Implementation and Extensibility of an Analytic Placer

Andrew B. Kahng, *Member, IEEE,* and Qinke Wang, *Student Member, IEEE*

***Abstract*—Automated cell placement is a critical problem in very large scale integration (VLSI) physical design. New analytical placement methods that simultaneously spread cells and optimize wirelength have recently received much attention from both academia and industry. A novel and simple objective function for spreading cells over the placement area is described in the patent of Naylor *et al.* (U.S. Pat. 6301693). When combined with a wirelength objective function, this allows efficient simultaneous cell spreading and wirelength optimization using nonlinear optimization techniques. In this paper, we implement an analytic placer (APlace) according to these ideas (which have other precedents in the open literature), and conduct in-depth analysis of characteristics and extensibility of the placer. Our contributions are as follows. 1) We extend the objective functions described in (Naylor *et al.*, U.S. Patent 6301693) with congestion information and implement a top-down hierarchical (multilevel) placer (APlace) based on them. For IBM-ISPD04 circuits, the half-perimeter wirelength of APlace outperforms that of FastPlace, Dragon, and Capo, respectively, by 7.8%, 6.5%, and 7.0% on average. For eight IBM-PLACE v2 circuits, after the placements are detail-routed using Cadence WRoute, the average improvement in final wirelength is 12.0%, 8.1%, and 14.1% over QPlace, Dragon, and Capo, respectively. 2) We extend the placer to address mixed-size placement and achieve an average of 4% wirelength reduction on ten ISPD'02 mixed-size benchmarks compared to results of the leading-edge solver, FengShui. 3) We extend the placer to perform timing-driven placement. Compared with timing-driven industry tools, evaluated by commercial detailed routing and static timing analysis, we achieve an average of 8.4% reduction in cycle time and 7.5% reduction in wirelength for a set of six industry testcases. 4) We also extend the placer to perform input/output-core coplacement and constraint handing for mixed-signal designs. Our paper aims to, and empirically demonstrates, that the APlace framework is a general, and extensible platform for "spatial embedding" tasks across many aspects of system physical implementation.**

***Index Terms*—Analytic placement, APlace, mixed-size, very large scale integration (VLSI).**

I. Introduction

AUTOMATED cell placement is a critical problem in very large scale integration (VLSI) design. As deep submicron technology scales, new challenges arise for placement tools: design sizes are larger, turnaround times are shorter, and a variety of additional physical and geometrical constraints must be fulfilled simultaneously.

A common placement formulation seeks to minimize wirelength under the constraint that cells do not overlap each other. The current state-of-the-art placement tools can be classified into two categories, based on how they obtain a placement without cell overlaps [15]. The first class consists of algorithms that refine the existing placement to obtain a better overlap-free placement. For example, TimberWolf [52] is a well-known annealing-based placement tool; it develops new placements by permuting an existing placement. The second class of algorithms uses top-down recursive partitioning to provide the necessary cell spreading. Within this approach, a min-cut objective [4], [32] has been successfully used. A number of placement tools fall into this category: recursive partition with min-cut objective [8], [23], [54], [60], (Capo, CPlace, FengShui), quadratic placement [37], [53], [56] (GORDIAN, PROUD), and analytic placement with linear wirelength [51] (GORDIAN-L). Recently, the Dragon [57], [58] placement tool was presented, combining annealing with recursive bisection. Constructive methods based on (hybrids of) partitioning and analytical techniques are usually fast and produce good results. However, wirelength minimization may come at the cost of routability, or the inability to handle hard constraints.

New analytical placement methods that simultaneously spread cells and optimize wirelength have recently received much attention from both academia and industry. In such methods, forces based on the current cell distribution are applied to iteratively reduce cell overlaps. A quadratic objective combining wirelength and additional forces is proposed by [15]; in each iteration, a better spreading of cells is achieved without excessive net stretching. Another model of cell attracting and repelling (ARP) is presented in [16] and [17]. Attractors (dummy cells) are added to sparse regions to drag cells from nearby dense regions, together with a cell repeller model that captures the wirelength objective. The ideas of additional forces and fixed pseudocells are also combined in [24]. The difficulty of force and attractor-directed placement methods is that wirelength is easily damaged by improper forces and attractors. Recently, a fast placement algorithm with good quality was presented in [55]. Quadratic wirelength optimization and a cell shifting technique are iteratively applied to obtain a high-quality placement without cell overlapping. After quadratic optimization, a special cell-shifting technique is used to reduce cell overlapping, and pseudopins and nets are added to hold cells from clustering again in the next quadratic optimization process.

Manuscript received June 15, 2004; revised October 12, 2004. This work was supported in part by Cadence Design Systems, Inc., in part by the California MICRO program, in part by the MARCO Gigascale Systems Research Center, in part by the National Science Foundation under Award MIP-9987678, and in part by the Semiconductor Research Corporation. A preliminary version of this work has appeared in the *Proc. Int. Symp. Phys. Design*, 2004, Phoenix, AZ, pp. 18–25 and the *Proc. Int. Conf. Computer-Aided Design*, 2004, San Jose, CA, pp. 565–572. This paper was recommended by Guest Editor L. Scheffer.

A. B. Kahng is with the Departments of Computer Science and Engineering and Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@cs.ucsd.edu).

Q. Wang is with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: qiwang@cs.ucsd.edu).

Digital Object Identifier 10.1109/TCAD.2005.846366

0278-0070/$20.00 © 2005 IEEE

A novel and simple objective function for spreading cells over the placement area was proposed in the recent patent of Naylor *et al.* [44]. Combined with a wirelength objective function, it allows efficient simultaneous cell spreading and wirelength optimization using nonlinear optimization techniques. The focus of our present paper is the implementation of an analytic placer (APlace) according to this idea, followed by in-depth analysis of characteristics and extensibility of the placer. While we implement the placer relying largely on the description in [44], it should be noted that this method integrates quite a few ideas that have been published in the open literature, as we describe in detail in Section II below. The main contributions of our paper include the following.

1) We perform analysis and empirical studies of relevant characteristics of the objective functions described in [44].
2) We extend the objective functions with congestion information to improve the routability of results.
3) We implement a top-down hierarchical (multilevel) placer (APlace) based on the objective functions. For IBM-ISPD04 circuits, the half-perimeter wirelength (HPWL) of APlace outperforms that of FastPlace, UCLA Dragon (v2.2.3), and Capo (v8.8), respectively, by 7.8%, 6.5%, and 7.0% on average. For eight IBM-PLACE v2 circuits, after APlace's results are detail-routed using Cadence WRoute (SE5.4), the average improvement in final wirelength is 12.0% over Cadence QPlace (SE5.4), and 8.1% over UCLA Dragon (v3.01), and 14.1% over Capo (v8.7).
4) We extend the placer to handle mixed-size placement. Our extension is compared to recent academic tools: UCLA mPG-MS [11], Feng Shui (v2.4) [35], and a three-stage placement-floorplanning-placement flow that uses Capo [1], [2]. For ten IBM-ISPD02 mixed-size circuits, the HPWL of our placer outperforms that of mPG-MS, Feng Shui and the Capo flow respectively by 24.7%, 4.0%, and 26.0% on average.
5) We extend the placer to address timing-driven placement. Our extension is compared to two industry placers: QPlace (SE v5.4) and amoebaPlace (SoC Encounter v3.2). When timing-driven placement is performed for six industry circuits and placements are detail-routed using Cadence WarpRoute (SoC Encounter v3.2), our placer has a minimum cycle time that outperforms that of QPlace and amoebaPlace respectively by 9.6% and 8.5%, as well as average improvements of 7.2% and 6.5% in routed wirelength, respectively.
6) We extend the placer to perform input/output (I/O)-core coplacement for area-array I/O designs. I/Os can be evenly distributed without damaging the wirelength figure of merit.
7) We also extend the placer with constraint handling for mixed-signal designs. Basic geometric constraints including alignment, spacing and symmetry constraints can be enforced during placement.

The remainder of this paper is organized as follows. Section II discusses the objective functions of the placer. Section III describes our implementation, and Section IV summarizes the placement results. In Sections V and VI, we extend the placer to handle mixed-size placement and timing-driven placement. In Sections VII and VIII, the placer is extended with I/O-core coplacement and constraint handling. The paper concludes in Section IX.

II. Problem Formulation

A basic goal of placement is to minimize wirelength subject to the constraint that cells do not overlap. Therefore, the objective function for analytic placement historically includes two terms: a *density objective* to spread cells, and a *wirelength objective* to minimize wirelength. In this section, we discuss the basic objective functions that our placer uses and experiments are performed to study characteristics of them.

A. Cell Spreading

One important objective of a placer is to distribute cells evenly over the placement area. Constructive placement methods can easily achieve this goal. Force-directed placement methods apply forces based on the current area distribution to move cells away from high-density regions and toward low-density regions. However, it is difficult to choose appropriate forces; wirelength is often seriously damaged when the cells are spread out.

To distribute cells evenly over the placement area, a generic strategy is to divide the placement region into grids and then attempt to equalize the total cell area in every grid. The straightforward "squared deviation" penalty for uneven cell distribution is

$$\text{Penalty} = \sum_{\text{Grid } g} (\text{TotalCellArea}(g) - \text{AverageCellArea})^2. \quad (1)$$

However, this penalty function is not smooth or differentiable, and is hence difficult to optimize. Naylor *et al.* [44] tried to smooth the above penalty function, proposing a "bell-shaped" cell potential function instead of a solid cell area function. For a cell c with center at $(\text{Cell } X, \text{Cell } Y)$ with area A_c, the potential at grid point $g = (\text{Grid } X, \text{Grid } Y)$ is given by

$$\text{Potential}\,(c, g) = K_c \cdot p(|\text{Cell } X - \text{Grid } X|) \cdot p(|\text{Cell } Y - \text{Grid } Y|) \quad (2)$$

where

$$p(d) = \begin{cases} 1 - 2d^2/r^2 & (0 \le d \le r/2) \\ 2(d-r)^2/r^2 & (r/2 \le d \le r) \\ 0 & (d > r) \end{cases}. \quad (3)$$

Here, $p(d)$ defines the bell-shaped function, which is illustrated in Fig. 1; r controls the *radius* of any given cell's potential (range of interaction); and K_c is a normalization factor so that $\sum_g \text{Potential}\,(c, g) = A_c$, i.e., each cell has a total potential equal to its area. Then, the penalty function in (1) is transformed to

$$\text{Penalty} = \sum_{\text{Grid } g} \left(\sum_{\text{Cell } c} \text{Potential}\,(c, g) - \text{ExpPotential}(g) \right)^2 \quad (4)$$

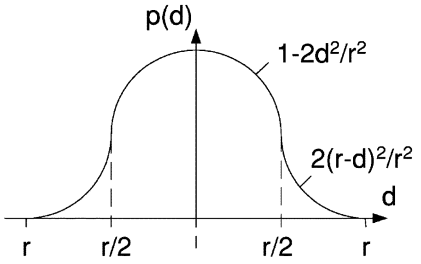

Fig. 1. Bell-shaped function.

TABLE I
CELL DISTRIBUTION RESULTS WITH DIFFERENT NUMBER OF GRID NODES AND CELL POTENTIAL RADII (r's) FOR THE ibm01-EASY CIRCUIT

	$r=2$			$r=4$		
grids	disc	iters	CPU(s)	disc	iters	CPU(s)
10	1.656	50	56	1.890	183	399
20	1.159	29	30	2.799	113	237
30	1.121	46	47	1.162	46	108
40	1.125	83	81	1.089	42	92
50	1.079	235	206	1.128	54	116
60	1.063	758	695	1.111	56	123
70	-	-	-	1.077	83	190
80	-	-	-	1.083	111	235
90	-	-	-	1.083	167	343
100	-	-	-	1.081	273	577

where $\mathrm{ExpPotential}(g) = \mathrm{TotalCellArea}/\mathrm{NumGrids}$ is the expected total potential at the grid point g.

We can use the conjugate gradient method to minimize the penalty function in (4). Our implementation will be described in detail in Section III. We calibrate our optimizer using the ibm01-easy testcase from the IBM-PLACE 2.0 benchmark suite [25]. Our stopping criterion is when the maximum (Manhattan) movement of any cell between consecutive iterations is less than 30 units. Results with different numbers of grids and cell potential radii (r's) are summarized in Table I.

Definition 1: Discrepancy within area A_w is defined as the maximum ratio of actual total cell area to expected cell area over all windows with area A_w.

We use *discrepancy* to measure evenness of cell distribution. In Table I, the second and fifth columns show the discrepancy within 1% of the total placement area, for each of the cell distributions that the placer obtains. The number of iterations, and total running times needed to meet the above convergence criterion, are also shown in the table.

From these tests, we make the following conclusions. 1) The finer the grid, the more iterations needed for the optimizer to converge; however, a more even cell distribution is obtained. 2) For finer grids, larger values of r help to reduce the number of iterations, although the running time per iteration is increased. 3) We observe serious oscillations in discrepancy when $r = 2$ and the number of grid nodes is larger than 60.

B. Wirelength Formulation

Minimization of wirelength is a common objective for circuit placement. Linear and quadratic wirelength objectives are typically used; see, e.g., [39] and [51] for comparisons. The quadratic objective function is used in many analytical placement methods because it is continuously differentiable and can be minimized efficiently by solving a system of linear equations. Unfortunately, this is not true for linear objective functions, and linear programming suffers from excessive computation times. The Gordian-L objective [51] minimizes a linear wirelength function using quadratic programming methods. Also, [38] proposes an α-order objective function to capture the strengths of both methods.

While wirelength and overall placement quality is typically evaluated according to the HPWL, this "linear wirelength" function cannot be efficiently minimized. Convex nonlinear approximations of HPWL, which do not require net models and which permit direct inclusion of nonlinear delay terms, are proposed and well-studied in such papers as [3], [6], [33], and [34]. The approach of Naylor *et al.* [44] follows along similar lines, and uses a log-sum-exp method to capture the linear HPWL while simultaneously obtaining the desirable characteristic of continuous differentiability. The log-sum-exp formula picks the most dominant terms among pin coordinates. For a net t with pin coordinates $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the wirelength objective is

$$\mathrm{WL}(t) = \alpha \cdot \left(\ln\left(\sum e^{x_i/\alpha}\right) + \ln\left(\sum e^{-x_i/\alpha}\right) \right) + \alpha \cdot \left(\ln\left(\sum e^{y_i/\alpha}\right) + \ln\left(\sum e^{-y_i/\alpha}\right) \right) \quad (5)$$

where α is a smoothing parameter. $\mathrm{WL}(t)$ is strictly convex, continuously differentiable and converges to $\mathrm{HPWL}(t)$ as α converges to 0. The log-sum-exp formula picks the most dominant terms; it has been previously used in physical design applications such as transistor sizing [50].

We minimize the wirelength objective function using the conjugate gradient optimizer. Initially, cells are randomly distributed over the placement area, and then the wirelength objective function in (5) is minimized. The program is stopped after 300 iterations. Results with different smoothing parameter values (α's) are shown in Table II. The wirelength calculated according to (5) for the initial random placement is displayed in the third column. Compared to the actual initial HPWL of this random placement (i.e., Initial $\mathrm{HPWL} = 7.311$), we see that the wirelength formulation becomes more accurate (closer to HPWL) when smaller values of α are used. However, with larger α values, the wirelength objective function is more smooth and can be minimized more quickly, and a smaller final HPWL is obtained (e.g., final $\mathrm{HPWL} = 0.803$ for $\alpha = 3336$).

Combining the above two objectives, the analytic placer optimizes the following function:

$$\mathrm{WLWeight} * \mathrm{TotalWL} + \mathrm{DensityWeight} * \mathrm{DensityPenalty}. \quad (6)$$

The density term drives the spreading of cells and is always changing with the current cell distribution. The wirelength term draws connected components back toward each other.

III. DETAILED IMPLEMENTATION

We now describe implementation details of APlace.

TABLE II
WIRELENGTH MINIMIZATION RESULTS WITH DIFFERENT SMOOTHING PARAMETERS (α's) FOR THE ibm01-EASY CIRCUIT

grids	α	init WL	final HPWL	CPU (s)
10	3336	7.533	0.803	194
20	1668	7.369	0.836	262
30	1112	7.337	0.913	99
40	834	7.326	1.274	238
50	667	7.321	1.400	245
60	556	7.318	1.499	245
70	476	7.316	1.583	252
80	417	7.315	1.653	266
90	370	7.314	1.712	240
100	334	7.314	1.764	248

Initial HPWL = 7.311

A. Conjugate Gradient Optimizer

We use the conjugate gradient method to optimize the objective function as described in (6). The conjugate gradient method is quite useful in finding an unconstrained minimum of a high-dimensional function $f(x)$. A detailed treatment, along with a survey of descent-based methods for nonlinear programming, can be found in [42].

In general, the conjugate gradient method finds the minimum by executing a series of *line minimizations* (i.e., line searches). A line minimization corresponds to one-dimensional function minimization along some search direction. The result of one line minimization is used as the start point for the next line minimization. The method has the following form:

$$d_k = \begin{cases} -g_k, & \text{for } k = 1 \\ -g_k + \beta_k d_{k-1}, & \text{for } k > 1 \end{cases} \tag{7}$$

$$x_{k+1} = x_k + \alpha_k d_k \tag{8}$$

where g_k denotes the gradient $\nabla f(x_k)$, α_k is a step length obtained by a line search algorithm, d_k is the search direction, and β_k is chosen so that d_k becomes the kth conjugate direction when the function is quadratic and the line search finds the minimum along the direction exactly. Varieties of the conjugate gradient methods differ in how they select β_k; the best-known formulas for β_k are due to Fletcher–Reeves, Polak–Ribiere, and Hestenes–Stiefel. Our implementation uses the Polak–Ribiere formula:

$$\beta_k^{\mathrm{PR}} = \frac{g_k^T (g_k - g_{k-1})}{\|g_{k-1}\|^2}. \tag{9}$$

A Golden Section search method is used to find the step length for each iteration. The length of the search interval reduces by a factor of 0.618 (the golden ratio) or more in each step, and converges linearly to zero.

The conjugate gradient iteration in (8) repeats until the following stopping criterion is reached: 1) a predetermined number of iterations has passed; 2) the step length returned by the line search function is small enough; or 3) the function value is not changing significantly with additional iterations.

B. Control Factors

The weights of the wirelength and density objective functions provide important control parameters to the placer. Intuitively, a larger wirelength weight will draw cells together and prevent them from spreading out, while a larger density penalty weight will spread the cells out (without attention to wirelength). These controls are managed by keeping the density weight fixed at some constant, and setting the wirelength weight to be large at the outset, but then decreasing this weight (by a factor of two, or a smaller factor near the final placement) whenever the conjugate gradient optimizer slows down and a stable solution emerges. After every weight change, the conjugate gradient optimizer is used to compute a new stable state wherein cells are distributed more evenly but wirelength is larger. The process repeats until the cells are spread evenly over the placement area.

The number of grid nodes, cell potential parameter r and wirelength smoothing parameter α are also important control knobs for APlace. According to the empirical studies discussed in Section II above, the number of grid nodes should increase during the whole placement process. Coarser grids at the beginning spread out the cells faster, while finer grids at the final stages help to reach a more even distribution. In our implementation, the cell potential radius r is set to 2 for coarser grids in order to reduce running times, and to 4 for finer grids. The wirelength smoothing parameter α is set to be half of the grid length.

C. Top-Down Hierarchical Algorithm

We use a top-down hierarchical approach to accelerate APlace. During initialization, a hierarchy of clusters is constructed using MLPart 4.21, a leading-edge, open-source min-cut hypergraph/circuit partitioner [7]. The top-down hierarchical algorithm is described in Fig. 2. Notations used are summarized as follows.

c	Cell.
n	Number of cells.
MaxLevel	Maximum number of cluster levels.
N_l	Number of clusters at level l.
$C_l(i)$	Cluster of cells at level l.
Area(C)	Total cell area of cluster C.
r	Radius of cell's potential.
GridLen	Spacing of grids.
α	Wirelength smoothing parameter.
f	Objective function of the placer.
$\{\mathrm{Gradient}_l(i)\}$	Gradient vector
$\{\mathrm{ClusterPosition}_l(i)\}$	Vector of cluster positions.
$\{\mathrm{CellPosition}(c)\}$	Vector of cell positions.

Subscript ranges, where not explicit, are $c = 1, \ldots, n; l = 0, \ldots, \mathrm{maxLevel}$ and $i = 1, \ldots, N_l$.

For each level in the cell/cluster hierarchy, a coarse grid is determined by the average cluster size. We compute the density penalty by regarding cells in a cluster as a macro cell with area equal to the total cell area of the cluster. For wirelength calculation, cells are assumed to be located at the center of the cluster.

Figs. 3 and 4 show how discrepancy and HPWL change with successive iterations for the ibm01-easy circuit. The clustering hierarchy for ibm01-easy has three levels. During the first approximately 900 iterations, the placer is working at cluster levels. Clustering helps to spread cells more quickly, but wirelength is impaired during cell expansion. It is clearly seen from the figures that when wirelength weight is decreased and the conjugate gradient optimizer restarts, discrepancy

Top-Down Hierarchical APlace Algorithm

Input:
User-defined objective density discrepancy $DestDisc$
User-defined minimum step length ϵ
User-defined total number of iterations $TotalIters$

Output:
Cell placement $\{CellPosition(c)\}$

Algorithm:
01. Construct a hierarchy of clusters $\{C_l(i)\}$
02. **For** $(l = MaxLevel;\ l >= 0;\ l = l - 1)$
03. Set initial placement $\{ClusterPosition_l(i)\}$
04. $GridLen \propto \sum_i sqrt(Area(C_l(i)))/N_l$)
05. $\alpha \propto GridLen$
06. $DensityWeight = 1$
07. $WLWeight \propto \frac{DensityPenalty(GridLen,\ r)}{Wirelength(\alpha)}$
08. **While** ($\#Iter < TotalIters$)
09. $f = DensityPenalty(gridLen,\ r) + WLWeight * Wirelength(\alpha)$
10. $Gradient_l = Gradient(f)$
11. $StepLength = LineSearch(f, Gradient_l)$
12. $ClusterPosition_l = CellPosition_l + StepLength * Gradient_l$
13. **If** $(StepLength < \epsilon)$
14. $WLWeight = WLWeight/2$
15. **If** $(Discrepancy(gridLen) > DestDisc)$
16. $Break$
17. $\{CellPosition(c)\} = \{ClusterPosition_0(i)\}$

Fig. 2. Top-down hierarchical APlace.

drops sharply and wirelength is often increased at first and then refined during the optimization. When both discrepancy and wirelength change slowly, we have a near stable suboptimal solution for the current objective function; additional iterations will not further reduce discrepancy and wirelength very much and the wirelength weight should be reduced. Guided by these figures, we set the iteration limit at 100 in our experiments below.

D. Detailed Placement

The placement results of APlace have cell overlaps and need to be legalized. A simplified Tetris [22] legalization algorithm is implemented in APlace; this algorithm also bears strong resemblance to the method proposed in a technical report of Li and Koh [40]. The Tetris legalization is applied after global placement: Cells are sorted according to their vertical coordinates, and then for each cell from left to right the current nearest available position is found. This greedy algorithm is very fast, with negligible running time compared to that of global placement, and increases the wirelength by about 4% for IBM-PLACE 2.0 circuits [25].

After legalization, the modules of orientation optimization and "row ironing" from UCLApack [10] are applied to the results. Row ironing helps to improve wirelength by applying a branch-and-bound placer in sliding windows. These algorithms are fast (running times are ignorable compared to that of global placement) and decrease the wirelength by about 2% for IBM-PLACE 2.0 circuits [25].

E. Congestion-Directed Placement

To improve routability of placement results, we have integrated congestion information into the objective functions to direct cell distribution. We use Kahng and Xu's accurate bend-based congestion estimation method [31] in our placer. If a particular grid is determined to be congested (respectively, uncongested), the expected total cell potential of the grid in (4) is reduced (respectively, increased) accordingly. The sum of expected area potential over all grids is kept constant, and equal

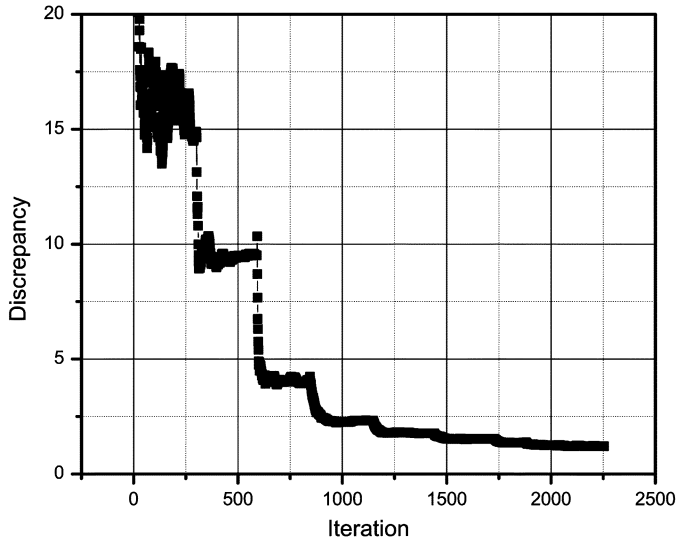

Fig. 3. Discrepancy as a function of iterations for the ibm01-easy circuit.

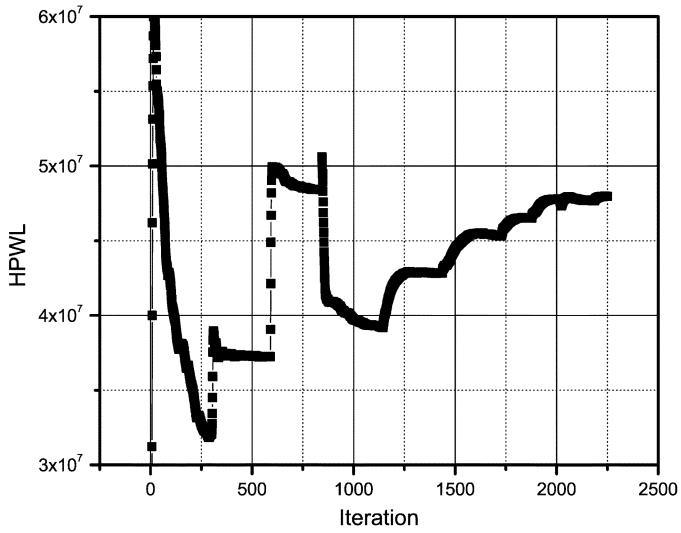

Fig. 4. HPWL as a function of iterations for the ibm01-easy circuit.

TABLE III
PLACEMENT AND GLOBAL ROUTING RESULTS OF APlace WITH VARYING CONGESTION ADJUSTMENT FACTORS (γ's) FOR THE ibm01-HARD CIRCUIT

γ	Placement				Global Routing	
	WL	WL_l	disc	iters	WL	over-cap
0.00	0.459	0.502	1.18	1006	0.119	4035
0.02	0.462	0.505	1.18	997	0.118	3488
0.04	0.464	0.509	1.23	1006	0.119	3249
0.05	0.474	0.523	1.26	1086	0.120	2486
0.06	0.477	0.529	1.28	1086	0.121	2576
0.08	0.486	0.541	1.36	1006	0.123	2806
0.10	0.507	0.562	1.56	804	0.129	3350

to the total cell area. Specifically, expected cell potential is adjusted as follows:

$$\text{ExpPotential}(g) \propto 1 + \gamma\left(1 - 2\frac{\text{Congestion }(g)}{\max_g\{\text{Congestion }(g)\}}\right) \quad (10)$$

where γ is the congestion adjustment factor and decides the extent of congestion-directed placement.

Table III summarizes placement runs for the ibm01-easy circuit with different congestion factors (γ's). Specific parameters used for these runs are $\epsilon = 30$ and $\text{destDisc} = 1.1$. Wirelengths before and after legalization are shown in the second and third columns. Discrepancy within 1% placement area is shown in the fourth column, and serves as a measure of evenness of cell distribution. The fifth column shows the total number of iterations. Global routing is performed using Cadence WRoute (SE5.4). The last two columns show the total wire length in the gcell grid, and total number of overcapacity gcells; these values provide figures of merit for routability of the placement results.

According to these results, routability is approximately 38% better with a congestion factor of 0.05, but deteriorates when the congestion factor is larger. Wirelength of congestion-aware placements is not seriously impaired.

IV. Experimental Results

APlace is implemented in C++. We use the IBM-PLACE v2.0 benchmarks [25] as our test cases. These circuits include easy and hard cases of eight designs. IBM-PLACE circuits are widely used in the literature, and are becoming common benchmarks for standard-cell placement.

Our results are compared with a leading industry placer, Cadence QPlace (SE5.4) and the recent academic tools UCLA Dragon (v3.01) [14] and Capo (v8.7) [10]. All placers read the same LEF/DEF files and write DEF files as the placement output. Results of Capo are legalized using Cadence QPlace (SE5.4, ECO mode). The placements are then sent to Cadence WRoute (SE5.4, global and final routing enabled and auto-stop disabled) to be routed.

All the experiments of APlace, Dragon, and Capo are performed on a Xeon server with 2.4-GHz CPU (double threads on each CPU) and 4 GB of memory; QPlace and WRoute are run on a Sun Ultra10 workstation with a 400-MHz CPU. We run QPlace with full placement mode (-full), and Dragon with fixed die mode (-fd). We run MetaPlacer for Linux, which incorporates Capo and orientation optimizer. Row ironing is disabled (-noRowIroning) for MetaPlacer for better routability of the placement results. One start of Dragon and of Capo is done for each case in the experiments, since the whole flow of placement and routing is very time consuming.

The APlace results with no congestion awareness are summarized in Table IV. Specific parameters used for these runs are: $\epsilon = 30$ and $\text{destDisc} = 1.1$. Wirelengths (meters) before and after legalization are shown in the third and fourth columns. Average wirelength increase after legalization is 3.7% (range: 1.3%–7.6%). Discrepancy within 1% placement area is shown in the fifth column and used to evaluate the evenness of cell distribution in APlace results. The last two columns show the total number of iterations and running times in minutes.

TABLE IV
Placement Results of APlace (no Congestion Awareness) for IBM-PLACE 2.0 Circuits

IBM-PLACE 2.0		APlace				
ckts	cells	WL	WL_l	disc	iters	CPU
ibm01_easy	12k	0.46	0.49	1.17	885	8
ibm01_hard		0.46	0.49	1.16	983	10
ibm02_easy	19k	1.44	1.47	1.14	1068	15
ibm02_hard		1.37	1.41	1.15	1145	19
ibm07_easy	45k	3.13	3.21	1.13	1100	43
ibm07_hard		3.04	3.16	1.14	1094	43
ibm08_easy	51k	3.41	3.48	1.15	923	43
ibm08_hard		3.32	3.44	1.13	995	45
ibm09_easy	51k	2.81	2.91	1.12	982	48
ibm09_hard		2.71	2.86	1.12	985	45
ibm10_easy	67k	5.34	5.46	1.15	1078	67
ibm10_hard		5.23	5.63	1.16	1060	61
ibm11_easy	68k	4.26	4.37	1.15	1090	66
ibm11_hard		4.15	4.32	1.13	1085	69
ibm12_easy	69k	7.86	7.96	1.17	1059	65
ibm12_hard		7.59	7.72	1.15	967	56

The placement results with congestion awareness are sent to Cadence WRoute (SE5.4) to be routed. The results are compared to QPlace, Dragon and Capo in Table V. Routing results include *success* (finished routing with no violations), *finished* (with violations) and *failure* (because of time limit). For all cases, the number of violations, final wirelength, the number of vias and running times of WRoute are shown in the last four columns.

According to the results, average *placed* wirelength improvement over QPlace is 8.5% (range: 2.0%–11.9%); average improvement over Dragon is 4.4% (range: −1.8%–9.7%); average improvement over Capo is 8.3% (range: 4.2%–10.9%). APlace is faster (0.8X) than Dragon, but much (13X) slower than Capo.

We observe that almost all of the circuits are successfully routable with good wirelength; finished routings with a small number of violations can be manually fixed. Average improvement of the *routed* wirelength over QPlace is 12.0% (range: 4.4%–18.6%); average improvement over Dragon is 8.1% (range: −0.1%–12.3%); average improvement over Capo is 14.1% (range: 5.6%–20.3%). The routability of APlace's results is better than that of Capo's; routed wirelength of APlace outperforms that of QPlace and Dragon with an average of less than 2% increase in the number of vias and less than 80% increase in running time of WRoute.

We also tested APlace on IBM-ISPD04 benchmarks [55]. These circuits are derived from IBM benchmarks, but have fixed pads on the boundary. The APlace results with no congestion awareness are summarized in Table VI. The results are compared with those of FastPlace, Dragon (v2.2.3) and Capo (v.8.8) reported in [55]. According to the results, average *placed* wirelength improvement over FastPlace is 7.8% (range: 0.8%–11.4%); average improvement over Dragon is 6.5% (range: −2.8%–13.1%); average improvement over Capo is 7.0% (range: 3.1%–11.1%). Running times (in seconds) cannot be directly compared: Capo is run on a Sun Sparc-2

TABLE V
PLACEMENT AND ROUTING RESULTS OF APlace (WITH CONGESTION AWARENESS) FOR EIGHT IBM-PLACE 2.0 CIRCUITS WITH COMPARISON TO QPlace, DRAGON AND CAPO

Ckts	Placer	Place		Route			
		WL	CPU	viol	WL	vias	CPU
ibm01e	QPlace	0.59	3	0	0.84	138563	58
	Dragon	0.57	27	0	0.86	141304	60
	Capo	0.57	1	587	0.85	146706	1446*
	APlace	**0.53**	**23**	**0**	**0.75**	**139134**	**100**
ibm01h	QPlace	0.56	3	0	0.80	138593	82
	Dragon	0.55	26	0	0.80	139993	90
	Capo	0.56	1	1029	0.84	173715	1446*
	APlace	**0.51**	**22**	**0**	**0.71**	**138745**	**82**
ibm02e	QPlace	1.56	7	0	2.10	291154	58
	Dragon	1.60	38	0	2.20	310679	243
	Capo	1.60	2	35	2.33	317659	636
	APlace	**1.47**	**37**	**0**	**1.98**	**286813**	**47**
ibm02h	QPlace	1.52	7	0	2.18	305420	104
	Dragon	1.47	40	0	2.16	304376	105
	Capo	1.56	2	147	2.21	319702	929
	APlace	**1.49**	**38**	**0**	**2.09**	**302186**	**70**
ibm07e	QPlace	3.72	12	0	4.61	572512	98
	Dragon	3.66	65	0	4.58	569087	103
	Capo	3.71	7	42	4.93	599806	996
	APlace	**3.30**	**68**	**0**	**3.99**	**532963**	**74**
ibm07h	QPlace	3.70	12	0	5.04	617942	184
	Dragon	3.44	66	15	4.63	606561	135
	Capo	3.56	8	1799	5.14	631456	1483*
	APlace	**3.26**	**55**	**0**	**4.10**	**547398**	**96**
ibm08e	QPlace	3.95	16	0	5.30	703891	133
	Dragon	3.61	143	0	4.58	659802	57
	Capo	3.93	7	0	5.07	682815	137
	APlace	**3.60**	**80**	**0**	**4.43**	**645776**	**94**
ibm08h	QPlace	3.85	17	0	5.26	726576	194
	Dragon	3.45	143	0	4.47	679761	100
	Capo	3.90	8	1484	5.15	728195	1458*
	APlace	**3.52**	**49**	**22**	**4.48**	**676674**	**278**

* WRoute reaches time limit (24h).

TABLE VI
PLACEMENT RESULTS OF APlace (NO CONGESTION AWARENESS) FOR IBM-ISPD04 CIRCUITS

IBM-ISPD04		FastPlace	Dragon	Capo		APlace	
ckts	cells	WL	WL	WL	CPU	WL	CPU
ibm01	12k	1.91	1.84	1.86	239	1.74	608
ibm02	19k	4.02	3.98	4.06	435	3.68	715
ibm03	22k	5.45	5.31	5.11	503	4.83	839
ibm04	27k	6.63	6.22	6.39	646	5.89	1426
ibm05	29k	10.96	10.35	10.56	644	10.23	1203
ibm06	32k	5.55	5.45	5.50	728	5.12	2039
ibm07	45k	9.56	9.26	9.63	1112	8.61	1524
ibm08	51k	10.01	9.66	10.26	1197	9.93	3769
ibm09	52k	11.26	11.03	10.56	1370	9.99	2859
ibm10	68k	19.31	19.60	19.70	1744	18.24	4127
ibm11	69k	16.03	15.36	15.73	1871	14.77	3816
ibm12	70k	25.04	24.74	25.83	1841	23.57	4674
ibm13	82k	19.46	19.32	18.73	2367	17.59	3616
ibm14	146k	36.09	35.77	36.69	4320	32.83	8694
ibm15	158k	45.21	43.39	43.85	5400	41.70	13043
ibm16	182k	48.43	49.54	49.63	5460	45.65	10869
ibm17	183k	68.09	73.45	69.07	6180	65.83	14059
ibm18	210k	46.89	48.59	47.46	6240	42.21	14373

750-MHz machine and APlace is run on a PIII 1.4 GHz machine. On average, APlace is less than 4.3X slower than Capo on IBM-ISPD04 benchmarks, and much slower (about 50X) than FastPlace.

V. MIXED-SIZE PLACEMENT

As VLSI designs scale to billion-transistor complexities, design productivity increasingly requires the reuse of predesigned or generated macro blocks (processing and interface cores, embedded memories, etc.). This presents a "boulders and dust" challenge to placers [5], where the sizes of placeable objects can vary by factors of 10 000 or more. In this section, we extend the APlace approach to address the mixed-size placement problem. Our focus is on two issues: 1) the cell-spreading potential function and 2) legalization.

A. Previous Work

Modern application specific integrated circuit (ASIC) designs are typically laid out in the fixed-die context, where the outline of the core area, as well as routing tracks and power/ground distribution, are fixed before placement [8]. Mixed-size placement becomes particularly complex in the fixed-die context because of its discreteness [2]. Traditional standard-cell frameworks often cannot address this challenge smoothly, and must resort to devices such as manual preplacement of blocks, with an attendant loss of overall solution quality.

Recently, a three-stage placement-floorplanning-placement flow [1], [2] has been proposed which places macro blocks and standard cells without overlap. In the first step, all macros are shredded into small pieces connected by fake wires. A standard-cell placer, Capo, is used to obtain an initial placement, and the initial locations of macros are produced by averaging the locations of faked cells created during the shredding process. In the second step, the standard cells are merged into soft blocks, and a fixed-outline floorplanner, Parquet, generates valid locations of macros and soft blocks of movable cells. Finally, with macros considered fixed, Capo is used again to replace small cells. This approach scales reasonably well, but wirelength results are often quite suboptimal.

A different approach is pursued by [11], wherein a simulated annealing-based multilevel placer, mPG-MS, recursively clusters both macro blocks and standard cells to build a hierarchy. The top-level netlist is placed, and then the placement is gradually refined by unclustering the netlist and improving the placement of smaller clusters by simulated annealing. Large objects are gradually fixed without overlap during coarse placement, and the locations of smaller objects are determined during further refinement. Significant effort is needed for legalization and overlap removal during this placement process.

Another recursive bisection-based placement tool, Feng Shui, has been presented more recently in [35]. Rather than addressing standard cells and macro blocks separately, the placer considers them simultaneously via a *fractional cut* technique which allows horizontal cut lines that are not aligned with row boundaries. When compared to previous academic tools, the Feng Shui placer achieves surprising solution quality (as evaluated by placed HPWL) and good scalability.

B. Potential Function for Mixed-Size Placement

As described in Section II-A, the density objective drives cell spreading. The placement area is divided into grids, each cell has a potential or influence with respect to nearby grids, and the

placer seeks to equalize the total cell potential at each grid. For standard-cell placement, the grid usually has a length greater than the average cell width, and the radius of cell's potential, r, is set to be a constant during optimization. However, for mixed-size placement, the size range between large and small objects can be as large as a factor of 10 000 [11], and the radius of influence of a cell's potential will need to change according to the cell's dimension. In particular, a larger block will have potential with respect to more grids.

After investigation of several possibilities, we have chosen to address the potential function for large macros in the following simple way. Suppose a macro block b has width w. The radius or scope of this block's influence is $w/2 + r$, i.e., every grid within the distance of $w/2 + r$ from the block's center has a nonzero potential from this block. Moreover, the total potential of the block over all grids is equal to the block's area. Therefore, the function $p(d)$ in (2) becomes

$$p(d) = \begin{cases} 1 - a * d^2, & (0 \le d \le w/2 + r/2) \\ b * (d - r)^2, & (w/2 + r/2 \le d \le w/2 + r) \\ 0, & (d > w/2 + r) \end{cases} \quad (11)$$

where

$$a = 4(w + r^2)/((w + 2r^2)(w + r)^2)$$
$$b = 4/(w + 2r^2) \quad (12)$$

so that the function $p(d)$ is continuous when $d = w/2 + r/2$.

C. Legalization

A second key issue is that analytic placement results have cell overlaps that must be legalized. After investigation of a variety of approaches, we perform legalization in mixed-size placement as follows. Cells are sorted based on a combination of vertical coordinate and width, so that larger blocks may be fixed at a position ahead of nearby small cells. We also scale the cell positions to the left side by a fixed factor (set to 0.90 in all of the discussion and results below) so that 1) cells will not be pushed outside the placement region and 2) horizontal overlaps among macros can be properly resolved by the legalization.

When an initial global placement has many overlaps among macros, legalization of mixed-size circuits can be extremely challenging. Indeed, a greedy algorithm such as "Tetris" may fail to find a valid position for one or more blocks, or wirelength may be seriously damaged by movement of blocks. Fortunately, the potential function described above allows our mixed-size placer to distribute cells quite evenly in the global placement, with little overlap among larger blocks. Hence, the greedy legalization approach is still an acceptable adjunct even for mixed-size placement: Wirelength increase after the legalization step for our testcases is 6.5% on average (cf. an increase of approximately 5% for pure standard-cell designs).

D. Experimental Results

We use ten circuits from the IBM-ISPD02 Mixed-Size Placement Benchmarks [26] as our testcases. These circuits are publicly available at the GSRC Bookshelf [19] and are widely used in the literature.

TABLE VII
RESULTS OF OUR PLACER FOR TEN MIXED-SIZE CIRCUITS

circuit	APlace-MS				detailed placement		
	WL	WL_l	inc.	CPU	WL_dp	impr.	CPU
ibm01	0.20	0.24	18.5	15	0.23	5.7	1
ibm02	0.51	0.52	0.7	45	0.50	2.5	3
ibm03	0.70	0.74	6.2	56	0.72	3.5	3
ibm04	0.81	0.85	4.8	48	0.83	2.8	4
ibm05	1.01	1.00	-0.5	15	0.98	2.0	5
ibm06	0.65	0.71	9.6	76	0.68	4.4	5
ibm07	1.03	1.09	5.8	98	1.05	3.7	8
ibm08	1.49	1.50	0.6	128	1.46	2.7	8
ibm09	1.25	1.45	15.7	113	1.38	5.2	9
ibm10	2.97	3.07	3.3	206	3.00	2.2	11

TABLE VIII
COMPARISON OF OUR RESULTS WITH THE CAPO FLOW mPG-MS AND FENG SHUI

circuit	Capo		mPG-MS		Feng Shui		APlace-MS	
	WL	CPU	WL	CPU	WL	CPU	WL	CPU
ibm01	0.31	20	0.30	18	0.24	3	0.23	16
ibm02	0.68	11	0.74	32	0.53	5	0.50	48
ibm03	1.04	59	1.20	32	0.75	6	0.72	59
ibm04	1.01	15	1.05	42	0.80	7	0.83	52
ibm05	1.11	5	1.09	36	1.01	8	0.98	20
ibm06	0.99	18	0.92	45	0.68	10	0.68	81
ibm07	1.53	25	1.37	68	1.17	13	1.05	106
ibm08	1.79	29	1.64	82	1.36	16	1.46	136
ibm09	1.99	29	1.86	84	1.38	15	1.38	122
ibm10	4.55	116	4.36	172	3.75	22	3.00	217

Table VII summarizes the results for the ten mixed-size circuits. The second and third columns show HPWLs (meters) before and after legalization. The fourth column shows the wirelength increase due to legalization, expressed as a percentage. Average wirelength increase after the legalization step for our testcases is 6.5%. As our placer does not perform detailed placement, we use Feng Shui (v2.4) [35] as the detailed placer. Wirelength (meters) after detailed placement and percentage improvement are shown in the sixth and seventh columns. We see that the average wirelength improvement after the detailed placement step is 3.5%. All of our experiments are performed on an Intel Xeon server with dual 2.4-GHz CPUs (double threads on each CPU) and 4 GB of memory. Running times of global and detailed placement (minutes) are shown in the fifth and eighth columns, respectively.

Table VIII compares our results with those of recently published papers that experiment with the same benchmarks. The results of the three-stage placement-floorplanning-placement flow, which uses the Capo standard-cell placer, were first presented in [1] and further improved in [2], results of the mPG-MS placer were presented in [11], and results of Feng Shui (v2.4) are from [35]. Final HPWL values and running times (minutes) for each placer are also shown in Table VIII.

According to the results, average wirelength improvement of our placer over the Capo flow is 26.0% (range: 11.5%–34.0%); average improvement over mPG-MS is 24.7% (range: 9.9%–40.1%); and average improvement over Feng Shui is 4.0% (range: −7.3%–20.0%). Running times of the placers cannot be directly compared, since the Capo flow used 2-GHz Linux/Pentium IV workstations, mPG used 750-MHz Sun Blade 1000 workstations, and Feng Shui used 2.5-GHz

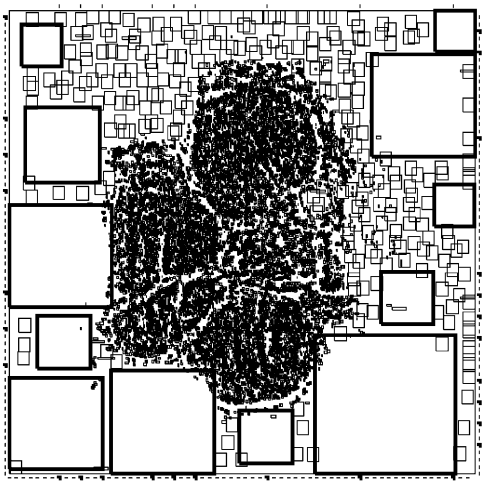

Fig. 5. Placement before legalization of our placer for the ibm02 circuit.

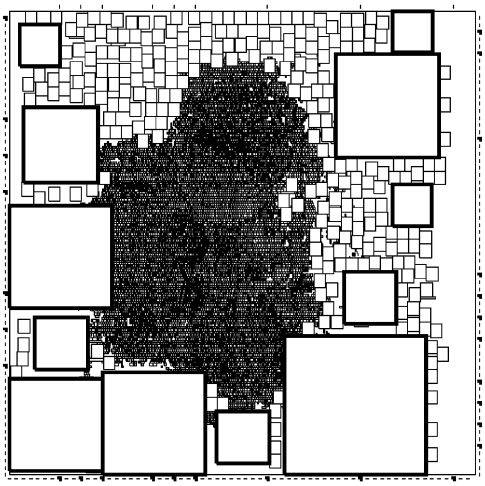

Fig. 6. Placement after legalization of our placer for the ibm02 circuit.

Linux/Pentium IV workstations. However, APlace is much slower than Feng Shui. Finally, Figs. 5 and 6 show placements for the ibm02 benchmark before and after legalization.

VI. Timing-Driven Placement

Design value depends on performance; with device and interconnect scaling, this presents greater challenges to timing-driven placement. In this section, we extend the placer to address timing-driven placement.

A. Previous Work

Timing-driven placement has been studied extensively. Existing approaches can be broadly divided into two classes: *path-based* and *net-based*.

A typical path-based approach [21], [28], [47], [48] usually considers all or a subset of paths directly within the problem formulation. The majority of this class of approaches are based on mathematical programming techniques. This class of algorithms usually maintains an accurate timing view during optimization, but its drawback is relatively high complexity due to the exponential number of paths that need to be simultaneously minimized.

For this reason, much of the recent timing-driven work [20], [45], [46] has been net-based. Unlike path-based approaches that handle paths directly, net-based approaches [15], [49] usually transform timing constraints or requirements into either net weight or net length (or delay) constraints, and employ a weighted wirelength minimization engine.

The process of generating net-length constraints or net-delay constraints is called *delay budgeting*. The main idea is to distribute slacks from the end-points of each path to constituent nets along the path, such that a zero-slack solution is obtained [12], [43]. 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.

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. The basic idea is to put a higher weight for nets that are more timing critical. Net-weighting techniques have some favorable properties: relatively low complexity, strong flexibility, and easy implementation. As circuit sizes increase and practical timing constraints become increasingly complex, these advantages make the net weighting method more attractive.

There are two principles for assigning net weights. The main principle used in most algorithms is that a timing critical net should receive a heavy weight. For example, VPR [41] uses the following formula to assign weight to an edge e:

$$w(e) = (1 - \text{slack}\ (e)/T)^{\delta} \tag{13}$$

where T is the current longest path delay, and δ is a constant called the *criticality exponent*.

The other principle is *path sharing*: In general, an edge with many paths passing through should have 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 net weights. Another work [36] proposed a solution that distinguishes timing-critical paths from noncritical paths, and scale the impact of all paths by their relative timing criticality. Given a weighting function $D(\text{slack}, T)$, the weight assigned to a particular edge e is

$$w(e) = \sum_{e \in \pi} D(\text{slack}\ (\pi), T) \tag{14}$$

where T is the current longest path delay, and slack (π) is the slack of a timing critical path π.

B. Slack-Derived Edge Weights

It is natural to apply the net weighting method in APlace to perform timing-driven placement.

Our placer uses the following formula to assign weight $w(e)$ to an edge e :

$$w(e) = 1 + \sum_{e \in \pi} \Big(D\big(\text{slack}\ (\pi), T\big) - 1 \Big) \tag{15}$$

where

$$D(s, T) = \begin{cases} (1 - s/T)^{\delta}, & s \le 0 \\ 1, & s \ge 0 \end{cases}. \tag{16}$$

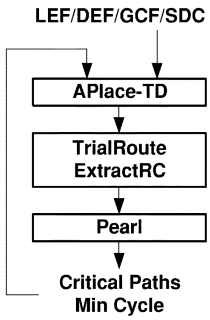

Fig. 7. Flow of timing-driven process in APlace-TD.

Here, δ is the criticality exponent. $T = (1 - u) \cdot \max_\pi\{\text{delay}\,(\pi)\}, \text{slack}\,(\pi) = T - \text{delay}\,(\pi)$ is the slack of path π and u is the expected improvement of the longest path delay after this timing-driven iteration. The timing-driven process may repeat a few iterations. The weight of a net that is always timing critical is accumulated.

C. Timing-Driven Placement Flow

Fig. 7 shows the flow of timing-driven process at the final stage of our placer. The near-finished placement of the placer is sent to TrialRoute (SoC Encounter v3.2) to perform a fast global and detailed routing. RC is extracted based on the routing result. Then, we use a commercial tool, Pearl (SE v5.4) to do static timing analysis (STA), and the resulting critical path delays are imported into the placer to decide net weights based on timing criticality and path sharing. The weighted wirelength objective is then optimized using the Conjugate Gradient solver together with the density objective.

D. Experimental Results

1) Impact of Net Weighting Parameters: Experiments are performed to study the parameters of the net weighting formula described in (15) and (16).

Table IX shows the timing-driven placement runs with varying expected improvements (u's) and criticality exponents (δ's) for an industry testcase, indust1. The circuit is in LEF/DEF/GCF/SDC format and has 7077 cells and 8032 nets.

Again, all experiments are performed on an Intel Xeon server with dual 2.4-GHz CPUs (double-threaded) and 4 GB of memory. Minimum cycle time (MCT) in nanoseconds is reported by the STA to measure performance of timing-driven placements, together with HPWL (in meters) and running times (in minutes) of the placements, and routed wirelength (in meters) and the number of vias of TrialRoute's results.

The value of expected improvement decides how many timing critical paths are considered for the net weighting. For each value of expected improvement, timing-driven placements are performed with criticality exponents between 1 and 19. The MCT improvement in percentage with timing-driven placement is shown in the last column of Table IX. The minimum cycle time initially decreases with the criticality exponent; since wirelength always increases, the minimum cycle time gradually deteriorates when the criticality exponent is larger.

2) Comparison With Industry Placers: Results of our placer are compared with two industry placers: QPlace (SE v5.4) and amoebaPlace (SoC Encounter v3.2) in Table X. QPlace is run on a Sun Ultra10 workstation with a 400-MHz CPU. Timing-driven and nontiming-driven placements are sent to Cadence WarpRoute (SoC Encounter v3.2) to do timing-driven routing. *RC* is then extracted and Pearl (SE v5.4) is used to perform STA.

We use six industry circuits as our testcases. Two of them, mac1 and mac2, are among the ISPD 2001 Circuit Benchmarks [27] that first appeared in [13]. These circuits are also used in [59] as benchmarks for timing-driven placement. Only Verilog files are available for these two cases; they are synthesized with a commercial tool, Cadence BuildGates (v5.12). We use a 0.18-μm standard-cell library as the LEF file and use the values reported in [59] as the clock cycles for the two circuits. The other four testcases are available in complete LEF/DEF/GCF/SDC format.

In Table X, the second and third columns show the number of cells and nets of the industry circuits. For each testcase, nontiming-driven, and timing-driven placement are performed by each placer. For the indust4 circuit, timing-driven QPlace fails because of incompatible timing constraint file format. Placed wirelengths (in meters) and running times (in minutes) of each placement are summarized in the fifth and sixth columns. APlace-TD usually has a better placed wirelength than industry placers; but it is slower.

Timing-driven routing is performed with WarpRoute; overcapacity gcells in percentage, the number of violations, routed wirelength (in meters), the number of vias and running times

TABLE IX
TIMING-DRIVEN RESULTS WITH VARYING EXPECTED IMPROVEMENTS (u's) AND CRITICALITY EXPONENTS (δ's) FOR THE INDUST1 CIRCUIT

		Placement		TrialRoute		STA	
u	δ	WL	CPU	WL	vias	min cycle	impr.
	0	0.45	11	0.59	34582	14.30	0.00
0.10	3	0.45	12	0.58	34467	13.86	3.08
	5	0.45	12	0.59	34557	13.76	3.78
	7	0.45	12	0.59	34563	13.86	3.08
	9	0.45	12	0.59	34659	13.62	4.76
	11	0.45	12	0.59	34691	13.66	4.48
	13	0.45	12	0.59	34777	13.57	5.10
	15	0.45	12	0.59	34605	13.84	3.22
	17	0.45	12	0.59	34624	13.57	5.10
	19	0.45	11	0.59	34480	13.58	5.03
0.20	1	0.45	12	0.58	34354	13.92	2.66
	3	0.45	13	0.59	34492	13.80	3.50
	5	0.45	13	0.59	34561	13.61	4.83
	7	0.45	13	0.59	34660	13.78	3.64
	9	0.45	12	0.59	34657	13.53	5.38
	11	0.46	12	0.60	34937	13.67	4.41
	13	0.46	12	0.60	35101	13.67	4.41
	15	0.46	12	0.60	35187	13.75	3.85
	17	0.46	12	0.61	35359	13.65	4.55
	19	0.47	12	0.61	35179	13.66	4.48
0.30	1	0.45	13	0.59	34268	13.77	3.71
	3	0.45	13	0.59	34482	13.60	4.90
	5	0.46	13	0.60	34668	13.74	3.92
	7	0.47	13	0.61	35166	13.97	2.31
	9	0.48	12	0.62	35559	13.79	3.57
	11	0.49	12	0.63	35772	13.81	3.43
	13	0.50	13	0.64	36093	13.95	2.45
	15	0.52	12	0.68	37232	13.99	2.17
	17	0.54	12	0.69	37632	13.88	2.94
	19	0.61	13	0.80	39339	14.46	-1.12

TABLE X
TIMING-DRIVEN AND NONTIMING-DRIVEN RESULTS OF OUR PLACER FOR SIX INDUSTRY CIRCUITS WITH COMPARISON TO QPlace AND amoebaPlace

			Placement			TD-Routing					STA
ckts	**cells**	**nets**	**placer**	**WL**	**CPU**	**over-cap**	**vios**	**WL**	**vias**	**CPU**	**min cycle**
indust1	7077	8032	QPlace	0.59	7	0.19	0	0.74	41517	2	14.67
			QPlace-TD	0.58	21	0.23	0	0.73	41651	1	15.04
			Amoeba	0.59	1	5.80	109	0.86	49191	10	16.04
			Amoeba-TD	0.61	1	8.77	765	0.88	51379	12	14.91
			APlace	**0.51**	**14**	**0.77**	**0**	**0.67**	**43498**	**1**	**14.28**
			APlace-TD	**0.51**	**11**	**0.89**	**0**	**0.68**	**43595**	**2**	**13.83**
indust2	20094	22973	QPlace	1.19	21	2.16	0	2.01	185462	1	48.92
			QPlace-TD	1.29	80	2.80	0	2.31	195440	1	38.87
			Amoeba	1.38	2	2.35	0	2.12	193166	1	59.98
			Amoeba-TD	1.40	5	2.88	0	2.11	194516	2	46.98
			APlace	**1.31**	**58**	**5.09**	**0**	**2.32**	**213298**	**2**	**34.92**
			APlace-TD	**1.31**	**55**	**5.18**	**0**	**2.34**	**214427**	**3**	**33.60**
indust3	40447	42413	QPlace	0.36	20	0.00	0	0.43	278615	4	27.40
			QPlace-TD	0.34	37	0.00	0	0.41	279545	5	27.20
			Amoeba	0.37	3	0.00	0	0.43	302592	2	27.67
			Amoeba-TD	0.36	5	0.00	0	0.42	306896	2	27.31
			APlace	**0.35**	**119**	**0.07**	**0**	**0.43**	**312729**	**3**	**27.65**
			APlace-TD	**0.34**	**112**	**0.05**	**0**	**0.41**	**310976**	**5**	**27.53**
indust4	35272	37543	QPlace	14.56	22	0.01	9	17.32	334235	7	401.50
			QPlace-TD	fail in QPlace-TD							
			Amoeba	15.08	3	0.01	15	16.80	341286	8	401.76
			Amoeba-TD	15.08	6	0.01	21	16.74	341848	8	402.09
			APlace	**12.84**	**65**	**0.01**	**51**	**15.33**	**345492**	**55**	**401.49**
			APlace-TD	**12.80**	**80**	**0.01**	**49**	**15.30**	**344970**	**75**	**401.28**
mac1	5937	6079	QPlace	0.33	3	1.45	0	0.52	56180	1	4.36
			QPlace-TD	0.33	6	1.41	0	0.52	55996	1	4.66
			Amoeba	0.34	1	0.42	0	0.46	56536	1	4.03
			Amoeba-TD	0.36	1	0.35	0	0.47	57827	1	4.46
			APlace	**0.28**	**6**	**0.14**	**0**	**0.40**	**54932**	**1**	**4.13**
			APlace-TD	**0.28**	**9**	**0.10**	**9**	**0.40**	**55236**	**4**	**4.06**
mac2	21491	21766	QPlace	1.30	11	3.39	0	2.43	216182	2	7.46
			QPlace-TD	1.29	22	2.92	0	2.27	211532	2	7.25
			Amoeba	1.38	1	0.34	0	2.23	214249	2	6.15
			Amoeba-TD	1.48	3	0.40	0	2.18	215438	2	6.64
			APlace	**1.15**	**37**	**0.51**	**9**	**2.13**	**220143**	**3**	**6.26**
			APlace-TD	**1.14**	**38**	**0.41**	**16**	**2.11**	**217971**	**6**	**6.18**

(in minutes) of WarpRoute's results are shown in the seventh through eleventh columns of Table X. Most of the placement results of APlace-TD can be successively routed with good wirelength; finished routings with a small number of violations can be manually fixed. According to the results, average (routed) wirelength improvement of APlace-TD over QPlace is 7.2% (range: −1.2%–7.1%); and average improvement over amoebaPlace is 6.5% (range: −11.1%–23.2%). Compared to nontiming-driven APlace, APlace-TD has a slightly better routed wirelength (0.7% on average).

In the last column, minimum cycle time (in nanoseconds) is reported from the STA tool as the performance measure of timing-driven or nontiming-driven placements.[1] Our placer usually has a better minimum cycle time. Average improvement in minimum cycle time of APlace-TD over QPlace is 9.6% (range: −1.2%–14.8%); and average improvement over amoebaPlace is 8.5% (range: −0.8%–28.5%). Compared to nontiming-driven APlace, APlace-TD improves the minimum cycle time by 2% on average (range: 0.1%–3.8%). The MCT improvements are especially negligible for the indust3 and indust4 circuits; as noted in the footnote above, minimum cycle times of these two circuits are less sensitive with different net weights.

[1]For some testcases, especially mac1 and mac2, timing-driven placements from the industry placement can have worse minimum cycle times than those from nontiming-driven placement. This may be due to improper timing constraints in the testcases—e.g., for "old" designs. We also note that the indust3 and indust4 testcases do not show any significant sensitivity of MCT to the placement.

VII. I/O-CORE COPLACEMENT

IC packaging technologies with peripheral I/O pads have well-known shortcomings: clock/power distribution is constrained, and large parasitics of peripheral I/O pads cause coupling and power issues for off-chip signaling. The area-array I/O regime is projected to eventually dominate IC implementation methodology, affording improved pad count and reliability, and reduced noise coupling.

Area-array I/O presents new challenges to placement tools. Caldwell *et al.* [9] conducted a thorough study of the implication of area-array I/O for placement methods. They determined that with alternating I/O and core placement methods, which

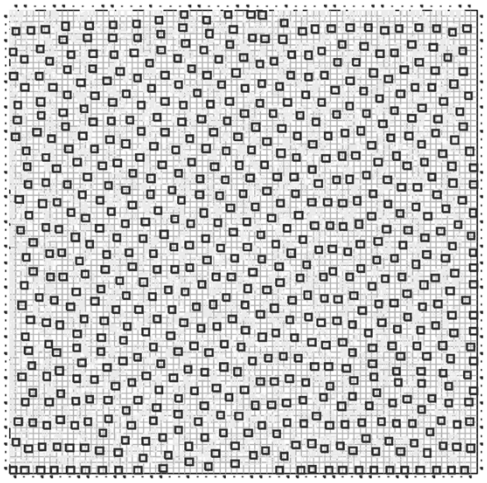

Fig. 8. I/O-core coplacement results of APlace for six industry circuits.

TABLE XI
I/O-CORE CO-PLACEMENT OF APlace FOR MAC1 CIRCUIT

Cases		APlace with IO-Core Co-Placement				
ckts	I/Os	WL_l	I/O disc	disc	iters	CPU (m)
indust1	0	0.71	2.02	1.42	1204	16
	1837	0.79	1.15	2.57	1204	16
indust2	0	1.33	7.31	1.19	1688	58
	258	1.42	3.53	1.29	1688	55
indust3	0	0.33	12.20	1.13	1612	79
	118	0.32	6.51	1.13	1208	64
indust4	0	13.31	13.67	1.08	990	60
	965	13.49	2.07	1.14	1061	68
mac1	0	0.33	4.66	1.45	991	15
	623	0.36	1.04	1.66	849	13
mac2	0	1.34	5.02	1.14	1406	63
	830	1.43	1.81	1.25	1124	58

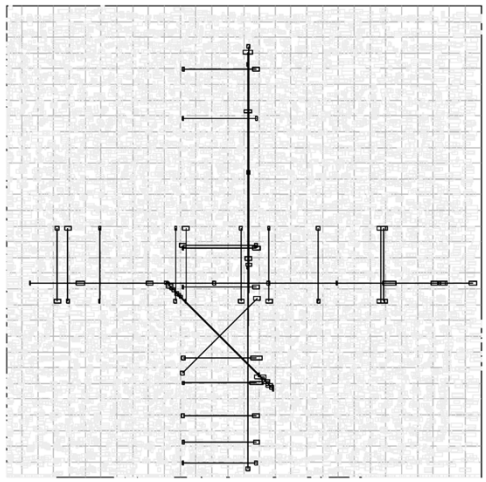

Fig. 9. Placement of APlace with 90 artificial geometric constraints for ibm01-easy circuit.

are often used in practice, the number of iterations needed to achieve good solutions can be surprisingly large. Also, a bad initial I/O placement can seriously handicap subsequent iterations. A simultaneous methodology is proposed in [18], which performs top-down hierarchical placement, with I/Os replaced to each partition by min-cost assignment. However, previous results from the simultaneous methodology are not good.

I/Os and core cells can be simultaneously placed in APlace. I/Os are spread over the placement area, in the same way and at the same time as core cells. The objective function of APlace in (6) is modified as

$$\begin{aligned} &\mathrm{WLWeight} * \mathrm{TotalWL} \\ &\quad + \mathrm{DensityWeight} * \mathrm{DensityPenalty} \\ &\quad + \mathrm{IODensityWeight} * \mathrm{IODensityPenalty} \end{aligned} \tag{17}$$

where the third term drives the spreading of I/Os.

We have tested extensions of APlace to perform I/O-core coplacement on six industry circuits. Cells directly connected to fixed pads are regarded as I/O cells. Specific parameters used for the experiments are: $\epsilon = 30$ and $\mathrm{destDisc} = 1.1$. Fig. 8 shows the placement for the mac1 circuit with 623 I/Os. Core cells and I/Os are displayed as yellow (grey) blocks and pink (dark) blocks, respectively. I/Os are distributed evenly over the placement area in the figure. The results are summarized in Table XI. For each circuit, the placement is performed without and with I/O-core coplacement. The third column shows the wirelength after legalization, in meters. I/O distribution is evaluated by the I/O discrepancy within 1% placement area. According to the results, I/O-core coplacement reduces I/O discrepancy by 61.3% on average, with an average increase of 5.3% in placed wirelength.

VIII. GEOMETRIC CONSTRAINTS

The need to increase the complexity and reduce the cost of electronic systems has greatly accelerated the demand for combining discrete components into ASICs. As more digital circuitry is integrated, the analog components of a system are more likely to represent a bottleneck in the path to size and cost reduction for a system. With increased demand for mixed-signal ASICs, there is a corresponding demand for software tools and design methodologies that increase the productivity of analog and mixed-signal ASIC designs.

Digitally targeted tools are often inadequate to handle the critical and specific requirements of analog layout. Performance of analog circuits is much more sensitive to the details of physical implementation than that of digital circuits. A large number of constraints have to be considered in order to avoid extra design iterations caused by too many parasitic effects. Layout synthesis is often a multiobjective optimization problem where, along with area, wiring length and delay, topological constraints must be taken into account.

Basic geometric constraints for mixed-signal placement include the following categories: fixed components, spacing, alignment, axial symmetry, and nodal symmetry. These categories cover most of the topological requirements in an analog layout system. More complex constraints, such as matching, can be built as combinations of these basic constraints.

Geometric constraints can be handled directly in APlace, since they can be converted to penalty functions and added to the objective function of the placer. Penalty functions for basic geometric constraints are summarized as follows.

TABLE XII
PLACEMENT RESULTS OF APLACE WITH 90 ARTIFICIAL GEOMETRIC CONSTRAINTS

IBM-Dragon	APlace with Constraints				
Ckts	WL	WL_l	disc	iters	CPU (m)
ibm01e	0.54	0.57	1.17	1309	57
ibm02e	1.50	1.55	1.10	1208	70
ibm07e	3.36	3.48	1.16	1049	123
ibm08e	3.78	3.99	1.12	887	130

1) For an x-alignment constraint for cells with coordinates $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, which is expressed as $x_1 = x_2 = \cdots = x_n$, the penalty function is $\sum (x_i - \bar{x})^2$.
2) For an x-spacing constraint between two cells with coordinates $\{(x_1, y_1)$ and $(x_2, y_2)\}$, which is expressed as $L < |x_1 - x_2| < U$, the penalty function is

$$\begin{cases} (|x_1 - x_2| - L)^2, & \text{when } |x_1 - x_2| < L \\ 0, & \text{when } L < |x_1 - x_2| < U \\ (|x_1 - x_2| - U)^2, & \text{when } |x_1 - x_2| > U \end{cases}$$

3) For an axial symmetry constraint with an axis $y = c$ between two cells with coordinates $\{(x_1, y_1)$ and $(x_2, y_2)\}$, which is expressed as $x_1 = x_2$ and $y_1 + y_2 = 2c$, the penalty function is $(x_1 - x_2)^2 + (y_1 + y_2 - 2c)^2$.
4) For a nodal symmetry constraint with a center (c_x, c_y) between two cells with coordinates $\{(x_1, y_1)$ and $(x_2, y_2)\}$, which is expressed as $x_1 + x_2 = 2c_x$ and $y_1 + y_2 = 2c_y$, the penalty function is $(x_1 + x_2 - 2c_x)^2 + (y_1 + y_2 - 2c_y)^2$.

Constraint handling is implemented in APlace, and experiments are performed for each IBM-PLACE easy circuit with a manually specified combination of 40 spacing constraints, 20 alignment constraints, 20 axial symmetry, and 10 nodal symmetry constraints. Cells in the artificial constraints are randomly selected from the netlist. Specific parameters used for the experiments are: $\epsilon = 30$ and $\mathrm{destDisc} = 1.1$. Fig. 9 shows the placement with 90 geometric constraints for the ibm01-easy circuit. Constraints are shown as cells connected by colored (dark) lines. Cells connected by blue lines have alignment constraints among them; cells connected by black lines have axial symmetry and spacing constraints; and cells connected by red lines have nodal symmetry and spacing constraints. We can see from the figure that geometric constraints are closely enforced. The placement results are summarized in Table XII. The second and third columns show the wirelength before and after legalization in meters. Compared to placement results without constraints, the legalized wirelength is increased by 8.2% on average.

IX. CONCLUSION AND FUTURE WORK

We have implemented APlace, an analytic placer based on ideas described in the recent patent of Naylor *et al.* [44], and have conducted in-depth analysis of characteristics and results of the placer. The implementation is successful: placed and routed wirelengths outperform QPlace, FastPlace, Dragon and Capo. We also extend the basic placer to perform top-down hierarchical placement, congestion-directed placement, mixed-size placement, timing-driven placement, I/O-core coplacement, and constraint handling for mixed-signal contexts. Our work empirically demonstrates that the APlace analytic framework is a general and extensible platform for "spatial embedding" tasks across many aspects of system physical implementation.

We are currently working on speedup of APlace. The efforts include: 1) usage of lookup tables when computing density penalties and exponential function; 2) implementation of a faster pseudo-Newton solver; and 3) application of the Augmented Lagrangian method for the constrained optimization problem.

Our other ongoing research directions include: 1) extension of the placer to power or IR drop directed placement; 2) extension to three-dimensional placement; 3) extension to thermal-directed placement; and 4) devising a unified analytic placement approach that can simultaneously address congestion, timing, power, and wirelength at a level beyond the existing state of the art.

REFERENCES

[1] S. N. Adya and I. L. Markov, "Consistent placement of macroblock using floorplanning and standard-cell placement," in *Proc. Int. Symp. Phys. Design*, 2002, pp. 12–17.
[2] S. N. Adya, I. L. Markov, and P. G. Villarrubia, "On whitespace in mixed-size placement and physical sysnthesis," in *Proc. Int. Conf. Computer-Aided Design*, 2003, pp. 311–318.
[3] C. J. Alpert, T. Chan, A. B. Kahng, I. Markov, and P. Mulet, "Faster minimization of linear wirelength for global placement," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 17, no. 1, pp. 3–13, Jan. 1998.
[4] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. Design Automation Conf.*, 1997, pp. 530–533.
[5] C. J. Alpert, G.-J. Nam, and P. G. Villarrubia, "Free space management for cut-based placement," in *Proc. Int. Conf. Computer Aided Design*, Nov. 2002, pp. 746–751.
[6] R. Baldick, A. B. Kahng, A. Kennings, and I. L. Markov, "Function smoothing with applications to VLSI layout," in *Proc. Asia South Pacific Design Automation Conf.*, Jan. 1999, pp. 225–228.
[7] A. E. Caldwell, A. B. Kahng, A. A. Kennings, and I. L. Markov, "Hypergraph partitioning for VLSI CAD: Methodology for reporting, and new results," in *Proc. Design Automation Conf.*, Jun. 1999, pp. 349–354.
[8] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placements?," in *Proc. Design Automation Conf.*, 2000, pp. 477–482.
[9] A. E. Caldwell, A. B. Kahng, S. Mantik, and I. L. Markov, "Implications of area-array I/O for row-based placement methodology," in *Proc. IEEE Symp. IC/Package Design Integr.*, Feb. 1998, pp. 93–98.
[10] UCLA/UMICH Physical Design Tools [Online]. Available: http://vlsicad.eecs.umich.edu/BK/PDtools
[11] C. C. Chang, J. Cong, and X. Yuan, "Multi-level placement for large-scale mixed-size IC designs," in *Proc. Asia South Pacific Design Automation Conf.*, 2003, pp. 325–330.
[12] C. Chen, X. Yang, and M. Sarrafzadeh, "Potential slack: An effective metric of combinational circuit performance," in *Proc. Int. Conf. Computer-Aided Design*, 2000, pp. 198–201.
[13] Y.-C. Chou and Y.-L. Lin, "A performance-driven standard cell placer based on a modified force-directed algorithm," in *Proc. Int. Symp. Physical Design*, 2001, pp. 24–29.
[14] Dragon [Online]. Available: http://er.cs.ucla.edu/Dragon/
[15] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. Design Automation Conf.*, 1998, pp. 269–274.
[16] H. Etawil, S. Areibi, and A. Vannelli. ARP: A convex optimization based method for global placement [Online]. Available: http://www.mat.univie.ac.at/~neum/glopt/mss/EtaAV01.pdf
[17] H. Etawil, S. Areibi, and A. Vannelli, "Attractor-repeller approach for global placement," in *Proc. Int. Conf. Computer-Aided Design*, 1999, pp. 20–24.
[18] T. Gao, C. L. Liu, and K. C. Chen, "A performance driven hierarchical partitioning placement algorithm," in *Proc. Eur. Design Automation Conf.*, Sep. 1993, pp. 33–38.
[19] VLSI CAD Bookshelf [Online]. Available: http://www.gigascale.org/bookshelf/

[20] B. Halpin, C.-Y. R. Chen, and N. Sehgal, "Timing driven placement using physical net constraints," in *Proc. Design Automation Conf.*, 2001, pp. 780–783.
[21] T. Hamada, C. K. Cheng, and P. M. Chau, "Prime: A timing-driven placement tool using a piecewise linear resistive network approach," in *Proc. Design Automation Conf.*, 1993, pp. 531–536.
[22] D. Hill, "Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design," U.S. Patent 6370673, Apr. 2002.
[23] D. J. Huang and A. B. Kahng, "Partitioning-based standard cell global placement with an exact objective," in *Proc. Int. Symp. Phys. Design*, 1997, pp. 18–25.
[24] B. Hu and M. Marek-Sadowska, "FAR: Fixed-points addition & relaxation based placement," in *Proc. Int. Symp. Phys. Design*, 2002, pp. 161–166.
[25] IBM-PLACE 2.0 Benchmark Suites [Online]. Available: http://er.cs.ucla.edu/benchmarks/ibm-place2/
[26] ISPD'02 Mixed-Size Placement Benchmarks [Online]. Available: http://vlsicad.eecs.umich.edu/BK/ISPD02bench/
[27] ISPD 2001 Circuit Benchmarks [Online]. Available: http://nthucad.cs.nthu.edu.tw/~ycchou/benchmark/placement.htm
[28] M. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," in *Proc. Design Automation Conf.*, 1989, pp. 370–375.
[29] A. B. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," in *Proc. Int. Symp. Phys. Design*, 2004, pp. 18–25.
[30] ——, "An analytic placer for mixed-size placement and timing-driven placement," in *Proc. Int. Conf. Computer Aided Design*, 2004, pp. 565–572.
[31] A. B. Kahng and X. Xu, "Accurate pseudo-constructive wirelength and congestion estimation," in *Proc. ACM Int. Workshop System-Level Interconnect Prediction*, 2003, pp. 61–68.
[32] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI design," in *Proc. Design Automation Conf.*, 1997, pp. 526–529.
[33] A. A. Kennings and I. L. Markov, "Smoothening max-terms and analytical minimization of half-perimeter wirelength," *VLSI Design*, vol. 14, no. 3, pp. 229–237, 2002.
[34] A. A. Kennings and I. L. Markov, "Analytical minimization of half-perimeter wirelength," in *Proc. Asia South Pacific Design Automation Conf.*, Jan. 2000, pp. 179–184.
[35] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh, and P. H. Madden, "Recursive bisection based mixed block placement," in *Proc. Int. Symp. Phys. Design*, 2004, pp. 84–89.
[36] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. Int. Conf. Computer-Aided Design*, 2002, pp. 10–14.
[37] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 10, no. 3, pp. 356–365, Mar. 1991.
[38] J.-M. Li, J. Lillis, L.-T. Liu, and C.-K. Cheng, "New spectral linear placement and clustering approach," in *Proc. Design Automation Conf.*, 1996, pp. 88–93.
[39] J.-M. Li, J. Lillis, and C.-K. Cheng, "Linear decomposition algorithm for VLSI design applications," in *Proc. Int. Conf. Computer-Aided Design*, 1995, pp. 223–228.
[40] C. Li and C.-K. Koh, "On Improving Recursive Bipartitioning-Based Placement," Purdue Univ., West Lafayette, IN, Tech. Rep. TR-ECE-03-14, 2003.
[41] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *Proc. ACM Symp. FPGAs*, 2000, pp. 203–213.
[42] K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming* [Online]. Available: http://ioe.engin.umich.edu/books/murty/linear_complementarity_webbook/
[43] R. Nair, C. L. Berman, P. Hauge, and E. J. Yoffa, "Generation of performance constraints for layout," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 8, no. 8, pp. 860–874, Aug. 1989.
[44] W. Naylor *et al.*, "Non-Linear Optimization System and Method for Wire Length and Delay Optimization for an Automatic Electric Circuit Placer," U.S. Patent 6301693, Oct. 2001.
[45] S.-L. Ou and M. Pedram, "Timing-driven placement based on partitioning with dynamic cut-net control," in *Proc. Design Automation Conf.*, 2000, pp. 472–476.
[46] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdhary, and B. Halpin, "Timing driven force directed placement with physical net constraints," in *Proc. Int. Symp. Phys. Design*, 2003, pp. 60–66.
[47] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A performance driven placement for small-cell ICs," in *Proc. Int. Conf. Computer-Aided Design*, 1991, pp. 48–51.
[48] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits," in *Proc. Design Automation Conf.*, 1995, pp. 211–215.
[49] R. S. Tsay and J. Koehl, "An analytic net weighting approach for performance optimization in circuit placement," in *Proc. Design Automation Conf.*, 1991, pp. 620–625.
[50] S. S. Sapatnekar and S. M. Kang, *Design Automation for Timing-Driven Layout Synthesis*. Norwell, MA: Kluwer, 1993.
[51] G. Sigl, K. Doll, and F. M. Johannes, "Analytical placement: A linear or a quadratic objective function?," in *Proc. Design Automation Conf.*, 1991, pp. 427–431.
[52] W.-J. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 14, no. 3, pp. 349–359, Mar. 1995.
[53] R. S. Tsay, E. Kuh, and C. P. Hsu, "PROUD: A sea-of-gates placement algorithm," *IEEE Design Test Comput.*, vol. 5, no. 6, pp. 44–56, Dec. 1988.
[54] P. Villarrubia, G. Nusbaum, R. Masleid, and P. T. Patel, "IBM RISC chip design methodology," in *Proc. Int. Conf. Comput. Design*, 1989, pp. 143–147.
[55] N. Viswanathan and C. C.-N. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. Int. Symp. Phys. Design*, 2004, pp. 26–33.
[56] J. Vygen, "Algorithms for large-scale flat placement," in *Proc. Design Automation Conf.*, 1997, pp. 746–751.
[57] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "A standard-cell placement tool for designs with high row utilization," in *Proc. Int. Conf. Comput. Design*, Sep. 2002, pp. 45–47.
[58] ——, "Routability driven white space allocation for fixed-die standard-cell placement," in *Proc. Int. Symp. Phys. Design*, 2002, pp. 42–47.
[59] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "Timing-driven placement using design hierarchy guided constraint," in *Proc. Int. Conf. Computer-Aided Design*, 2002, pp. 177–180.
[60] M. C. Yildiz and P. H. Madden, "Improved cut sequences for partitioning based placement," in *Proc. Design Automation Conf.*, 2001, pp. 776–779.

Andrew B. Kahng (A'89–M'03) received the A.B. degree in applied mathematics is from Harvard College and the M.S. and Ph.D. degrees in computer science are from University of California (UC) at San Diego, La Jolla.

From 1989 to 2000, he was a member of the Computer Science Department, UC, Los Angeles. He is a Professor of Computer Science Engineering and Electrical and Computer Engineering at UC. He has published over 200 papers in the very large scale integration computer-aided design literature, receiving three Best Paper awards and an NSF Young Investigator award. His research is mainly in physical design and performance analysis of very large scale integration (VLSI), as well as the VLSI design-manufacturing interface. Other research interests include combinatorial and graph algorithms, and large-scale heuristic global optimization. Since 1997, he has defined the physical design roadmap for the International Technology Roadmap for Semiconductors (ITRS), and since 2001 has chaired the U.S. and international working groups for Design technology for the ITRS. He has been active in the MARCO Gigascale Silicon Research Center since its inception. He was also the founding General Chair of the ACM/IEEE International Symposium on Physical Design, and cofounded the ACM Workshop on System-Level Interconnect Planning.

Qinke Wang (S'03) received the B.S. degree in computer science and technology and the M.S. degree in computer software from Tsinghua University, Beijing, China, in 1998 and 2001, respectively. He is currently pursuing the Ph.D. degree at the University of California at San Diego, La Jolla.

His current research interests include analytical placement, multiproject wafer and reticle cost optimization, and interconnect architectures.