight q represents the expected number of wires crossing a cut line through the net bounding box when a net of a high pin count is routed, no matter whether the cut line is vertical or horizontal or where it is. Given a net BBOX overlapping with a bin (i, j) as shown in Figure 9 and the net weight q in the unit of routing tracks, the routing demand of the bin increases by $q \times (\frac{w \times l}{Y \times W}, \frac{w \times l}{X \times L})$.

When a net BBOX overlaps with macros, which usually block the first and second metal layers completely, the above model is revised such that possible detours due to macro blockage can be considered during the estimation. The revision is based on partitioning the net boundary box (NBB) into a set of sub-bounding boxes (SBB) efficiently. Given a net BBOX and the overlapping macros, SBBs are obtained by extending each boundary of each macro such that it cuts through the NBB, as shown in Figure 10(a). The SBB set also forms a coarse global routing grid on which a multilayer routing is performed to find a Steiner tree that connects all the SBBs where pins are located. If such a Steiner tree does not exist, expand the NBB by 2X and search again, repeating until the NBB covers the full core area. For each edge of the Steiner tree, if it passes through any boundary of any SBB, place a pseudo-pin at the center of the boundary of that SBB. For each SBB which covers the real pins and pseudo-pins, resize it to contain both real pins and pseudo-pins as shown in Figure 10(b) and estimate routing demand using the weighted BBOX model.

The probabilistic analysis-based modeling generally assumes that (i) all nets are optimally routed with the shortest length, (ii) at most one change in

direction per grid, and (iii) no change if direction in grid with pin, unless more than one pin in the same grid. A net-based stochastic model for 2-pin nets is presented to compute expected horizontal and vertical track usage with consideration of routing blockage [Lou et al. 2002]. It is further extended based on the experimental evidence that shows only a negligible number of nets will have detours and the number of nets with many bends can be ignored [Westra et al. 2004]. Peak routing demand and regional routing demand are estimated using Rent's rule [Yang et al. 2002b]. Hu and Marek-Sadowska propose a Rent's-Rule-based implicit white-space allocation method to catch the congestion picture and to weight the BBOX length of the nets based on the pin locations [Hu and Marek-Sadowska 2002]. Its implicit modeling helps to combine estimation and optimization in one step. Pin density per bin can be used as a metric for intrabin routing congestion, but it cannot model the interbin boundary congestion. Therefore, it is combined with probabilistic analysis-based modeling for completeness [Brenner and Rohe 2003].

5.1.2 *Topology-based Modeling.* In a TP-based modeling method, for each net, a Steiner tree topology is generated on the given routing grid. Because these routing topologies usually have a fairly strong correlation with the topologies a global router generates, TP-based modeling can be quite accurate. At least, it generates a global routing solution, that is, it provides an upper bound for routability estimation. If a TP-based modeling method uses a topology similar to what the after-placement-router does, the fidelity of the model can be guaranteed. However, topology generation is often of high complexity; therefore, most research focuses mainly on efficiency.

In one approach [Mayrhofer and Lauther 1990], a precomputed Steiner tree topology on a few grid structures is used for wiring-demand estimation. This approach is tailored for recursive partition-based placement. In another approach [Chang et al. 2003a], two algorithms of logarithmic complexity were recently proposed: a fast congestion-avoidance two-bend routing algorithm, LZ-router, for two-pin nets, and IncA-tree algorithm, which can support incremental updates for building a rectilinear Steiner arborescence tree (A-tree) for a multi-pin net. The LZ-router uses auxiliary data structures (similar to a segment-tree data structure) to find good quality routes by performing a binary search of the possible routes for a two-pin net. The *wire density* of a bin/region is defined as the wire usage of the bin/region divided by its area. For a net connecting two pins, P_1 and P_2, which are bounded by a rectangle bounding box B (Figure 11(a)), if the maximum of the wire density of the vertical (horizontal) boundary bins of B on the vertical (horizontal) layer is $WD^V_{B_b}$ ($WD^H_{B_b}$), the wire density of region B on the horizontal (vertical) layer is $WD^H_{B_r}$ ($WD^V_{B_r}$), then the *possible maximum wire density* of VHV (HVH) routing is the maximum of $WD^V_{B_b}$ ($WD^H_{B_b}$) and $WD^H_{B_r}$ ($WD^V_{B_r}$). The VHV routing pattern (Figure 11(b)) will be chosen if its possible maximum wire density is smaller than that of HVH, otherwise, the HVH routing pattern (Figure 11(c)) is chosen. Assuming the VHV routing is used, the LZ-router recursively makes a horizontal cut on B and selects the one with a smaller wire density to route. It stops when the choice narrows to a single row. Given a $g_x \times g_y$ bin structure, the complexity

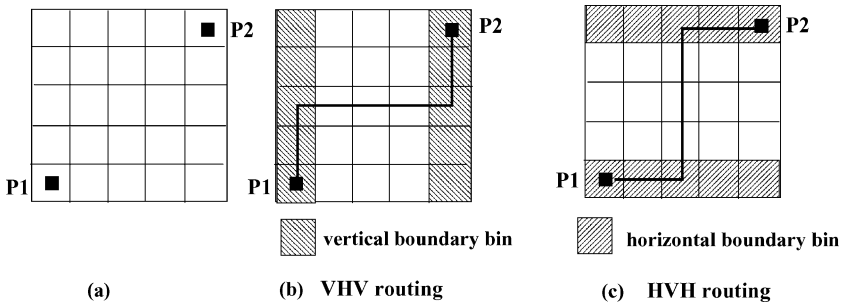


Fig. 11. An illustration of HVH and VHV routing selection of LZ-router.

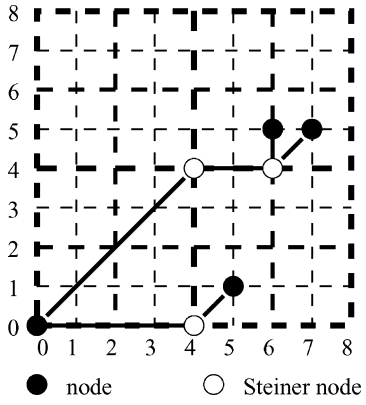


Fig. 12. An illustration of IncA-tree algorithm.

for the LZ-router to route two pins with coordinates (i, j) and $(i + x, j + y)$ is $O(log(\mid x \mid + \mid y \mid)log(g_x + g_y))$.

The incremental A-tree (IncA-tree) algorithm is developed to efficiently update the routing topology for each pin location change. Given a grid structure consisting of $(2^m + 1) \times (2^m + 1)$ grids on the first quadrant, it can be recursively quadri-partitioned until each partition becomes the unit grid. For example, the grid structure in Figure 12 is first quadri-partitioned by the cut lines $x = 4$ and $y = 4$ to form four partitions. If there are some pins located inside a partition (including locations on the bottom and left boundaries, but excluding the locations on the right and top boundaries), the lower-left corner of the partition is the root for a subtree connecting all the pins inside this partition. For example, $(4, 4)$ is the root for any pin at location (x, y) with $4 \leq x < 8$ and $4 \leq y < 8$. For a partition with some pins located inside, its root has an edge connecting to the lower-left corner of the previous level quadri-partition. In the above example, $(4, 4)$ has an edge connecting to $(0, 0)$. By recursively performing such quadri-partition, an A-tree can be built such that each pin at location (x, y) can connect to the origin $(0, 0)$ with $max(\log x, \log y)$ edges. Any pin insertion (deletion) to location (x, y) only incurs, at most, $\log(x + y)$ edge insertions (deletions). Therefore, each operation of moving a pin from (x_1, y_1) to (x_2, y_2) incurs, at most, $\log(x_1 + y_1) + \log(x_2 + y_2)$ edge changes.

With the fast two-pin routing and incremental A-tree routing, for an n-pin net with bounding box length L on a $g_x \times g_y$ bin structure, the complexity of updating a non-root pin move is $O(\log L)$ times the complexity of LZ-routes $O(\log L \log(g_x + g_y))$, which is $O((\log L)^2 \log(g_x + g_y))$. For moving the root, the complexity is $O(n(\log L)^2 \log(g_x + g_y))$. While providing superior guidance for congestion optimization during the coarse placement, the runtime overhead of this congestion cost updating grows slowly due to the low logarithmic complexity. It is obvious that the IncA-tree may generate routes with longer wire length than the A-tree does and using it may overestimate the congestion. However, it is never intended to be used as the final measurement of the placement congestion. Instead, it is used to guide the placement optimization.

5.2 Optimization Techniques

After routability is modeled, a routing-congestion picture is obtained on the global-routing grid structure. Basically, there are two ways to apply the modeling results to the placement optimization process: net weighting and cell weighting (cell inflation).

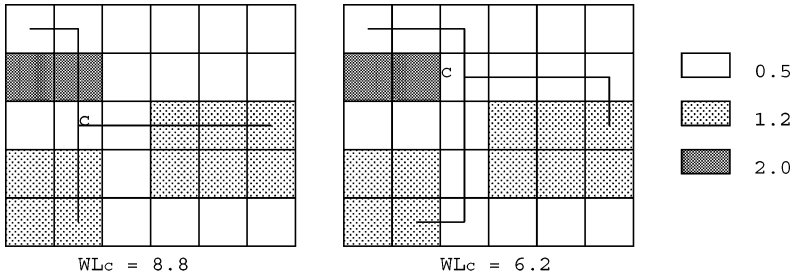
Net weighting directly transfers a congestion picture into net weights and optimizes weighted wirelength. It can easily be incorporated into iterative placement algorithms such as simulated-annealing-based methods [Hu and Marek-Sadowska 2002; Chang et al. 2003a].

Cell weighting (a.k.a cell inflation) inflates cell sizes based on congestion estimation, so that cells in congested bins can be moved out of the bins after being inflated. It is more suitable for incorporation into constructive placement techniques, such as analytical placers [Parakh et al. 1998], quadrisection-based placers [Brenner and Rohe 2003], as well as iterative placement techniques, such as simulated annealing-based placers [Yang et al. 2003].

Example: Routability Control in mPL-R

As a concrete example of routability optimization in global placement, techniques recently developed for mPL (Section 3.2) are described [Li et al. 2004]. This work consists of both demand-driven congestion reduction and supply-driven white-space allocation. Both of these components are defined in terms of the estimated routing overflow of each bin of a uniform rectilinear grid; that is, the amount by which routing demand exceeds routing supply in that bin (Section 5.1.2).

Demand-Based Congestion Control via Topology-Based Weighted Wirelength. To reduce routing demand, the wirelength-driven placement of each subset of cells is supplemented by a routability-driven step. Immediately after the cells in a given subset are placed to minimize wirelength, they are moved again so as to reduce estimated routing congestion in the subregions they occupy. The changes in congestion are computed by explicit updates to the estimated routing topology. A secondary objective during this process is still to reduce the wirelength. Candidate cells for re-placement are selected based on the routing topology of nets incident on them. Nets are sorted in descending order according



(a) (b) (legend)

Fig. 13. Congestion driven cell re-placement. The legend corresponds to the congestion in different routing regions. (a) Original placement of cell c in the optimal location for half-perimeter wirelength gives a weighted wirelength of 8.8. (b) Re-placement of c in a neighboring region gives a weighted wirelength of 6.2.

to the bin-capacity overflow they cause. The first s nets are picked, such that the total amount of routing resources they use exceeds the total overflow of the current placement. The cells connected by these nets are re-placed, as described next.

For each cell c to be re-placed, the grid cell gc_{ij} corresponding to its optimal bin location $\ell^* = (i^*, j^*)$ for half-perimeter wirelength is determined by holding all other cells fixed. Then the cell is placed in each of the neighboring bins of within a certain distance d, that is, $\{b_{ij} \,|\, |i^* - i| + |j^* - j| \leq d\}$. Each time c is placed in a bin, the topology for the nets incident on it is recomputed using the LZ router described in Section 5.1.2. The new placement for c is evaluated using a weighted wirelength of all the nets incident on c:

$$WL_c = \sum WGT_{net_k} \times WL_{net_k}, \tag{8}$$

where WGT_{net_k} is the weight on net_k, calculated as the average congestion of the grids cell net_k crosses, and WL_{net_k} is the half perimeter wirelength of net_k. Finally, c is placed in the bin that results in the shortest weighted wirelength. Figure 13 shows an example. Starting from the optimal location ℓ^* for half-perimeter wirelength, Weighted wirelength (8) is evaluated at each bin in the neighborhood of ℓ^*, and the bin that gives the shortest weighted wirelength is selected as c's final location. In this example, the original location ℓ^* gives a weighted wirelength of 8.8, whereas the final location gives a smaller weighted wirelength of 6.2.

Detailed experiments show that mPL-R's topology-based method of routability optimization is extremely effective. Compared to purely wirelength-driven mPL, mPL-R alone reduces the number of overflowed global routing bins by 83% over the complete set of IBM-Dragon Version 2 easy and hard benchmarks and leads to successful routing (by Cadence WarpRoute) of 14 out of 16 easy benchmarks and 9 out of 16 hard benchmarks. When combined with the white-space allocation method described next, successful routing of all 32 benchmarks is achieved. Routed wirelength for the combined flow is reduced by 11.6% on average and is shorter that that of all other leading tools.

Supply-Based Congestion Control via White Space Allocation. In the global placement of the proposed flow, the amount of white space in a region may not accurately match its routing demand. To further reduce congestion, hierarchical white-space allocation is applied after global placement, as part of legalization and detailed placement. A slicing tree is constructed based on the geometric locations of all cells. The congestion level at each node of the tree is calculated and cell positions are adjusted at each node in a top-to-bottom fashion in order to redistribute available white space to relieve congestion.

The slicing tree used in the proposed flow is similar to that in a partitioning-based global placement. However, cutlines are selected based on the geometric locations of the cells instead of the minimization of cut size. Cut directions are selected simply to keep aspect ratios of the resulting subregions suitably bounded. Every node in the tree maintains its cut direction, cut location, congestion, and total cell area as well as cell list. After the initial slicing tree construction, the congestion at each node in the tree is calculated from the bottom up. The congestion of a leaf node can be estimated by the total routing overflow of the grid cells contained in that leaf node. The congestion of an internal node is computed by a post-order traversal of the tree.

The cutlines of the slicing tree are then adjusted one at a time in top-down order. Each cutline is translated from the more congested of its two subregions toward the less congested one, so that the amounts of white space allocated to the sibling subregions are linearly proportional to their congestion levels. The movement of the cutline is expressed as a coordinate scaling, as described below. This scaling is applied to the cells of the subregions in order to redistribute them in a way that relieves congestion without changing their relative ordering within the parent subregion.

Consider a region r with lower-left corner (x_0, y_0), upper-right corner (x_1, y_1) and the original vertical cut direction at $x_{cut} = (x_0+x_1)/2$. The area of this region is $A_r = (x_1 - x_0)(y_1 - y_0)$. Assume that the total area of cells for left subregion r_0 and right subregion r_1 are S_0 and S_1, and the corresponding congestion levels are u_0 and u_1, respectively. To distribute the total amount of white space, which is $(A_r - S_0 - S_1)$, the amount of white space allocated to subregion r_0 is $(A_r - S_0 - S_1)\frac{u_0}{u_0+u_1}$. The new cutline location x'_{cut} can be derived as follows:

$$\gamma = \frac{S_0+(A_r-S_0-S_1)\frac{u_0}{u_0+u_1}}{A_r},$$
$$x'_{cut} = \gamma x_1 + (1-\gamma)x_0,$$

where γ is the ratio of the left subregion area to A_r after the cutline adjustment.

As an output of this step, a global placement that contains overlaps is obtained. Moreover, cells may not be placed along a row. The cutline adjustment approach is performed in the same spirit as fractional cut (Section 3.1.1), where horizontal cuts are not aligned with row boundaries. Figure 14(b) gives an example of this process. The total cell area and congestion at every tree node of the slicing tree are given. Cut lines are adjusted from top to bottom such that the white space in each subregion is proportional to its congestion level. After white space allocation, a local-swapping-based detailed placement is applied to obtain a legal placement.

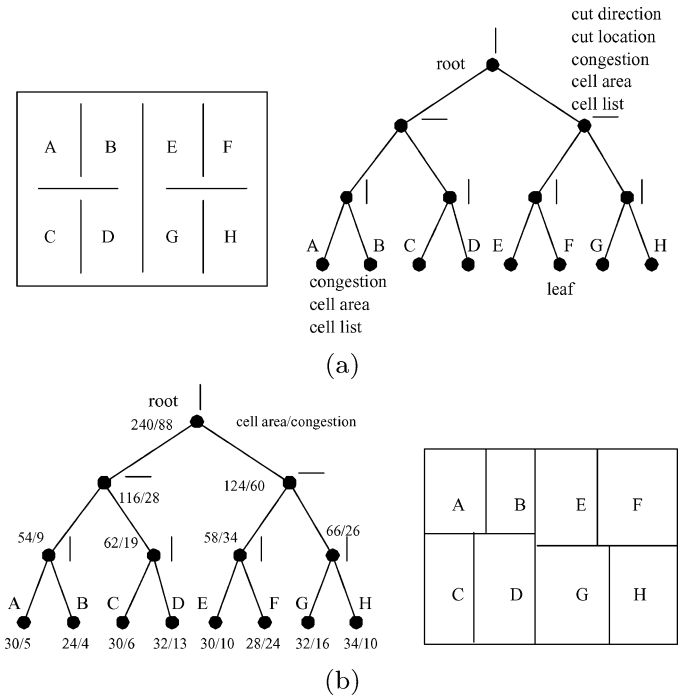


Fig. 14. (a) A slicing tree and its corresponding cut lines and regions. (b) A slicing tree after congestion estimation and regions after cut lines adjustment.

This simple white-space allocation scheme is remarkably effective. When used alone as part of detailed placement, it consistently reduces both routed wirelength and the number of overflowed global-routing bins. When combined with the topology-based congestion control in mPL-R as described above, it leads to higher completion rates and shorter routed wirelengths than can be achieved by any other leading tool.

6. CONCLUSION

Algorithms for large-scale circuit placement play a vital role in today's interconnect-limited nanometer designs. Recent studies suggest that the potential exists for a full technology generation's worth of performance gains in the placement step alone. In this article, we have reviewed the current state of the art, from the basic paradigms for scalable wirelength-driven placement to techniques for performance and routability optimization. We believe that hierarchical/multilevel methods are needed for scalability, and weighted wirelength minimization provides a general framework for performance and routability optimization in placement.

Ideally, systematic empirical comparisons would be used to understand the trade-offs of the different algorithms summarized in this article. However, direct numerical comparisons of these algorithms are difficult, partly due to limited accessiblity to these algorithms, and partly due to differences in their assumptions. Recently, comparisons based on wirelength minimization have been

attempted [Adya et al. 2003]. We are not aware of any comprehensive quantitative comparison in terms of performance or routability optimization. More work is needed to build a common framework for direct comparisons of different placement methods.

REFERENCES

ADYA, S., CHATURVEDI, S., ROY, J., PAPA, D. A., AND MARKOV, I. 2004. Unification of partitioning, placement, and floorplanning. In *Proceedings of the International Conference on Computer-Aided Design* (Nov.). 550–557.

ADYA, S., YILDIZ, M., MARKOV, I., VILLARRUBIA, P., PARAKH, P., AND MADDEN, P. 2003. Benchmarking for large-scale placement and beyond. In *Proceedings of the International Symposium on Physical Design*. 95–103.

ADYA, S. N. AND MARKOV, I. L. 2002. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proceedings of the International Symposium on Physical Design* (Apr.). 12–17.

AGNIHOTRI, A. R., YILDIZ, M. C., KHATKHATE, A., MATHUR, A., ONO, S., AND MADDEN, P. H. 2003. Fractional cut: Improved recursive bisection placement. In *Proceedings of the International Conference on Computer-Aided Design*. 307–310.

ALPERT, C. J. 1998. The ISPD98 circuit benchmark suite. In *Proceeding of the International Symposium on Physical Design*. 85–90.

ARROW, K., HURIWICZ, L., AND UZAWA, H. 1958. *Studies in Nonlinear Programming*. Stanford University Press, Stanford, Calif.

BETZ, V. AND ROSE, J. 1997. VPR: A new packing, placement, and routing tool for FPGA research. In *Proceedings of the International Workshop on FPL*. pp. 213–222.

BRANDT, A. 1986. Algebraic multigrid theory: The symmetric case. *Appl. Math. Comp. 19*, 23–56.

BRANDT, A. AND RON, D. 2002. Multigrid solvers and multilevel optimization strategies. *Multilevel Optimization and VLSICAD*. Kluwer Academic Publishers, Boston, Mass., Chap. 1.

BRENNER, U., PAULI, A., AND VYGEN, J. 2004. Almost optimum placement legalization by minimum cost flow and dynamic programming. In *Proceedings of the International Symposium on Physical Design*. 2–8.

BRENNER, U. AND ROHE, A. 2003. An effective congestion-driven placement framework. *IEEE Trans. CAD 22,* 4 (Apr.), 387–394.

BRENNER, U. AND ROHE, A. Apr 2002. An effective congestion-driven placement framework. In *Proceedings of the International Symposium on Physical Design*.

BREUER, M. 1977. Min-Cut Placement. *J. Design Automat. Fault Tolerant Comput. 1*, 4 (Oct.), 343–362.

BRIGGS, W., HENSON, V., AND MCCORMICK, S. 2000. *A Multigrid Tutorial*, 2nd ed. SIAM, Philadelphia, Pa.

BURSTEIN, M. AND YOUSSEF, M. N. 1985. Timing influenced layout design. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 124–130.

CADENCE DESIGN SYSTEMS, INC. 1999. QPlace version 5.1.55, compiled on 10/25/1999. Envisia ultra placer reference.

CALDWELL, A., KAHNG, A. B., AND MARKOV, I. 2000a. Improved algorithms for hypergraph partitioning. In *Proceedings of the Asia South Pacific Design Automation Conference.*

CALDWELL, A., KAHNG, A., AND MARKOV, I. 2000b. Can recursive bisection produce routable placements? In *Proceedings of the 37th IEEE/ACM Design Automation Conference*. ACM, New York, 477–482.

CALDWELL, A., KAHNG, A., AND MARKOV, I. 2000c. Iterative partitioning with varying node weights. *VLSI Design II*, 3, 249–258.

CALDWELL, A., KAHNG, A., AND MARKOV, I. 2000d. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Trans. on CAD 19*, 11, 1304–1314.

CHAN, T., CONG, J., KONG, T., AND SHINNERL, J. 2003a. Multilevel circuit placement. *Multilevel Optimization in VLSICAD.* Kluwer, Boston, Mass., Chap. 4.

CHAN, T., CONG, J., KONG, T., AND SHINNERL, J. 2000. Multilevel optimization for large-scale circuit placement. In *Proceedings of the IEEE International Conference on Computer-Aided Design* (San Jose, Calif.). IEEE Computer Society Press, Los Alamitos, Calif. 171–176.

CHAN, T., CONG, J., KONG, T., SHINNERL, J., AND SZE, K. 2003b. An enhanced multilevel algorithm for circuit placement. In *Proceedings of the IEEE International Conference on Computer Aided Design* (San Jose, Calif.). IEEE Computer Society Press, Los Alamitos, Calif.

CHAN, T., CONG, J., AND SZE, K. 2005. Multilevel generalized force-directed method for circuit placement. In *Proceedings of the International Symposium on Physical Design*.

CHANG, C.-C., CONG, J., PAN, D., AND YUAN, X. 2003a. Multilevel global placement with congestion control. *IEEE Trans. CAD 22*, 4 (Apr.), 395–409.

CHANG, C. C., CONG, J., AND XIE, M. 2003b. Optimality and scalability study of existing placement algorithms. In *Proceedings of the Asia South Pacific Design Automation Conference*. 621–627.

CHANG, C.-C., LEE, J., STABENFELDT, M., AND TSAY, R. S. 1994. A practical all-path timing-driven place and route design system. In *Proceedings of the Asia-Pacific Conference on Circuits and Systems*. 560–563.

CHEN, C., YANG, X., AND SARRAFZADEH, M. 2000. Potential slack: An effective metric of combinational circuit performance. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, New York, 198–201.

CHEN, H., CHENG, C.-K., CHOU, N.-C., KAHNG, A., MACDONALD, J., SUARIS, P., YAO, B., AND ZHU, Z. 2003. An algebraic multigrid solver for analytical placement with layout-based clustering. In *Proceedings of the IEEE/ACM Design Automation Conference*, ACM, New York, 794–799.

CHENG, C. AND KUH, E. 1984. Module placement based on resistive network optimization. *IEEE Trans. CAD. CAD-3*, 3.

CHENG, C.-L. E. 1994. RISA: Accurate and efficient placement routability modeling. In *Proceedings of the International Conference on Computer-Aided Design*, 690–695.

CHU, C. AND VISWANATHAN, N. 2004. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. In *Proceedings of the International Symposium on Physical Design* (Apr.). 26–33.

CONG, J. 2001. An interconnect-centric design flow for nanometer technologies. *Proc. IEEE 89*, 4 (Apr.), 505–527.

CONG, J. AND LIM, S. 2000. Edge separability based circuit clustering with application to circuit partitioning. In *Proceedings of the Asia South Pacific Design Automation Conference* (Yokohama Japan). 429–434.

CONG, J., ROMESIS, M., AND SHINNERL, J. 2005a. Fast floorplanning by look-ahead enabled recursive bipartitioning. In *Proceedings of the Asia Asia South Pacific Design Automation Conference.*

CONG, J., ROMESIS, M., AND SHINNERL, J. 2005b. Robust mixed-size placement by recursive legalized bipartitioning. Report 040057, Computer Science Dept., University of California, Los Angeles, Calif., `ftp://ftp.cs.ucla.edu/tech-report/2005-reports/040057.pdf`.

CONG, J., ROMESIS, M., AND XIE, M. Nov 2003a. Optimality and stability of timing-driven placement algorithms. In *Proceedings of the IEEE International Conference on Computer Aided Design*. (San Jose, Calif.). IEEE Computer Society Press, Los Alamitos, Calif.

CONG, J., ROMESIS, M., AND XIE, M. 2003b. Optimality, scalability and stability study of partitioning and placement algorithms. In *Proceedings of the International Symposium on Physical Design*. 88–94.

CONG, J., ROMESIS, M., AND XIE, M. 2004. UCLA Optimality Study Project. `http://cadlab.cs.ucla.edu/~pubbench`.

DUNLOP, A. AND KERNIGHAN, B. 1985. A procedure for placement of standard-cell VLSI circuits. *IEEE Trans. CAD. CAD-4*, 1 (Jan.).

DUNLOP, A. E., AGRAWAL, V. D., DEUTSCH, D. N., JUKL, M. F., KOZAK, P., AND WIESEL, M. 1984. Chip layout optimization using critical path weighting. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 133–136.

EISENMANN, H. AND JOHANNES, F. 1998. Generic global placement and floorplanning. In *Proceedings of the 35th ACM/IEEE Design Automation Conference*. ACM, New York, 269–274.

EVANS, L. C. 2002. *Partial Diferential Equations*. American Mathematical Society, Providence, R. I.

FIDUCCIA, C. M. AND MATTHEYSES, R. M. 1982. A linear-time heuristic for improving network partitions. In *Proceedings of the Design Automation Conference*, ACM, New York, 175–181.

GAO, T., VAIDYA, P. M., AND LIU, C. L. 1991. A new performance driven placement algorithm. In *IEEE/ACM International Conference on Computer-Aided Design*. ACM, New York, 44–47.

GOLUB, G. AND VAN LOAN, C. 1989. *Matrix Computations*, 2nd ed. The Johns Hopkins University Press, Baltimore, Md.

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

HAGEN, L., HUANG, J. H., AND KAHNG, A. B. 1997. On implementation choices for iterative improvement partitioning algorithms. *IEEE Trans. CAD 16*, 10, 1199–1205.

HAGEN, L. W., HUANG, D. J.-H., AND KAHNG, A. B. 1995. Quantified suboptimality of VLSI layout heuristics. In *Proceedings of the Design Automation Conference*. ACM, New York, 216–221.

HALPIN, B., CHEN, C., AND SEHGAL, N. 2001. Timing driven placement using physical net constraints. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 780–783.

HAMADA, T., CHENG, C. K., AND CHAU, P. M. 1993. Prime: A timing-driven placement tool using a piecewise linear resistive network approach. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 531–536.

HAUGE, P. S., NAIR, R., AND YOFFA, E. J. 1987. Circuit placement for predictable performance. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ACM, New York, 88–91.

HU, B. AND MAREK-SADOWSKA, M. 2002. Congestion minimization during placement without estimation. In *Proceedings of the International Conference on Computer-Aided Design*. 739–745.

HU, B. AND MAREK-SADOWSKA, M. 2003. Wire length prediction based clustering and its application in placement. In *Proceedings of the Design Automation Conference*.

HU, B. AND MAREK-SADOWSKA, M. 2004. Fine granularity clustering based placement. *IEEE Trans. CAD 23,* 1264–1276.

HUR, S.-W. AND LILLIS, J. 2000. Mongrel: Hybrid techniques for standard-cell placement. In *Proceedings of the IEEE International Conference on Computer-Aided Design* (San Jose, Calif., Nov.). IEEE Computer Society Press, Los Alamitos, Calif., 165–170.

JACKSON, M. AND KUH, E. S. 1989. Performance-driven placement of cell based IC's. In *Proceedings of ACM/IEEE Design Automation Conference*. ACM, New York, 370–375.

KAHNG, A. AND REDA, S. 2004. Placement feedback: A concept and method for better min-cut placements. In *Proceedings of ACM/IEEE Design Automation Conference*. ACM, New York, 357–362.

KAHNG, A. AND WANG, Q. 2004. Implementation and extensibility of an analytic placer. In *Proceedings of the International Symposium on Physical Design*. 18–25.

KARYPIS, G. 1999. Multilevel algorithms for multi-constraint hypergraph partitioning. Tech. Rep. 99-034, Department of Computer Science, University of Minnesota, Minneapolis, Minn.

KARYPIS, G. 2003. Multilevel hypergraph partitioning. *Multilevel Optimization and VLSICAD*. Kluwer Academic Publishers, Boston, Mass., Chap. 3.

KHATKHATE, A., LI, C., AGNIHOTRI, A. R., ONO, S., YILDIZ, M. C., KOH, C.-K., AND MADDEN, P. H. 2004. Recursive bisection based mixed block placement. In *Proceedings of the International Symposium on Physical Design*.

KLEINHANS, J., SIGL, G., JOHANNES, F., AND ANTREICH, K. 1991. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. CAD CAD-10*, 356–365.

KONG, T. 2002. A novel net weighting algorithm for timing-driven placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, New York, 172–176.

LI, C. AND KOH, C.-K. 2003. On improving recursive bipartitioning-based placement. Tech. Rep. TR-ECE 03-14, Purdue University.

LI, C., XIE, M., KOH, C., CONG, J., AND MADDEN, P. 2004. Routability-driven placement and white space allocation. In *Proceedings of the International Conference on Computer-Aided Design*. 394–401.

LOU, J., THAKUR, S., KRISHNAMOORTHY, S., AND SHENG, H. 2002. Estimating routing congestion using probabilistic analysis. *IEEE Trans. CAD 21*, 1 (Jan.), 32–41.

LUK, W. K. 1991. A fast physical constraint generator for timing driven layout. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 626–631.

MAREK-SADOWSKA, M. AND LIN, S. P. 1989. Timing driven placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, New York 94–97.

MARQUARDT, A., BETZ, V., AND ROSE, J. 2000. Timing-driven placement for FPGAs. In *Proceedings of the ACM Symposium on FPGAs*. ACM, New York, 203–213.

MAYRHOFER, S. AND LAUTHER, U. 1990. Congestion-driven placement using a new multi-partitioning heuristic. In *Proceedings of the International Conference on Computer-Aided Design*. 332–335.

MORTON, K. W. AND MAYERS, D. F. 1994. *Numerical Solution of Partial Differential Equations*. Cambridge University Press.

NAIR, R., BERMAN, C. L., HAUGE, P., AND YOFFA, E. J. 1989. Generation of performance constraints for layout. *IEEE Trans. CAD Integ. Circ. Syst. 8*, 8, 860–874.

NAYLOR, W. C., DONELLY, R., AND SHA, L. 2001. Nonlinear optimization system and method for wire length and delay optimization for an automatic electric circuit placer.

ONO, S. AND MADDEN, P. 2005. On structure and suboptimality in placement. In *Proceedings of the Asia South Pacific Design Automation Conference*.

PARAKH, P. N., BROWN, R. B., AND SAKALLAH, K. A. 1998. Congestion driven quadratic placement. In *Proceedings of the Design Automation Conference*. 275–278.

QUINN, N. AND BREUER, M. 1979. A force-directed component placement procedure for printed circuit boards. *IEEE Trans. Circ Syst CAS CAS-26*, 377–388.

RAMACHANDARAN, P., ONO, S., AGNIHOTRI, A., DAMODARA, P., SRIHARI, H., AND MADDEN, P. 2005. Optimal placement by branch-and-price. In *Proceedings of the Asia South Pacific Design Automation Conference*. (Jan.).

REN, H., PAN, D., AND KUNG, D. 2004. Sensitivity guided net weighting for placement driven synthesis. In *Proceedings of the International Symposium on Physical Design*. 10–17.

SAAD, Y. 1996. *Iterative Methods for Sparse Linear Systems*. PWS publishing, Pacific Grove, Calif.

SANKAR, Y. AND ROSE, J. 1999. Trading quality for compile time: Ultra-fast placement for FPGAs. In *FPGA '99, ACM Symposium on FPGAs*. ACM, New York, 157–166.

SARRAFZADEH, M., KNOL, D. A., AND TELLEZ, G. E. 1997a. A delay budgeting algorithm ensuring maximum flexibility in placement. *IEEE Trans. CAD of Integ. Circ. Syst. 16*, 11, 1332–1341.

SARRAFZADEH, M., KNOL, D. A., AND TELLEZ, G. E. 1997b. Unification of budgeting and placement. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 758–761.

SARRAFZADEH, M., WANG, M., AND YANG, X. 2002. *Modern Placement Techniques*. Kluwer, Boston, Mass.

SENN, M., SEIDL, U., AND JOHANNES, F. 2002. High quality deterministic timing driven FPGA placement. In *Proceedings of the ACM Symposium on FPGAs*. ACM, New York.

SIGL, G., DOLL, K., AND JOHANNES, F. 1991. Analytical placement: A linear or a quadratic objective function? In *Proceedings of the 28th ACM/IEEE Design Automation Conference*. ACM, New York, 427–432.

SRINIVASAN, A., CHAUDHARY, K., AND KUH, E. S. 1991. RITUAL: A performance driven placement for small-cell ICs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, New York, 48–51.

SWARTZ, W. AND SECHEN, C. 1995. Timing-driven placement for large standard cell circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 211–215.

TELLEZ, G. E., KNOL, D. A., AND SARRAFZADEH, M. 1996. A performance-driven placement technique based on a new net budgeting criterion. In *Proceedings of the International Symposium on Circuits and Systems*. 504–507.

TSAY, R., KUH, E., AND HSU, C. 1988. Proud: A fast sea-of-gates placement algorithm. *IEEE Desi. Test Comput*. 44–56.

TSAY, R. S. AND KOEHL, J. 1991. An analytic net weighting approach for performance optimization in circuit placement. In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM, New York, 620–625.

VORWERK, K., KENNINGS, A., AND VANNELLI, A. 2004. Engineering details of a stable force-directed placer. In *Proceedings of the International Conference on Computer-Aided Design*. 573–580.

VYGEN, J. 1997. Algorithms for large-scale flat placement. In *Proceedings of the 34th ACM/IEEE Design Automation Conference*. ACM, New York, 746–751.

VYGEN, J. 2000. Four-way partitioning of two-dimensional sets. Report 00900-OR, Research Institute for Discrete Mathematics, University of Bonn, Bonn, Germany.

WANG, Q., JARIWALA, D., AND LILLIS, J. 2005. A study of tighter lower bounds in LP relaxation-based placement. In *Proceedings of the Great Lakes Symposium on VLSI*. To appear.

WANG, M., YANG, X., AND SARRAFZADEH, M. 2000. Dragon2000: Standard-cell placement tool for large circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ACM, New York, 260–263.

WESTRA, J., BARTELS, C., AND GROENEVELD, P. 2004. Probabilistic congestion prediction. In *Proceedings of the International Symposium on Physical Design*. 204–209.

XIU, Z., MA, J., FOWLER, S., AND RUTENBAR, R. 2004. Large-scale placement by grid warping. In *Proceedings of the Design Automation Conference*. 351–356.

YANG, X., CHOI, B., AND SARRAFZADEH, M. 2002a. Timing-driven placement using design hierarchy guided constraint generation. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, New York, 177–180.

YANG, X., CHOI, B., AND SARRAFZADEH, M. 2003. Routability-driven white space allocation for fixed-die standard-cell placement. *IEEE Trans, CAD 22*, 4 (Apr.), 410–419.

YANG, X., KASTNER, R., AND SARRAFZADEH, M. 2002b. Congestion estimation during top-down placement. *IEEE Trans. CAD 21*, 1 (Jan.), 72–80.

YILDIZ, M. AND MADDEN, P. 2001a. Global objectives for standard cell placement. In *Proceedings of the 11th Great-Lakes Symposium on VLSI*. 68–72.

YILDIZ, M. AND MADDEN, P. 2001b. Improved cut sequences for partitioning-based placement. In *Proceedings of the Design Automation Conference*. 776–779.

YOUSSEF, H., LIN, R. B., AND SHRAGOWITZ, S. 1992. Bounds on net delays. *IEEE Trans. Circ. Syst. 39*, 11, 815–824.

YOUSSEF, H. AND SHRAGOWITZ, E. 1990. Timing constraints for correct performance. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ACM, New York, 24–27.

Received October 2004; revised January 2005; accepted January 2005