Large-Scale Circuit Placement: Gap and Promise *

Jason Cong, Tim Kong,† Joseph R. Shinnerl, Min Xie, and Xin Yuan
UCLA Computer Science Department {cong,shinnerl,xie,yuanxin}@cs.ucla.edu
†Magma Design Automation kongtm@magma-da.com

ABSTRACT

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

Keywords

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. Although complex logic functions can be composed in a hierarchical fashion following the logical hierarchy, recent studies [22] 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 [32, 16]. In this section, we summarize the results from these studies.

2.1 Placement Examples with Known Optima

Recently, four suites of placement examples with known optimal wirelength (PEKO) were constructed [32, 16]. 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 $D(i)$, (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 [3] 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 [1].

All the nets in PEKO are local, i.e., the wirelength of every net has the minimum possible value. However, in real

*Financial support from the Semiconductor Research Consortium under contracts 98-DJ-605, 98-TJ-686, and 2001-TJ-910 and from the National Science Foundation under grant CCR-0096383 is gratefully acknowledged.

ICCAD '03 San Jose, California USA

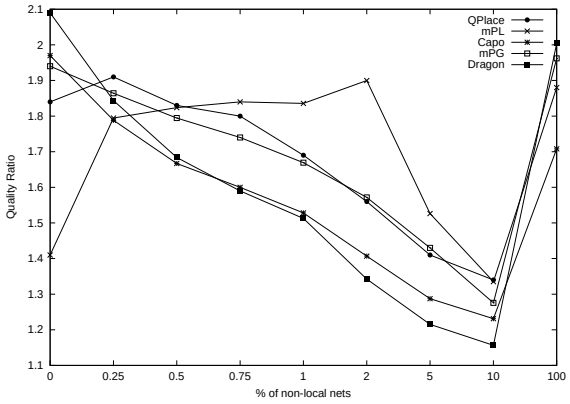

Figure 1: Average solution quality vs percentage of non-local nets, from PEKO (0% non-local nets) through PEKU (0.25% to 10% of non-local nets) to G-PEKU (100% non-local nets). Each data point is an average quality ratio for a given placer over all circuits in the given suite.

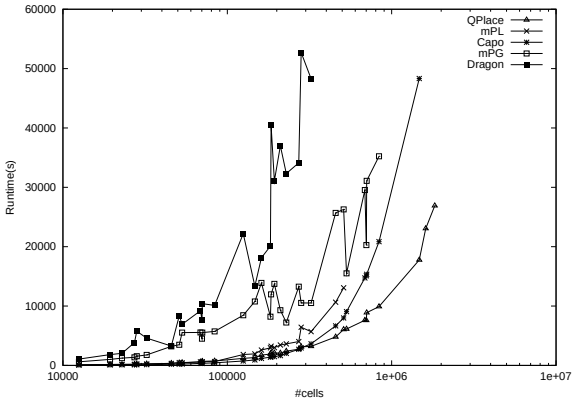

Figure 2: Run time vs. cell number for several algorithms on the PEKO suite.

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 [24]. 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 Examples with Known Upper bounds on wirelength) consist of both PEKO-style local nets and additional, randomly generated non-local nets. An upper bound on the optimal wirelength is derived simply by adding the wirelengths of non-local nets to the known total wirelength of the local nets. In the study [24], the percentage of non-local nets was gradually increased from 0.25% to 10%. The G-PEKU and PEKU suites are also available online [1].

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 [62], Capo v.8.5 [12], mPL v.2.0 [14], mPG v.1.0 [15], and QPlace v.5.1.55 [10]. 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 non-local nets. Figure 2 shows how the run times of these tools changes with increase in cell number. We make the following three observations.

(i) None of the placers achieves a quality ratio close to 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 non-local 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.

It is not known whether the gaps on real circuits are similar to those observed on the benchmarks discussed above. The construction of placement examples that resemble real circuits more closely, including examples optimized for timing [25] or routability, is an active area of research.

3. SCALABLE PARADIGMS

We assert that scalability without some hierarchical form of computation is impossible. The use of hierarchy may be subtle or indirect, but never completely absent. In this paper, we use scalability in its traditional sense and therefore

consider not just $\mathcal{O}(N)$ algorithms but rather any framework likely to have applicability lasting for several technology generations.

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, bottom-up, or flat. Top-down algorithms (Section 3.1) use variants of recursive partitioning. Bottom-up approaches (Section 3.2) use variants of recursive clustering and are known as *multilevel* methods. 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 circuit partitioning in some way. Seminal work on partitioning-based placement was done by Breuer [7] and Dunlop and Kernighan [27]. Most contemporary methods have exploited further advances in fast algorithms to push these frameworks beyond their original capabilities. Fast, high-quality $\mathcal{O}(N)$ partitioning algorithms give top-down partitioning attractive $\mathcal{O}(N \log N)$ scalability overall.

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.

Connections between subregions can be modeled by *terminal propagation* [27], 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 [62, 67]. 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 [67]. 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 [66].

3.1.2 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 [20, 59] and subsequently refined by Gordian [41], 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, e.g., 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 [55]. 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.

BonnPlace [61, 6] is the leading contemporary variation of this framework. It employs a sophisticated, novel, and linear-time *displacement-minimizing* partitioning instead of cutsize minimization during subregion assignment. Instead of introducing explicit equality constraints into the analytical minimization, the quadratic-wirelength objective is altered to minimize cells' displacements from their assigned subregions.

3.1.3 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 [61, 6], 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 [12] greedily selects cell orientations in order to reduce wirelength and improve routability. Feng Shui [66] 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 [11].

In Dragon [62, 53], 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 closely resembles the multilevel methods discussed next.

3.2 Multilevel Methods

Placement algorithms in the multilevel paradigm have only recently drawn attention [50, 13, 15, 14, 26]. These methods are based on coarsening, relaxation, and interpolation, defined as follows.

(i) **Coarsening.** Hierarchies are built from the bottom up by recursive aggregation, i.e., clustering or extensions.

(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.

The scalability of this 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

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 [40, 26] may be more effective than more sophisticated ideas like edge-separability clustering (ESC) [23] 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 [37].

To reduce the likelihood of poorly chosen clusters, the notion of a cluster can be relaxed, as in the algebraic multigrid framework [8]. 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*.

3.2.2 Initial Placement at Coarsest Level

Following recursive aggregation, a placement at the coarsest 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 [13, 26, 14] uses nonlinear programming; mPG uses simulated annealing [15]. 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 [4, 14].

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 [38] and (ii) randomized, greedy, and discrete Goto-based cell swapping [31].

3.2.4 Interpolation

Simple declustering and linear assignment can be effective [13]. With this approach, each component cluster is initially placed at the center of it (single) 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, e.g., 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 [14]. 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.

A constructive approach, as in Ultrafast VPR [50], 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.

3.3 Embedded Multilevel Optimization

Most leading methods owe their performance not just to external design but also to sophisticated and hierarchical iterative internal calculation. 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. A clear demonstration of this approach is the recent application of AMG-based linear-system solvers to iterated force-directed quadratic-wirelength minimization [19, 29]. A quadratic 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. 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 be reused over several iterations.

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. [17] 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 the following. (i) Formulate the problem as a linear or nonlinear programming problem by introducing auxiliary variables (i.e., *arrival time*) at circuit nodes [39, 56, 34]. Different mathematical programming techniques can then be used to solve the problem. (ii) Explicitly minimize the length of a set of critical paths. This set of critical paths can be pre-computed in a static manner or dynamically adjusted from iteration to iteration. TimberWolf [57] used simulated annealing to minimize a set of pre-specified timing-critical paths, while mathematical programming techniques [9, 45] 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, e.g., top-down partitioning, it is very difficult or infeasible to maintain an accurate global timing view.

4.2 Net-based Algorithms

Net-based algorithms [28, 48, 60, 29], on the contrary, do not directly enforce path-based constraints. Instead, timing constraints or requirements on paths are transformed into either length constraints or weights on individual nets. 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* [35, 30, 44, 68, 58, 52, 51, 18]. 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 [48, 69, 18]. 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. As a result, it may severely overconstrain the placement problem. Recently, some attempts have been made to unify delay budgeting and placement [51, 63, 33], where a complete or coarse [63, 33] 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 naturally integrated 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.

Unfortunately, despite these 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 [46] 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, α is a constant.

These methods ignored another important principle – *path sharing*. In generally, 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, this naive method suffers from a severe drawback: it cannot distinguish timing-critical paths from non-critical paths. The variant ϵ-network path counting [54] suffers from the same problem [42].

A recent work [42] proposed a nice solution. 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 was shown [42] 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. Yet such computation can be carried out very efficiently in linear time, and experimental results have confirmed its effectiveness. Compared with VPR [46] under the same placement framework, it reduced the longest path delay by 15.6% on average with no runtime overhead and only a 4.1% increase in total wirelength.

A potential problem in the 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 n-th iteration; more critical nets will 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 non-critical nets are potentially increased, resulting in changes in net criticalities, and, thus, in net weights. Therefore, it is important to ensure convergence of weighted-wirelength optimization.

Note that certain path-based approaches suffer from similar problems, e.g., a need to dynamically adjust the set of paths being optimized [57].

Two ways to solve this problem have appeared in the literature. The first approach is to perform timing analysis and recompute net weighting periodically. VPR [46] and PATH [42] follow this approach. Based on simulated annealing, both methods perform timing analysis and net reweighting once per temperature. The second approach is to make use of historic information [29], i.e., 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 decrease its weight.

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 in order to achieve the best overall performance.

Routability-driven placement involves mainly (i) routability modeling and (ii) solution 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 each boundary of the routing grid structure.

There have been many studies on routability modeling. There are two major categories: topology-free modeling (TP-free), where no explicit routing is done, and topology-based modeling (TP-based), where routing trees are explicitly constructed on some routing grid.

TP-free modeling is faster in general. Examples of this class include bounding-box (BBOX)-based modeling [21], probabilistic analysis-based modeling [43], Rent's rule-based modeling [65], and pin density-based modeling [5]. In RISA modeling [21], the wiring supply is modeled based on the pre-wiring, cells and mega cells, and the wiring demand of a net is modeled by a weighted BBOX length. A net-based stochastic model for 2-pin nets is presented to compute expected horizontal and vertical track usage with consideration of routing blockage [43]. Peak routing demand and regional routing demand are estimated using Rent's rule [65]. Pin density per bin can be used as a metric for intrabin routing congestion, but it can not model the interbin boundary congestion. Therefore, it is combined with probabilistic analysis-based modeling for completeness [5].

In a TP-based modeling method, for each net, a Steiner tree topology is generated on the given routing grid. Such a modeling method can generate at least a global routing solution, i.e., it can provide a upper bound for the 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, a precomputed Steiner tree topology on a few grid structures is used for wiring-demand estimation [47]. Two algorithms of logarithmic complexity have recently been proposed: a fast congestion-avoidance two-bend routing algorithm, LZ-router, for topology generation for two-pin nets, and IncA-tree algorithm, which can support incremental updates for building a rectilinear Steiner arborescence tree (A-tree) for a multipin net [15].

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 bin weights and optimizes weighted wirelength. It can easily be incorporated into iterative placement algorithms such as simulated-annealing-based methods [36, 15].

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 [49], quadrisection-based placers [5], as well as iterative placement techniques, such as simulated annealing-based placers [64].

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 paper, 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 paper. 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 [2]. 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.

7. REFERENCES

[1] http://cadlab.cs.ucla.edu/∼pubbench.

[2] S. N. Adya, M. Yildiz, I. L. Markov, P. G. Villarrubia, P. N. Parakh, and P. H. Madden. Benchmarking for large-scale placement and beyond. In *Proc. Intl. Symp. on Physical Design (ISPD)*, pages 95–103, 2003.

[3] C. J. Alpert. The ispd98 circuit benchmark suite. In *Proc. International Symposium on Physical Design*, pages 85–90, 1998.

[4] A. Brandt and D. Ron. *Multigrid Solvers and Multilevel Optimization Strategies*, chapter 1 of *Multilevel Optimization and VLSICAD*. Kluwer Academic Publishers, Boston, 2002.

[5] U. Brenner and A. Rohe. An effective congestion-driven placement framework. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(4):387–394, April 2003.

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

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

[8] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, Philadelphia, second edition, 2000.

[9] M. Burstein and M. N. Youssef. Timing influenced layout design. In *Proc. ACM/IEEE Design Automation Conference*, pages 124–130, 1985.

[10] Cadence Design Systems, Inc. Envisia ultra placer reference. QPlace version 5.1.55, compiled on 10/25/1999.

[11] A. Caldwell, A.B.Kahng, and I. Markov. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Trans. on CAD*, 19(11):1304–1314, 2000.

[12] A. Caldwell, A. Kahng, and I. Markov. Can recursive bisection alone produce routable placements? In *Proc. 37th IEEE/ACM Design Automation Conf.*, 2000.

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

[14] T. Chan, J. Cong, T. Kong, J. Shinnerl, and K. Sze. An enhanced multilevel algorithm for circuit placement. In *Proc. IEEE International Conference on Computer Aided Design*, San Jose, CA, Nov 2003.

[15] C.-C. Chang, J. Cong, D. Pan, and X. Yuan. Multilevel global placement with congestion control. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(4):395–409, April 2003.

[16] C. C. Chang, J. Cong, and M. Xie. Optimality and scalability study of existing placement algorithms. In *Proc. Asia South Pacific Design Automation Conference*, pages 621–627, 2003.

[17] C.-C. Chang, J. Lee, M. Stabenfeldt, and R. S. Tsay. A practical all-path timing-driven place and route design system. In *Proc. Asia-Pacific Conference on Circuits and Systems*, pages 560–563, 1994.

[18] C. Chen, X. Yang, and M. Sarrafzadeh. Potential slack: An effective metric of combinational circuit performance. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 198–201, 2000.

[19] H. Chen, C.-K. Cheng, N.-C. Chou, A. Kahng, J. MacDonald, P. Suaris, B. Yao, and Z. Zhu. An algebraic multigrid solver for analytical placement with layout-based clustering. In *Proc. IEEE/ACM Design Automation Conf.*, pages 794–799, 2003.

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

[21] C.-L. E. Cheng. RISA: accurate and efficient placement routability modeling. In *Proc. Int. Conf. on Computer Aided Design*, pages 690–695, Nov. 1994.

[22] J. Cong. An interconnect-centric design flow for nanometer technologies. *Proceedings of the IEEE*, 89(4):505–527, April 2001.

[23] J. Cong and S. K. Lim. Edge separability based circuit clustering with application to circuit partitioning. In *Asia South Pacific Design Automation Conference, Yokohama Japan*, pages 429–434, 2000.

[24] J. Cong, M. Romesis, and M. Xie. Optimality, scalability and stability study of partitioning and placement algorithms. In *Proc. International Symposium on Physical Design*, pages 88–94, 2003.

[25] J. Cong, M. Romesis, and M. Xie. Optimality and stability of timing-driven placement algorithms. In *Proc. IEEE International Conference on Computer Aided Design*, San Jose, CA, Nov 2003.

[26] J. Cong and J. R. Shinnerl, editors. *Multilevel Optimization in VLSICAD*. Kluwer Academic Publishers, Boston, 2003.

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

[28] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In *Proc. ACM/IEEE Design Automation Conference*, pages 133–136, 1984.

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

[30] T. Gao, P. M. Vaidya, and C. L. Liu. A new performance driven placement algorithm. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 44–47, 1991.

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

[32] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng. Quantified suboptimality of vlsi layout heuristics. In *Proc. Design Automation Conference*, pages 216–221, 1995.

[33] B. Halpin, C. Chen, and N. Sehgal. Timing driven placement using physical net constraints. In *Proc. ACM/IEEE Design Automation Conference*, pages 780–783, 2001.

[34] T. Hamada, C. K. Cheng, and P. M. Chau. Prime: a timing-driven placement tool using a piecewise linear resistive network approach. In *Proc. ACM/IEEE Design Automation Conference*, pages 531–536, 1993.

[35] P. S. Hauge, R. Nair, and E. J. Yoffa. Circuit placement for predictable performance. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 88–91, 1987.

[36] B. Hu and M. Marek-Sadowska. Congestion minimization during placement without estimation. In *Proc. Int. Conf. on Computer Aided Design*, pages 739–745, Nov. 2002.

[37] B. Hu and M. Marek-Sadowska. Fine granularity clustering for large-scale placement problems. In *Proc. Design Automation Conference*, pages 67–74, Jun. 2003.

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

[39] M. Jackson and E. S. Kuh. Performance-driven placement of cell based IC's. In *Proc. ACM/IEEE Design Automation Conference*, pages 370–375, 1989.

[40] G. Karypis. *Multilevel Hypergraph Partitioning*, chapter 3 of *Multilevel Optimization and VLSICAD*. Kluwer Academic Publishers, Boston, 2002.

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

[42] T. Kong. A novel net weighting algorithm for timing-driven placement. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 172–176, 2002.

[43] J. Lou, S. Thakur, S. Krishnamoorthy, and H. Sheng. Estimating routing congestion using probabilistic analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(1):32–41, January 2002.

[44] W. K. Luk. A fast physical constraint generator for timing driven layout. In *Proc. ACM/IEEE Design Automation Conference*, pages 626–631, 1991.

[45] M. Marek-Sadowska and S. P. Lin. Timing driven placement. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 94–97, 1989.

[46] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *ACM Symposium on FPGAs*, pages 203–213, 2000.

[47] S. Mayrhofer and U. Lauther. Congestion-driven placement using a new multi-partitioning heuristic. In *Proc. Int. Conf. on Computer Aided Design*, pages 332–335, 1990.

[48] R. Nair, C. L. Berman, P. Hauge, and E. J. Yoffa. Generation of performance constraints for layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):860–874, 1989.

[49] P. N. Parakh, R. B. Brown, and K. A. Sakallah. Congestion driven quadratic placement. In *Proc. Design Automation Conference*, pages 275–278, 1998.

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

[51] M. Sarrafzadeh, D. A. Knol, and G. E. Tellez. A delay budgeting algorithm ensuring maximum flexibility in placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(11):1332–1341, 1997.

[52] M. Sarrafzadeh, D. A. Knol, and G. E. Tellez. Unification of budgeting and placement. In *Proc. ACM/IEEE Design Automation Conference*, pages 758–761, 1997.

[53] M. Sarrafzadeh, M. Wang, and X. Yang. *Modern Placement Techiques*. Kluwer Academic Publishers, Boston, 2002.

[54] M. Senn, U. Seidl, and F. Johannes. High quality deterministic timing driven FPGA placement. In *ACM Symposium on FPGAs*, 2002.

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

[56] A. Srinivasan, K. Chaudhary, and E. S. Kuh. RITUAL: A performance driven placement for small-cell ICs. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 48–51, 1991.

[57] W. Swartz and C. Sechen. Timing-driven placement for large standard cell circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 211–215, 1995.

[58] G. E. Tellez, D. A. Knol, and M. Sarrafzadeh. A performance-driven placement technique based on a new net budgeting criterion. In *International Symposium on Circuits and Systems*, pages 504–507, 1996.

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

[60] R. S. Tsay and J. Koehl. An analytic net weighting approach for performance optimization in circuit placement. In *Proc. ACM/IEEE Design Automation Conference*, pages 620–625, 1991.

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

[62] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large circuits. *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pages 260–263, Apr 2000.

[63] X. Yang, B. Choi, and M. Sarrafzadeh. Timing-driven placement using design hierarchy guided constraint generation. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 177–180, 2002.

[64] X. Yang, B.-K. Choi, and M. Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(4):410–419, April 2003.

[65] X. Yang, R. Kastner, and M. Sarrafzadeh. Congestion estimation during top-down placement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(1):72–80, January 2002.

[66] M. C. Yildiz and P. H. Madden. Global objectives for standard cell placement. In *Eleventh Great-Lakes Symposium on VLSI*, pages 68–72, 2001.

[67] M. C. Yildiz and P. H. Madden. Improved cut sequences for partitioning-based placement. In *Proc. Design Automation Conference*, pages 776–779, 2001.

[68] H. Youssef, R. B. Lin, and S. Shragowitz. Bounds on net delays. *IEEE Transactions on Circuits and Systems*, 39(11):815–824, 1992.

[69] H. Youssef and E. Shragowitz. Timing constraints for correct performance. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 24–27, 1990.