Mixed-Size Placement via Line Search

Kristofer Vorwerk and Andrew Kennings
Department of E&CE
University of Waterloo
Waterloo, Ontario, Canada
{kpvorwer,akenning}@cheetah.vlsi.uwaterloo.ca

***Abstract*— We describe a remarkably simple yet very effective line search technique for cell placement. Our method "corrects" errors in force scaling by sampling different force weights in each iteration of placement and selecting the best candidate placements based on an objective function. Our technique is not only very fast, but it does away with the need for the *ad hoc* scaling that has plagued prior force-directed methods. We describe the implementation of our method within a multi-level flow and show that it can achieve good wire lengths with competitive run-times compared to other academic tools. Specifically, we produce placements with 12% and 15% better HPWL than `FengShui 5.0` and `Capo 9.1`, respectively, on the `ICCAD04` mixed-size benchmarks, while presenting run-times that are 37% faster than `Capo 9.1`.**

I. Introduction

Research in very large scale integrated placement continues to receive significant attention as both the complexity and size of deep sub-micron designs increase. Historically, global placement techniques have focused on simulated annealing [1] and min-cut partitioning-based methods [2, 3]. Despite offering excellent results, simulated annealing has been largely abandoned due to its exponential performance scaling. Top-down min-cut partitioning techniques, on the other hand, have remained a touchstone against which alternative methods have compared due to their attractive balance of quality and run-time. However, there has been a shift toward analytic placement approaches which promise a faster, higher-quality, and more scalable means of placing modern designs. In contrast with min-cut techniques, analytic methods are often better-able to model specific circuit characteristics, such as pin offsets, and appear to be more amenable to cell sizing, buffer insertion, and ECO.

Analytic methods generally minimize an unconstrained, smoothed wire length objective, but in so doing, may introduce large amounts of cell overlap into the placement. Several methods have been proposed to deal with this. The analytic technique presented in [4] uses a log-sum-exponential approximation to half-perimeter wire length (HPWL), coupled with a non-convex penalty function to reduce unevenness in cell distribution over the placement area. Similarly, [5] uses a log-sum-exponential wire length approximation but employs an inverse Laplacian transform to derive a smooth density function which is used to remove overlap. However, these techniques can be *highly* complex or potentially legally encumbered (e.g., [4, 6]). The implementations of such methods are non-trivial and can be difficult to reproduce.

Force-directed placement methods perturb the objective formulation based on cells' area distribution in order to move cells from high-density regions to low-density regions. In [7–9], spreading forces are repeatedly added to a quadratic objective. Similarly, fixed point approaches [10, 11] use specially-located attractors to pull cells from dense to less-dense regions of the placement area.

Force-based placers do not generally rely on an objective function to assess the quality of placements in each iteration—these methods strive to generate a placement which is good "by construction". The lack of a clear objective function can be a source of instability in these methods, as it requires reliance on heuristics to keep quality "in check" by preventing unexpected jumps in wire length between iterations.

Moreover, choosing appropriate force weights can be difficult in these methods, as wire length can be seriously damaged when the cells are spread out by large forces. In [8], an *ad hoc* scheme is described to control force weighting (and, by extension, the rate of cell spreading). Early on, weights are kept small to promote wire length minimization and are slowly increased to encourage overlap removal. Nevertheless, this kind of *ad hoc* scaling can lead to large iteration counts and, therefore, long run-times.

In this paper, we introduce a remarkably *simple* yet very *effective* line search technique for cell placement. Our method "corrects" errors in force scaling by testing different force weights in each iteration of placement and selecting the best candidate placements based on a straightforward objective function. The line search that we describe is not only very fast, but it does away with the need for the *ad hoc* force scaling that has plagued prior methods [7, 8]. We describe the implementation of our line search-directed placer, `LSD`, which is loosely based upon the open-source tool `FDP`, and show that it can achieve competitive wire lengths with reduced run-times compared to other academic approaches.

The rest of this paper is organized as follows. Section II presents a background of relevant analytic placement techniques. Section III describes our line search approach. Section IV discusses the incorporation of this method within a multi-level flow, as well as further enhancements to allow for I/O assignment and cell rotation. Finally, Section V presents numerical results, and Section VI offers concluding remarks.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant #203763-03, and a grant from Altera Corporation.

0-7803-9254-X/05/$20.00 ©2005 IEEE.

II. Background

A circuit is typically modeled as a hypergraph $G_h(V_h, E_h)$ with vertices $V_h = \{v_1, v_2, \cdots, v_n\}$ representing cells and hyperedges $E_h = \{e_1, e_2, \cdots, e_m\}$ corresponding to signal nets. Vertices are weighted by cell area while hyperedges are weighted according to criticalities or multiplicities. Vertices are either free or fixed. Cell placements in the x- and y-directions are captured by placement vectors $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{y} = (y_1, y_2, \cdots, y_n)$.

The quadratic optimization problem (QP) can be used to minimize wire length and is formulated (in the x-direction) by

$$\min_x \sum_{i,j} a_{ij}(x_i - x_j)^2 = \min_x \frac{1}{2}\mathbf{x}^T \mathbf{Q}_x \mathbf{x} + \mathbf{c}_x^T \mathbf{x} + \mathbf{d}_x \qquad (1)$$

where a_{ij} represents the weight of the edge connecting cells i and j in the weighted graph representation of the circuit. The matrix $\mathbf{Q}_x$ is the Hessian which encapsulates the hyperedge connectivities. The vector $\mathbf{c}_x$ is a result of fixed cell-to-free cell connections, and the vector $\mathbf{d}_x$ is a result of fixed cell-to-fixed cell connections. In this formulation, cell overlap is ignored, and the vector $\mathbf{x}$ provides only relative cell positions.

In [8], the system of equations is modified by including an additional vector of forces in each iteration of the placement. The force vector is derived from the distribution of cells throughout the placement region. It perturbs the placement to remove overlap by "pushing" cells away from regions of high density and "pulling" cells toward regions of low density. That is, at iteration i, the cell positions are determined from the system of equations given by

$$\mathbf{Q}_x \mathbf{x}_i + \mathbf{c}_x + \sum_{l=1}^{i-1} \kappa_l \mathbf{f}_l + \kappa_i \mathbf{f}_i = 0 \qquad (2)$$

where $\mathbf{f}_i$ represents the spreading forces computed at iteration i and κ_i represents the weighting with respect to wire length. At each iteration, forces throughout the placement region are computed using an analogy similar to charge attraction or repulsion in an electric field.

In FDP, a more linearized placement is achieved by employing the BoxPlace heuristic [9] to move cells to the median locations of their connected nets. This type of re-placement ignores spreading forces and, in effect, performs a relative reordering by lifting cells over top of other cells in order to situate them in more favorable positions.

III. Line Search

In this section, we describe our line search placement technique. The pseudocode for this method is presented in Figure 1. Our approach entails computing spreading forces, computing wire length-minimizing forces, and moving cells in a weighted step in the direction of each force, where the weights are determined using a *line search* and an *objective function*.

A. Computing Forces

During each call to LinePlace, we compute spreading forces using a quad-tree [8]. Given a current placement, cell area is inserted into all levels of the quad-tree based on cell position. Then, for each bin in the bottom level of the quad-tree, the multipole forces acting on the bin are accumulated using interaction lists and near neighbors. Finally, the forces for a given cell are calculated by summing the individual forces upon the bins which that cell overlaps in the bottom level of the quad-tree. Spreading forces are then normalized by dividing each x- and y-component by the l^2-norm of the force vector.

```
1  Procedure: LinePlace
2  Inputs: Force ratio (α), "sliding" force multiplier (slideWt)
3  begin
4    Compute spreading forces, f_x^S, f_y^S in the x,y directions;
5    Compute wire length forces, f_x^WL, f_y^WL in the x,y directions;
6    Combine the forces: f^U ← α × f^S + (1 − α) × f^WL;
7    Orig_SM ← spread metric of current placement;
8    Orig_HPWL ← HPWL of current placement;
9    for each force multiplier γ do
10     Save cells' positions;
11     curWt ← γ × slideWt;
12     for each cell i ∈ V_h do
13       //Update cells positions in x,y
14       x'(i) ← x(i) + curWt × f_x^U(i);
15       y'(i) ← y(i) + curWt × f_y^U(i);
16     od
17     New_SM ← spread metric of new placement;
18     New_HPWL ← HPWL of new placement;
19     score ← β × New_SM / Orig_SM + (1 − β) × New_HPWL / Orig_HPWL;
20     Record the score, curWt, and the current placement;
21     Restore cells' positions;
22   end
23   Restore cell positions from the best (lowest) score found;
24   Set slideWt to the curWt that yielded the best placement;
```

Fig. 1. Pseudocode for the line search placement technique which places cells by moving them by a weighted amount in the direction of forces.

We have found that spreading forces, by themselves, are excellent at reducing overlap but poor at producing placements with good HPWL. Thus, after deriving spreading forces from the quad-tree, we solve a QP to derive a *new* set of "deflected" spreading forces. To accomplish this, we first compute the force gradient required to hold the current cells in place. To this gradient, we add the *new* spreading forces ($\mathbf{f}^S$) to the QP, as in (2), using a constant multiplier $\kappa = 1$. The solution to this perturbed QP yields a set of new positions for the cells in the netlist; however, the cells are *not* moved to these new positions—rather, the new positions are subtracted from the *original* positions of the cells to yield a new set of spreading forces. Thus, the QP accomplishes a *deflection* of the forces and yields new spreading forces which, empirically, result in less harmful wire length. In other words, we effectively solve a force-directed placement iteration along the lines of [7, 8], but do not specifically accept the returned solution as the new placement. Rather, we take this placement only to provide a *suggested* direction in which cells can move to reduce overlap,

though the length of the step in this direction is yet unknown.[1]

The spreading forces derived by our quadratic "deflection" technique are not sufficient, by themselves, to achieve high-quality placements since the quadratic objective tends to be sufficiently imprecise [5, 12]. To compensate, we compute *additional* wire length-minimizing forces using BOXPLACE [8, 9]. In our approach, BOXPLACE is used to compute new cell locations which reduce HPWL. As with the QP deflection, however, cells are not actually moved by BOXPLACE—rather, the new cell positions are subtracted from the cells' original positions to yield a directional "wire length" force.

The magnitudes of the individual components of the wire length force tend to be much larger than the magnitudes of the components of the corresponding spreading forces. We scale the wire length force such that, for each cell, the components possess no greater a magnitude than the cell's corresponding spreading force components. This helps to equalize the distance that a cell would move as a result of either force. Next, we add the spreading and wire length forces together to create a single, "unified" force, as shown in Figure 2. The parameter α controls the preference between spreading and wire length minimization—as the placement progresses, it allows us to "rotate" the unified force toward either HPWL minimization or overlap reduction.

B. Line Searching over the Force Weights

The unified force vector indicates the directions in which to move cells, but does not indicate *how far* to move them to achieve a good trade-off in overlap reduction versus increase in HPWL. In other words, it is unclear how to scale the absolute magnitudes of the unified force components to achieve good spreading without ruining wire length. To this extent, we use a line search to discover how far to step cells in the direction of the forces in each iteration.

Our technique tests 20 different force weight multipliers to determine the best possible step size in which to move the cells. For each candidate multiplier γ (which were determined empirically), each force component is multiplied by γ and by a "sliding" force weight (`slideWt`), and move the cells in the direction of the resultant (scaled) force. Subsequently, the HPWL is measured and a "spread metric" for the placement is computed [8]. The quality of the placement is assessed by comparing the ratios of the new and original HPWLs and spread metrics:

$$\text{score} = \beta \times \frac{\texttt{New\_SM}}{\texttt{Orig\_SM}} + (1-\beta) \times \frac{\texttt{New\_HPWL}}{\texttt{Orig\_HPWL}}. \qquad (3)$$

In this formulation, β controls the preference between HPWL and spread metric weighting (and was determined empirically). After recording the objective function value, the cells' positions are restored, and the next force weight multiplier is tested, as above.

Once all points have been tested, the placement with the best score is selected. The sliding force weight (`slideWt`) is then set to the force weight (`curWt`) that yielded this best placement. In this manner, our line search implements a "sliding window" that tests sets of points within a specific range around the current force weight. This allows the technique to consider a wider range of potential force weights throughout the placement process without having to test a larger number of weights in any individual iteration.

One call to LINEPLACE is extremely fast, generally requiring less than a second to complete on `ibm18` (on a Pentium 3.2 GHz machine). This performance is due to the fact that spreading and wire length forces need only be computed once—the individual iterations of the line search simply measure incremental changes to the spread metric and HPWL. Furthermore, the technique is amenable to parallel computation because the forces are computed separately, as is the evaluation of the HPWL and spread metric. Thus, parallel threads could be employed to compute each force and to evaluate the objective function criteria; we view this opportunity for parallelism as being especially important with the introduction of multi-core architectures [13].

In addition, the line search offers a significant advantage over the work of [7–9] because it implements an easily-tunable objective function which can be geared toward high-performance placement (by encouraging faster spreading) or toward high-quality placement (by preferring lower HPWL). Since placements are tested against an objective function, LINEPLACE can find the best force weight in which to move cells, thereby accounting for (and "correcting") small errors in the forces. Furthermore, our line search can be extended to account for additional objectives (such as timing and power) simply by computing new forces and modifying the objective function accordingly.

[1] We note that, in the absence of fixed cells in a circuit, our placement method still works—we simply avoid QP-deflected force computation and rely entirely on forces from the quad-tree.

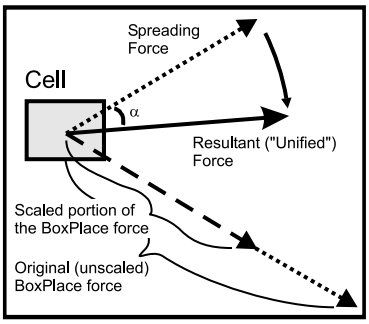

Fig. 2. Illustration of how spreading and wire length forces are combined into a single, unified force. The unified force can be "rotated" during placement via the parameter α.

IV. MULTI-LEVEL PLACEMENT FLOW

We have found that a multi-level clustering approach can significantly improve `LSD`'s placement quality. We have found that multi-level clustering minimizes the negative impact of spreading forces by helping to keep highly-connected cells together. The line search method from Section III has been integrated into a multi-level flow based on `FDP` [9], which we now briefly describe.

A. Multi-Level Placement

The pseudocode for our multi-level placement flow is presented in Figure 3. A QP is first solved to determine the initial positions of cells in the flat netlist. The flat netlist is then clustered (using multiple passes) to 1000 cells via the physical First Choice heuristic described in [9]. At most 40% of cells are aggregated in one pass, so several "levels" of clustered netlists are created, with the bottom-most netlist containing approximately 1000 cells and the top-most netlist being the flat netlist. At each level, clusters are positioned at a weighted average location of their contained cells.

The most-clustered netlist is then iteratively placed to a stopping value of approximately 30% overlap. In each inner loop, LINEPLACE is used to place cells, as previously described. The parameter α controls the "rotation" of the unified force so that it points toward either wire length or overlap minimization. In our implementation, we initially set α to yield a 60%/40% mix between spreading and wire length minimization, and then slowly increase it to favor 100% spreading forces as the placement progresses.

Once the desired spread metric has been obtained, the netlist is declustered, with cells placed at the centers of their former clusters. BOXPLACE is called to move cells in the *flat* netlist to improve wire length. Subsequently, the entire circuit is *reclustered* up to a maximum of *one less* level than the number of levels required for the previous clustering tree. As before, line search-based placement is used to reduce overlap between the clusters to approximately 30% and the process repeats. The placer proceeds to legalization when it has placed the flat netlist to approximately 30% overlap.

Two points are worth noting about this approach. First, by repeatedly decrementing the maximum number of levels of clustering, our approach places increasingly larger netlists until it ultimately places the original (flat) netlist. Second, the use of BOXPLACE and reclustering allow the physical clustering technique to more accurately decide which cells should be paired in subsequent clusters, which, in turn, improves the quality of the clustering.

```
1   Procedure: MULTI-LEVEL PLACEMENT
2   begin
3     Determine initial cell positions by solving a QP;
4     NumLevels ← ∞;
5     while NumLevels > 1 do
6       Recluster flat netlist to at most NumLevels levels;
7       NumLevels ← number of levels in current clustering tree;
8       metricTarget ← current spread metric - 3% overlap;
9       Set α to an initial value (determined empirically);
10      slideWt ← 1;
11      while true do
12        call LINEPLACE(α, slideWt);
13        if spread metric shows less than ≈ 30% overlap then
14          break ;
15        else if spread metric ≤ metricTarget then
16          Perform cell orientation optimization;
17          call BOXPLACE to reposition cells and I/Os;
18          Use linear assignment to reassign I/Os;
19          metricTarget ← current spread metric - 3% overlap;
20        fi
21        Increase α (based on an experimentally-determined rate);
22      od
23      Decluster netlist (placing cells at centers of clusters);
24      call BOXPLACE on flat netlist;
25      NumLevels ← NumLevels − 1;
26    od
27    Legalization and detailed improvement;
28  end
```

Fig. 3. Pseudocode for our multi-level flow.

B. Movable I/Os

In many academic benchmark suites, I/Os are scattered randomly around the periphery of the core [14, 15]. We have devised a means of automatically reassigning and improving the I/Os to reduce overall wire length based on a linear assignment problem. Since the Bookshelf file format does not specify a set of valid I/O positions, we assume that the valid locations for I/OS are the set of positions in which they initially appear. That is, we do not actually move pad locations, but reassign the I/Os among the set of valid positions to improve wire length. The edge costs in the assignment problem model the bounding box cost of assigning an I/O to a candidate position.

We use the Goldberg-Tarjan push-relabel, minimum-cost flow solver [16]. While this technique has a worst-case bound of $O(|V|^2|E|\log|V|)$, it tends to be very fast in practice. Furthermore, we note that the sizes of these assignment problems are generally well-bounded, as circuits do not typically possess more than 1000 placeable I/Os. After solving the flow problem, I/Os are fixed in their assigned locations and the placer continues as usual.

To further enhance the quality of the assignment, we allow I/Os to move into the core (along with other cells) during the occasional calls to BOXPLACE (after every 3% improvement in overlap). This has the effect of repositioning I/Os in better locations to reduce wire length. Subsequently, the minimum-cost linear assignment formulation is used to snap I/Os back to the nearest valid positions along the periphery.

By occasionally repositioning with BOXPLACE and snapping using linear assignment, the I/Os are continually relocated in more favorable positions as the design spreads. Empirically, we have found that repositioning I/Os can yield a $\approx 5\%$ average improvement in wire length. Moreover, we note that this approach is applicable to array-style I/O architectures in which I/Os are located throughout the placement area (and not just around the periphery).

C. Cell Rotation

In our flow, cells are greedily rotated and flipped to improve wire length after every $\approx 3\%$ improvement in the spread metric. Once the spread metric indicates that there exists less than $\approx 50\%$ overlap remaining in the design, cells are only allowed to flip (but not rotate), as rotating can re-introduce large overlaps and cause convergence problems if performed at the end of placement. We also note that BOXPLACE and the QP can account for pin offsets as part of their numerical formulation, and this helps to improve quality on designs with non-zero offsets.

D. Legalization and Detailed Improvement

Our global placements are not valid in that residual overlap may be present—movable objects may overlap with fixed objects or, indeed, with each other. We begin by performing a Tetris-like legalization along the lines of [2]. We legalize toward the left edge of the layout area[2], snapping movable objects to their closest row. We keep track of the current working edge of each row to estimate if a movable object must be shifted to the right. This legalization serves to assign movable objects to rows, but is not sufficient to create a feasible placement as some movable objects may extend off the right edge of the placement area. To obtain feasibility, we use ideas from timing analysis and cell rippling.

We begin by creating buckets of cells. Multiple adjacent standard cells are inserted into a bucket to form a "cluster of cells" based on their row assignments and x-positions. Large cells (e.g., macro cells) are inserted into their own buckets. The widths of the buckets are determined by the widths of the cells contained. Based on the row assignments and spanned rows of each bucket, a constraint graph can be constructed in the x-direction from left to right. The left and right edge of the placement region is then considered a "primary input" and "primary output", respectively, for the graph. Similarly, the left and right edges of fixed obstacles are also translated into primary inputs and primary outputs. A "timing analysis" can be performed on this graph to yield "paths" of buckets (and their cells) that do not fit within the placement area or between adjacent fixed objects. Such buckets will be identified by the timing analysis as buckets with negative slack. Finally, we consider these negative slack buckets in turn, and perform cell rippling [17] to shift movable objects from buckets with negative slack into buckets with positive slack. As cells are rippled, we update the "timing analysis" incrementally and terminate once there exist no buckets with negative slack.

Once we have a legalized placement with no overlap and no placement violations, we proceed to detailed improvement. Presently, our detailed improvement consists of greedy swapping of same-sized cells (as this avoids the re-introduction of overlap) and single row branch-and-bound optimization. We do not employ methods for multiple row optimizations [2] or for whitespace optimization although we believe that the future incorporation of such techniques would improve our results.

V. Numerical Results

We compare our line search-directed placer, `LSD`, with and without I/O reassignment, to other well-known academic tools (including `Capo 9.1` [3], `FengShui 5.0` [2], `FastPlace 1.0` [11], and `Dragon 3.01` [18]) on both standard cell and mixed-size problems.[3] We have run all placers with default command line parameters on a 3.2 GHz Xeon with 2 GB RAM running `Redhat Linux`. All run-times are reported in minutes, and wire lengths are reported as pin-to-pin HPWL (divided by 10^6).

A. Standard Cell Circuits

Our first set of experiments focuses on standard cell placement problems obtained by modifying the `ISPD02 IBM-MS` Mixed-size Placement Benchmarks [15]. These benchmarks were modified by shrinking macro cells to standard cell dimensions and adjusting the layouts to maintain a chip aspect ratio close to 1.0 with 5% white space.[4] Results on standard cell placement problems are presented in Table I. Comparisons to `FengShui` are not available because it crashed on all designs. Some `FastPlace` results (as noted in the table by an asterisk) are *not legalized* because the `FastPlace` detailed placer did not complete after 5 hours.

We observe that `LSD`, without I/O reassignment, achieves results that are, on average, 4% better than `Capo`, 16% better than `FastPlace`, and only 2% worse than `Dragon`. Our run-times are competitive, being 25% faster than `Capo`, 81% faster than `Dragon`, though admittedly slower than `FastPlace`. Yet, with I/O assignment, our tool offers 16% better quality than `FastPlace` and 6% better quality than `Capo` for a modest 10% increase in run-time. It is interesting to note the unstable behavior of `FastPlace` and `FengShui` on these circuits given that only minor differences exist in whitespace and cell sizing between our standard cell benchmarks and those used in [11].

B. Mixed-size Problems

Our second set of experiments focuses on mixed-size placement problems. We use the `ICCAD04 IBM-MSwPins` Mixed-size Placement Benchmarks [14]. Results are presented in Table II. We observe that `LSD`, without I/O reassignment, achieves results that are 10% better than `Capo` and 8% better than `FengShui` on mixed-size designs. Run-times are also favorable, with `LSD` being 37% faster than `Capo`, and only 26% slower than `FengShui`. I/O assignment offers an improvement of $\approx 5\%$ in HPWL at the cost of only $\approx 14\%$ in run-time, bringing the quality of our placer to 12% and 15% better than `FengShui` and `Capo`, respectively.

VI. Conclusions

We have described the implementation of a novel line search technique for generic placement. Our method achieves competitive results using a surprisingly simple approach which "corrects" weighting errors in the spreading forces by testing and evaluating a set of force multipliers. This technique alleviates the need for the *ad hoc* scaling formerly used in [9]. Consequently, our method achieves improved quality with good run-times compared to other academic tools.

We feel that there are still opportunities to improve the quality and performance of our flow. Specifically, we are working at reducing our tool's memory footprint to improve the handling of large designs. Moreover, we are investigating better ways of handling global and detailed placement in designs with fixed obstacles.

[2]This does not imply that we pack tightly to the left—we attempt to preserve the global placement as much as possible and do not perform any compaction of whitespace.

[3]We omit direct comparisons with several tools due to space and unavailability of executables. Comparisions of these tools to `Capo` and `Fengshui` in the literature allow for indirect comparisions.

[4]The modifications to the benchmarks are available for download [19].

TABLE I. Results on modified ISPD02 circuits (macros reduced to row height).

Circuit	FastPlace 1.0		Dragon 3.01		Capo 9.1		LSD		LSD with I/O Assgn.		WL Ratio (LSD vs.)			
	WL	CPU	WL	CPU	WL	CPU	WL	CPU	WL	CPU	FP	Dragon	Capo	LSD I/O
ibm01	2.01	0.1	1.68	15.9	1.68	1.9	1.76	2.1	1.74	2.3	0.88	1.05	1.05	1.01
ibm02	3.83	0.2	3.64	14.2	3.71	4.1	3.76	4.9	3.63	5.3	0.98	1.03	1.01	1.04
ibm03	5.76 *	> 300	5.45	13.2	5.69	5.4	5.29	4.6	5.64	4.7	n/a	0.97	0.93	0.94
ibm04	6.22 *	> 300	6.19	25.7	6.59	6.8	6.22	5.3	6.12	5.9	n/a	1.00	0.94	1.02
ibm05	11.37	0.4	9.83	40.4	9.95	8.2	9.95	7.5	9.55	8.3	0.88	1.01	1.00	1.04
ibm06	5.73	0.3	5.07	29.7	5.71	7.2	5.23	5.8	5.19	6.3	0.91	1.03	0.92	1.01
ibm07	13.36	0.6	8.76	38.2	9.39	11.8	8.88	8.6	8.78	9.3	0.66	1.01	0.95	1.01
ibm08	10.14	0.8	8.76	92.7	9.54	12.9	8.98	10.9	8.99	12.2	0.89	1.03	0.94	1.00
ibm09	11.77 *	> 300	12.69	83.2	12.61	14.1	11.44	9.6	10.82	10.6	n/a	0.90	0.91	1.06
ibm10	18.94	1.1	17.34	73.8	18.45	20.8	18.01	13.6	17.77	14.5	0.95	1.04	0.98	1.01
ibm11	16.53 *	> 300	16.42	52.1	18.08	21.3	16.68	13.0	16.16	14.1	n/a	1.02	0.92	1.03
ibm12	26.51	1.4	22.50	80.7	24.32	26.8	23.00	14.7	22.62	16.6	0.87	1.02	0.95	1.02
ibm13	20.26 *	> 300	20.93	67.8	24.07	26.6	19.62	16.1	19.85	18.3	n/a	0.94	0.82	0.99
ibm14	38.15	3.4	32.81	165.2	34.02	50.9	35.55	29.9	34.01	34.8	0.93	1.08	1.04	1.05
ibm15	45.65 *	> 300	47.17	217.4	49.91	64.4	45.85	36.6	42.65	42.5	n/a	0.97	0.92	1.08
ibm16	45.89	5.0	44.97	250.8	47.00	69.2	47.30	46.7	45.06	47.7	1.03	1.05	1.01	1.05
ibm17	122.06	7.1	65.14	532.2	67.76	81.5	64.89	52.0	63.04	57.9	0.53	1.00	0.96	1.03
ibm18	58.49	7.8	41.36	475.8	43.55	73.4	44.46	58.6	45.81	66.8	0.76	1.07	1.02	0.97
										Avg.	**0.86**	**1.01**	**0.96**	**1.02**

TABLE II. Results on the ICCAD04 mixed-size circuits.

Circuit	FengShui 5.0		Capo 9.1		LSD		LSD with I/O Assgn.		WL Ratio (LSD vs.)		
	WL	CPU	WL	CPU	WL	CPU	WL	CPU	FS	Capo	LSD I/O
ibm01	2.46	1.8	2.54	2.7	2.42	2.2	1.97	2.4	0.98	0.95	1.23
ibm02	5.75	3.1	5.10	5.0	5.11	4.3	4.58	4.7	0.89	1.00	1.12
ibm03	8.48	3.4	9.37	13.2	7.08	5.1	6.46	5.1	0.83	0.76	1.10
ibm04	8.71	3.9	9.41	9.8	7.69	4.8	7.42	5.6	0.88	0.82	1.04
ibm06	6.98	5.4	7.35	9.8	6.20	5.9	6.55	6.7	0.89	0.84	0.95
ibm07	11.78	7.2	12.70	14.7	10.57	9.2	10.12	9.0	0.90	0.83	1.04
ibm08	14.16	8.7	13.65	16.3	13.30	10.9	12.08	12.3	0.94	0.97	1.10
ibm09	14.95	8.4	15.88	16.7	13.30	10.8	13.03	12.8	0.89	0.84	1.02
ibm10	35.89	12.9	32.88	29.9	30.70	14.0	27.98	20.8	0.86	0.93	1.10
ibm11	20.37	12.4	22.51	24.1	18.41	14.4	17.57	16.6	0.90	0.82	1.05
ibm12	40.33	13.5	39.01	33.1	36.46	22.3	35.74	24.3	0.90	0.93	1.02
ibm13	25.41	14.6	27.83	30.1	23.60	17.8	21.91	20.0	0.93	0.85	1.08
ibm14	38.34	28.8	40.60	55.7	37.84	33.1	36.70	38.4	0.99	0.93	1.03
ibm15	50.71	35.0	55.89	67.6	47.69	37.4	49.07	42.3	0.94	0.85	0.97
ibm16	59.73	38.7	65.38	76.0	61.27	42.3	56.97	50.4	1.03	0.94	1.08
ibm17	71.08	43.1	73.29	84.4	69.45	54.2	70.48	65.3	0.98	0.95	0.99
ibm18	45.34	45.2	47.77	65.2	44.88	66.2	44.76	72.1	0.99	0.94	1.00
								Avg.	**0.92**	**0.89**	**1.05**

REFERENCES

[1] T. Taghavi, X. Yang, and B.-K. Choi, "Dragon2005: large-scale mixed-size placement tool," in *Proc. ISPD*, 2005, pp. 245–247.

[2] A. R. Agnihotri, S. Ono, and P. H. Madden, "Recursive bisection placement: feng shui 5.0 implementation details," in *Proc. ISPD*, 2005, pp. 230–232.

[3] J. A. Roy *et al.*, "Capo: robust and scalable open-source min-cut floorplacer," in *Proc. ISPD*, 2005, pp. 224–226.

[4] A. B. Kahng, S. Reda, and Q. Wang, "Aplace: a general analytic placement framework," in *Proc. ISPD*, 2005, pp. 233–235.

[5] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proc. ISPD*, 2005, pp. 185–192.

[6] K. Chaudhary and S. K. Nag, "Method for analytical placement of cells using density surface representations," United States Patent 6,415,425, July 2002.

[7] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. DAC*, 1998, pp. 269–274.

[8] K. Vorwerk, A. Kennings, and A. Vannelli, "Engineering details of a stable analytic placer," in *Proc. ICCAD*, 2004, 573–580.

[9] K. Vorwerk and A. Kennings, "An improved multi-level framework for force-directed placement," in *Proc. DATE*, 2005, pp. 902–907.

[10] B. Hu and M. Marek-Sadowska, "FAR: Fixed-points addition & relaxation based placement," in *Proc. ISPD*, 2002, pp. 161–166.

[11] N. Viswanathan, M. Pan, and C. C.-N. Chu, "Fastplace: an analytical placer for mixed-mode designs," in *Proc. ISPD*, 2005, pp. 221–223.

[12] A. Kennings and I. Markov, "Analytical minimization of half-perimeter wirelength," in *Proc. ASPDAC*, 2000, pp. 179–184.

[13] D. Pham *et al.*, "The design and implementation of a first-generation CELL processor," in *Proc. International Solid-State Circuits Conference*, 2005.

[14] S. N. Adya *et al.*, "Unification of partitioning, placement and floorplanning," in *Proc. ICCAD*, 2004, pp. 550–557.

[15] S. Adya and I. Markov, "ISPD02 mixed-size placement benchmarks," http://vlsicad.eecs.umich.edu/BK/ISPD02bench, Current July 2004.

[16] B. V. Cherkassky and A. V. Goldberg, "On implementing the push-relabel method for the maximum flow problem," *Algorithmica*, vol. 19, no. 4, pp. 390–410, 1997.

[17] S.-W. Hur and J. Lillis, "Mongrel: Hybrid techniques for standard cell placement," in *Proc. ICCAD*, 2000, pp. 165–170.

[18] X. Y. M. Wang and M. Sarrafzadeh, "Dragon2000: Standard-cell placement tool for large industry circuits," in *Proc. ICCAD*, 2000, pp. 260–263.

[19] http://gibbon.uwaterloo.ca.