Engineering Details of a Stable Force-Directed Placer*

Kristofer Vorwerk
Dept. of E&CE, University of Waterloo
Waterloo, Ontario, Canada
kpvorwer@cheetah.vlsi.uwaterloo.ca

Andrew Kennings
Dept. of E&CE, University of Waterloo
Waterloo, Ontario, Canada
akenning@cheetah.vlsi.uwaterloo.ca

Anthony Vannelli
Dept. of E&CE, University of Waterloo
Waterloo, Ontario, Canada
vannelli@cheetah.vlsi.uwaterloo.ca

Abstract

Analytic placement methods that simultaneously minimize wire length and spread cells are receiving renewed attention from both academia and industry. In this paper, we describe the implementation details of a force-directed placer, FDP. *Specifically, we provide (1) a description of efficient force computation for spreading cells, (2) an illustration of numerical instability in these methods and a means by which these instabilities are avoided, (3) spread metrics for measuring cell distribution throughout the placement region and (4) a complementary technique which aids in directly minimizing* HPWL. *We present results comparing our analytic placer to other academic tools for both standard cell and mixed-size designs. Compared to* Kraftwerk *and* Capo 8.7, *our tool produces results with an average improvement of 9% and 3%, respectively.*

INTRODUCTION

As problem instances have increased in size and complexity, placement has, more and more, become the bottleneck in deep sub-micron designs. Typically, placement seeks to minimize wire length subject to the constraints that cells must be placed into prescribed locations without overlap. There are several approaches to this problem. Timberwolf [1] and VPR [2] use simulated annealing to improve an existing placement. Top-down partitioning-based methods [3, 4, 5] recursively divide the placement region and the circuit netlist into smaller pieces using either bi-section or quadri-section and a min-cut (or other) objective function. Analytic-based placers use quadratic or linear optimization [6, 7, 8, 9, 10] to perform cell placement. While analytic placers usually minimize an indirect measure of wire length, they can do so very efficiently, and are therefore capable of handling large problems.

Several placement methods are often combined to improve the performance or quality of the placement. For example, GORDIAN [7], GORDIAN-L [9] and BonnPlace [6] combine analytic placement and top-down recursive partitioning. In such frameworks, the analytic placement provides useful information regarding the relative positions of cells, whereas the partitioning enforces the requirement that cells must not overlap with each other.

Alternatively, an analytic placer can employ *forces* such that fairly non-overlapping placements are obtained without partitioning. In [10], Eisenmann and Johannes use forces to push cells from dense to less-dense regions. In [8, 11], fixed, or dummy, points are introduced as a means of pulling cells from dense to less-dense regions.

Force-based methods are of great interest for several reasons. Mixed-size problems—that is, circuits with a combination of standard cells and macro blocks—are handled seamlessly by such techniques. Furthermore, these methods provide continuous cell locations and therefore appear very amenable to timing- and congestion-driven placement, physical re-synthesis, and ECO. Yet, in our experience, developing a force-directed placer based on [10] is not trivial—while the work of Eisenmann and Johannes offers a good starting point, we have discovered the need for numerous additional techniques to stabilize the method and to improve the quality of results. To our knowledge, none of the recent literature on force-directed placement (c.f. [12, 13, 14, 10]) has addressed the stability and quality issues that we broach.

In this light, the purpose of our paper is two-fold: we describe a practical implementation of an Eisenmann-based placer, and expound upon the strategies that we have developed for improving quality and performance. We describe a means of efficiently computing spreading forces, metrics which allow for stopping placement based on cell distribution, a means of preventing the destabilization of placements due to numerical instability, and a direct HPWL-minimizing algorithm to improve overall quality. Experimental results confirm that our enhancements improve upon Kraftwerk (the commercial implementation of [10]) by 9% on average for mixed-size problems, bringing the quality of results to a level on-par with other state-of-the-art approaches.

The rest of the paper is organized as follows. We begin by presenting an overview of force-directed placement. Then, we discuss the implementation details of our placer and our additional methods for enhancing quality. Experimental results are then presented, followed by concluding remarks.

BACKGROUND

Quadratic Optimization

A circuit is 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 $x = (x_1, x_2, \cdots, x_n)$ and $y = (y_1, y_2, \cdots, y_n)$.

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

0-7803-8702-3/04/$20.00 ©2004 IEEE.

Circuit hypergraphs are typically transformed into graphs in which each hyperedge is represented by a set of equally-weighted edges. The *star model* adds a new center vertex and represents the original net by edges connecting the center to the previously existing vertices. The *clique model* connects all pairs of vertices incident to the original hyperedge by equally-weighted edges. Clique models of large hyperedges become prohibitively expensive due to the quadratic edge count. Consequently, large edges are either dropped completely, or a combination of clique and star models are employed in which cliques are used to model small hyperedges and stars are used to model large hyperedges.

The overall method for minimizing wire length is accomplished by solving the quadratic optimization problem (x-direction only) given 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. Assuming that some cells are fixed, the Hessian is a symmetric, positive-definite matrix.[1] 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. The optimization problem is strictly convex and has a unique minimizer given by the solution of a single, positive-definite system of linear equations, $\mathbf{Q}_x\mathbf{x} + \mathbf{c}_x = 0$. In this formulation, cell overlap is ignored, and the vector $\mathbf{x}$ provides only relative cell positions (i.e., the resultant placement would be infeasible and would have to be legalized).

Briefly, we note that numerous studies have shown that quadratic optimization tends to produce placements of inferior quality. Better results may be achieved using iterative quadratic optimizations (at the expense of potentially significant increases in run-time) with weighting given by

$$\min_{x^\nu} \sum_{ij} \frac{a_{ij}}{|\mathbf{x}_i^{\nu-1} - \mathbf{x}_j^{\nu-1}|}(x_i^\nu - x_j^\nu)^2 \qquad (2)$$

where $\mathbf{x}^{\nu-1}$ and $\mathbf{x}^\nu$ denote the vectors of vertex positions at iterations $\nu-1$ and ν, respectively. Thus, a quadratic objective function is used for efficient optimization, but re-weighting is used to approximate linear rather than quadratic distances as in GORDIAN-L [9].

Cell Spreading

Eisenmann and Johannes [10] proposed to directly modify the system of equations by including an additional vector of *forces* at 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

$$\mathbf{Q}_x\mathbf{x}_i + \mathbf{c}_x + \sum_{l=1}^{i-1} \alpha_l\mathbf{f}_l + \alpha_i\mathbf{f}_i = 0 \qquad (3)$$

where $\mathbf{f}_i$ represents the spreading forces computed at iteration i and α_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 force field. We note that spreading forces are *accumulated* over iterations to avoid placements from "collapsing" back onto themselves.

At each iteration i of placement, the vector of spreading forces $\mathbf{f}_i$ is calculated using current cell locations and Poisson's equation given by

$$f(x,y) = k\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} D(x',y')\frac{\vec{r}(x,y) - \vec{r}(x',y')}{|\vec{r}(x,y) - \vec{r}(x',y')|^2}dx'dy' \qquad (4)$$

where $f(x,y)$ is the force on a cell at location (x,y), $\vec{r}(x,y)$ is the vector representation of the point (x,y) and $D(x,y)$ is the density at point (x,y). $D(x,y)$ represents the ratio of total cell occupancy to allowable capacity at point (x,y); consequently, $D(x,y)$ measures the over-utilization of any point within the placement region. In practice, the force computation is accomplished using discrete bins, with the continuous integration in (4) being replaced with discrete summations. We elaborate on this implementation in the following section.

IMPLEMENTATION

Core Framework

In developing FDP, various open-source matrix packages were investigated. Based on experimental results, we selected the Boost uBLAS library [15] for its efficient matrix-vector operations [16]. We then developed an ILU(0) preconditioned BiCGSTAB solver, and the Reverse Cuthill-Mckee heuristic [17] to reorder the Hessians. When constructing the Hessians, we use a clique model for nets smaller than 20 and a star model for all larger nets. Using a clique model for nets up to this size seems to produce better quality placements than when using the sizing advocated by [18].

Force Computation

While spreading forces have been mentioned frequently in the literature, the methods by which these forces are computed have not been explained in detail. We believe that a full description of how to compute these forces is beneficial. We compute spreading forces using a Barnes-Hut quad-tree for n-body force calculation [19]. This method was chosen for its good overall performance and its relatively high-quality force calculation. The accuracy of the force calculation was found to be especially important in the early stages of force-directed placement, and it is for this reason that we advocate this approach. Moreover, since the quad-tree already tracks cell locations and area, it can be employed in other algorithms throughout the placement.

[1] In any real circuit, the I/O pads are fixed and this condition is satisfied.

In our implementation, a quad-tree with at most 9 levels is constructed for the placement region. 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 occupies (overlaps) in each level of the tree.

Large cells may span several quad-tree bins and may receive contributions to their forces from each of the bins that they occupy. As a result, the largest cells tend to receive the largest forces. For placements with a wide distribution of cell dimensions the relative magnitudes of forces between cells can vary considerably—this can cause the largest cells to spread too quickly compared to the smaller cells, harming wire length. We have found that dividing the magnitude of the forces on each cell by the square root of the number of bins that it occupies (at each level in the quad-tree) leads to a more even distribution of force magnitudes (as shown in Figure 1), and empirically results in better placements.

Spread Metrics and Stopping Criteria

It is important to be able to measure the progress of the placer in minimizing cell overlap. In the absence of an adequate *spread metric*, it is difficult to assess how much better (or worse) one placement iteration is than another. Moreover, it is difficult to judge whether or not the spreading forces are properly weighted, or to determine when to terminate the placement and continue with legalization. To this end, two spreading metrics were developed and are presented here. Each metric assesses the spreading of cells in a slightly different manner, and each metric has its own strengths and weaknesses.

The first metric is based on the violation measure $D(i,j)$ in the Barnes-Hut quad-tree. The metric is calculated by descending the tree and adding the factor by which the occupancy of a geometric region exceeds its allowable capacity scaled by the square of the quad-tree level; that is, we compute:

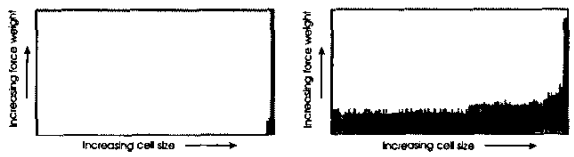
$$\sum_{l=1}^{L} \frac{1}{l^2} \sum_{i,j:\mathrm{occ}(i,j)>\mathrm{cap}(i,j)}^{2^l} \frac{\mathrm{occupancy}(i,j)}{\mathrm{capacity}(i,j)} \qquad (5)$$

for bins (i,j) at each level l of the tree. This weighting places more emphasis on capacity violations in the top levels of the quad-tree. Given that the worst possible value would occur when all cell area is focused on the smallest bin at each level of the quad-tree, we can normalize this spread metric to within the interval $[0,1]$.

We employ a second spread metric based on Klee's measure problem [20, 21]. The $O(n \log n)$ segment tree technique by Bentley [21] was implemented to measure the area of the union of the modules in the placement region. The Klee's measure is then divided by the total cell area to give a normalized value in the interval $[0,1]$, representing the percentage overlap remaining in the placement. We have found that the average of the two spread metrics offers a reasonable impression of the amount of spreading throughout the entire placement flow.

Unlike the approach used by [10], in which global placement is stopped once there are "no empty squares within the placement area which is larger than four times the average area of a cell" [10], the spread metrics described here can be used to stop placement in designs with large amounts of white space. Our experiments have shown that the placement can finish and proceed to legalization with 30%–35% overlap remaining. This stopping point offers a good tradeoff between placer performance and spreading of cells. Our legalization strategy is usually capable of retaining quality with this much overlap.

As noted in [22], the force-directed approach can require *many* iterations to completely purge overlap from a circuit. Consequently, it is computationally more efficient to use an intelligent legalization scheme to remove the final overlap.



Before scaling the force magnitude

After scaling the force magnitude

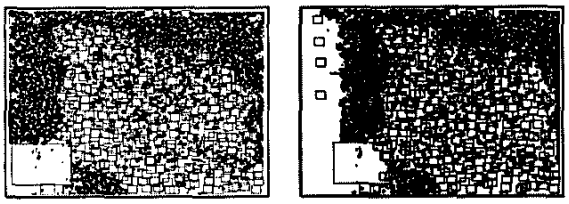
Figure 1: Distribution of force magnitudes before and after scaling by the square root of the number of quad-tree bins. Note that prior to scaling, forces on large cells are significant whereas the forces on small cells are almost negligible (barely visible in the diagram). After scaling, forces are more uniform, although larger cells still receive justifiably larger forces.

Stability

During the development of FDP, it was discovered that the incremental addition of a small amount of spreading forces from one iteration to another could result in a large "destabilization" of the placement. In these cases, initially-spread cells could "collapse" into the edge of the placement region. This phenomenon is illustrated in Figure 2.

Initially, it was suspected that the force weights were too large, and that they were causing cells to move too much in a given iteration. As a result, a "save/restore" mechanism was implemented to discard a placement if the resulting spread metrics indicated a significant increase in the amount of overlap (and force weights were lowered). Unfortunately, this technique did not eliminate the destabilization.

It was soon realized that the Hessians for most problems were ill-conditioned, and that no changes to the force weighting would improve stability. An examination of the matrix equations followed. We begin by noting that the previous



Placement before destabilization

Placement after destabilization (one iteration later)

Figure 2: Illustration of destabilization during placement. Destabilization is an undesirable side-effect of numerical ill-conditioning in the Hessian matrices.

iteration is given by:

$$\mathbf{Q}_x\mathbf{x} + \mathbf{c}_x + \mathbf{f}_x = 0 \tag{6}$$

whereas the current iteration is given by:

$$\mathbf{Q}_x\mathbf{x}' + \mathbf{c}_x + (\mathbf{f}_x + \Delta\mathbf{f}_x) = 0. \tag{7}$$

Subtracting (6) from (7) yields

$$\mathbf{Q}_x(\Delta\mathbf{x}) + \Delta\mathbf{f}_x = 0. \tag{8}$$

In physical terms, (8) indicates that the increment in position of modules is a function of the incremental force and the springs between the cells. It shows that if $\mathbf{Q}_x$ is poorly conditioned, then scaling $\Delta\mathbf{f}_x$ linearly would not help since the conditioning applies to the Hessian being solved, chiefly $\|\mathbf{Q}_x^{-1}\Delta\mathbf{f}_x\|_1 = \|\mathbf{Q}_x^{-1}\|_1\|\Delta\mathbf{f}_x\|_1$.

To improve conditioning, the placement could be stabilized using fixed points as in [11] and the cumulative forces reset to zero. However, a less computationally-intensive approach was devised. Instead of changing the cumulative forces, a "virtual" fixed point (with no height or width) is placed at the position of the movable modules in the netlist. The positions of the virtual fixed points are then updated in each subsequent iteration—doing so only affects the diagonals of the Hessians, while improving conditioning. With the addition of these fixed points (connected by potentially weighted nets), Equation (8) becomes

$$\mathbf{Q}_x(\mathbf{x}' - \mathbf{x}) + \mathbf{w}\mathbf{I}(\mathbf{x}' - \mathbf{x}) + \Delta\mathbf{f}_x = 0 \tag{9}$$

which yields

$$(\mathbf{Q}_x + \mathbf{w}\mathbf{I})(\Delta\mathbf{x}) + \Delta\mathbf{f}_x = 0 \tag{10}$$

$$\Delta\mathbf{x} = -(\mathbf{Q}_x + \mathbf{w}\mathbf{I})^{-1}\Delta\mathbf{f}_x. \tag{11}$$

Since the diagonals will be larger, (11) has the effect of also being faster for a conjugate gradient method to solve. Moreover, if one matrix dominates the other ($\mathbf{w}\mathbf{I} \gg \mathbf{Q}_x$), the system becomes, in effect:

$$\Delta\mathbf{x} \approx (\mathbf{w}\mathbf{I}^{-1})\Delta\mathbf{f}_x \approx \mathbf{w}^{-1}\Delta\mathbf{f}_x. \tag{12}$$

Equation (12) reveals that, if a sufficient number of virtual fixed cells were to be added at the locations of the current movable cells, then the step that will be taken will be in the direction of the incremental force, and that only some fraction (controlled by $\mathbf{w}$) of that distance will be traveled. Physically, a weighted step in the direction of the force can be interpreted as adding *friction* to the placement.

Experiments conducted with "friction" have indicated that only a small percentage of the rows in the Hessian—usually, only around 5% of the least diagonally-dominant rows—need to have virtual fixed cells attached in order to improve conditioning and stability. After friction is applied and the previous iteration's cell positions are restored, subsequent placement iterations usually do not destabilize and the quality of results remains unaffected.

IMPROVING QUALITY

We note that standard Eisenmann-based placement does a good job of spreading cells across the placement area. In practice, however, we found that a number of modifications and additions were necessary to achieve results on-par with what may be considered state-of-the-art.

BoxPlace Improvement

Foremost among the methods that we have implemented to improve quality is the inclusion of a technique called "BoxPlace" [23]. BoxPlace moves each cell to the median location of its connected nets, thereby reducing HPWL directly.[2] The pseudocode for the BoxPlace routine is shown in Figure 3.[3] In BoxPlace, the range of (x,y) values which minimize HPWL for the nets connected to cell i is calculated as follows. First, the x and y minimums and maximums of the bounding boxes for all nets (excluding the points contributed by cell i) are inserted into two vectors—one for each direction. The vectors are then sorted by increasing value. The median ($\lfloor n/2 \rfloor$) and median plus one$^{\text{th}}$ ($\lfloor n/2 \rfloor + 1$) locations yield the "box" (or range) of positions into which cell i can be moved to improve HPWL. By using a grid-based structure, cells can be tracked to ensure that they are inserted into a relatively non-overlapping area within their target box.

In our embodiment, calls to BoxPlace are performed after every 3% improvement (reduction) in cell overlap. Since BoxPlace can *reintroduce* overlap, we re-evaluate the spreading after each pass of BoxPlace—if too much overlap was introduced, we restore the previous cell locations and stop.

We also use BoxPlace to compute "minimizing forces". These forces are simply vectors which point to the locations to which cells should move to reduce HPWL. These forces are scaled appropriately, and then combined linearly (in a ratio of 40%/60%) with the spreading forces. This vector addition reorients the angles of the spreading forces to point in a direction that favors spreading, but additionally minimizes wire length as shown in Figure 4.

[2]The BoxPlace concept can also be extended to move cells based on timing constraints, although this modifi cation is not discussed here.

[3]Our implementation caches bounding box information to improve performance. The details of this caching code have been omitted for brevity.

```
1   Procedure: BOXPLACE
2   Input: A netlist, N
3   begin
4     E ← all eligible, movable cells in N;
5     for each pass ∈ 0, 1, ..., MAX_PASSES do
6       lastPosn ← position of all cells;
7       Randomly permute E;
8       for each n_i ∈ E do
9         Find the range ('box') of (x,y) values
10          that minimizes HPWL for n_i;
11        Move n_i to a relatively non-overlapping
12          area somewhere within the 'box';
13      od
14      if too much overlap was reintroduced then
15        Restore the cell positions from lastPosn;
16        break ;
17      fi
18    od
19  end
```

Figure 3: Pseudocode for the BoxPlace algorithm.

We have found that BoxPlace, when called individually and when added (vectorally) to the spreading forces, undoes much of the damage to HPWL caused by the spreading forces alone. BoxPlace helps our placer to achieve a more linearized placement without degrading performance as much as a GORDIAN-L-type of linearization. We have found that BoxPlace forces are especially important in improving the quality of our placement above and beyond that of Kraftwerk.

In addition, BoxPlace aids cell spreading in a subtle fashion. One of the problems with achieving a high-quality placement lies in the fact that forces do not allow cells to "flip" sides. That is, once a cell is located to the right of another cell, the spreading forces will not allow the two to cross paths, even if doing so would improve HPWL. This problem can be particularly troublesome given that relative cell ordering is often established early on in placement after the initial quadratic problem is solved. This is when there exists the greatest amount of "uncertainty" in the placement and spreading forces are most likely to do the greatest amount of damage to HPWL. BoxPlace, on the other hand, performs this "flipping" and situates cells in more favorable positions to reduce wire length.

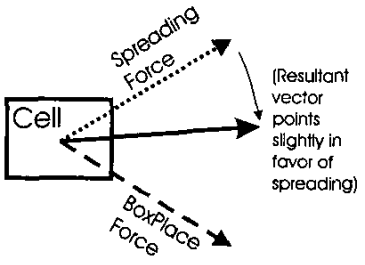


Figure 4: Illustration of the addition of spreading and BoxPlace forces, and the resultant vector.

Dynamic Force Weighting

The weights applied to the spreading forces impact the speed at which a circuit spreads, and can affect the quality of the overall wire length [10, 22]. We have found that *weighting forces via a constant, as advocated by [10], leads to poor placements.* In our implementation, the force weight is adapted *dynamically* in each iteration to achieve both fast spreading and good quality.

In FDP, the force vectors for each iteration are normalized with respect to the largest force vector. Then, the force weighting schedule weights the elements of $\mathbf{f}_x$ and $\mathbf{f}_y$ by a scalar value k. The value of k in each iteration is adjusted based on the following experimental observations.

- Initially, the force weight k should be as small as possible. While good spreading can still be achieved if large force weights were used in the first 10 to 50 iterations of placement, wire length may be significantly worsened.
- Once the relative ordering of cells has been well established, and the circuit begins to spread consistently, k can be increased in each iteration to encourage faster spreading. k should be reduced if the spread metrics detect too much reduction in the amount of overlap (and vice-versa for too little improvement in the spreading). Near the end of placement, force weights must be quite large to overcome the quadratic "springs".

In other words, force weights are adapted continuously using, in effect, a three-step state machine in which the spread metrics are used to help in dynamically weighting forces—if there is too much spreading, force weights are lowered, and if there is too little, force weights are raised. The spread metrics can also help to determine if the placement is "oscillating" (no longer spreading), in which case our placer can stop and proceed to legalization.

It should be further noted that, to achieve consistent force weights across a variety of circuits, the placement region (and all cells) are scaled by the average cell height and width. We have found that, in general, this scaling does a good job of normalizing the force weighting such that different sizes of placement regions are affected in roughly the same way by the same force weight.

Clustering

Spreading forces can harm wire length in some cases by pushing the largest cells to the edges of the placement region too quickly. This is due to the fact that large cells tend to possess considerable overlap in the early stages of placement, and may receive large "pushing" forces early on. The more that cells are similarly-sized, however, the less of a negative impact that spreading forces impart on overall wire length.

We use Hybrid First Choice clustering [24] to cluster netlists to at most 70% of their original size, prior to placement. Clustering tends to "smooth" the differences in cell heights and widths, and ensures that nodes with high affinities stay

close together throughout the placement (so that they are not unduly pushed apart by the spreading forces).

Legalization

Placements produced by FDP are not "valid" in that cells are not assigned to rows and some residual overlap may still be present. Our legalization strategy is presently quite simple. We initially snap cells to their closest rows in order to minimize total cell movement. Greedy juggling of cells is performed between rows in order to meet width restrictions imposed by the fixed die. Finally, we apply greedy (same-size) cell swaps and single-row branch and bound on groups containing six or fewer cells.

In the case of circuits containing macro cells, we first test the placement using a sequence pair analysis along the lines of [25]. In our experience, macro cells generally fit and no additional work is required. (Of course, we could choose to apply operations on the sequence pair as in [25] if the macro cells do not fit inside the fixed die.) Using the results of the sequence pair analysis, macro cells are shifted to align with rows and remove overlap in the y-direction, and shifted to the left and right to remove overlap in the x-direction. Currently, no attempt is made to optimize whitespace in the final placement. Examples of two placements before and after legalization are shown in Figure 5.

EXPERIMENTAL RESULTS

Standard Cell Benchmarks

In the first set of experiments, we were interested in investigating the stability and capability of our framework to effectively place standard cell circuits. We modified the ISPD02 benchmarks in [26, 27, 28], shrinking macro cells to standard dimensions, and adjusting problems to maintain an aspect ratio of 1.0 with 5% whitespace.[4]

These tests were then applied to Kraftwerk [10], the commercial version of the force-directed approach upon which our placer is based.[5] We found that Kraftwerk produces overlapping placements with usually between 30%–35% overlap using Klee's measure, so FDP was set to terminate appropriately.

The ratios comparing the HPWL of the two placements are shown in Table 1. In general, FDP achieved results that were 7% better in pre-legal wire length, thereby validating the efficacy of the methods described in this paper.

We also compared our results to Capo 8.7 [4] and Dragon 3.01 [29], all of which were executed on a Pentium 4, 2.8 GHz machine running Fedora Linux. Dragon was run in fixed-die mode; note that some of the results for Dragon are not available because it crashed on a handful of designs. Run times for each placement are expressed in minutes, and the wire lengths are for the half-perimeter (divided by 10^6). The results reported for FDP are legalized, and the time represents the cumulative time required for placement *and* legalization. Results featuring these standard cell benchmarks, as well as a set of unit-sized benchmarks (with the same aspect ratio and whitespace), are shown in Table 2. In the standard cell benchmarks, FDP achieved results that were 2% better than Capo 8.7, with run times that were within a factor of 4 (for ibm10 and above). In the unit-sized benchmarks, FDP proved to be 3% better than Capo 8.7 on average.[6]

FDP tends to excel on unit-sized problems for two reasons. First, there is a more even distribution of spreading force magnitudes, so it is unlikely that some cells will be pushed "unfairly" to the wrong side of the placement region. Second, the heuristic improvement strategies employed in our legalizer are afforded more alternatives for swapping when cell dimensions are the same size.

Table 1: Standard cell benchmarks with an aspect ratio of 1.0 and 5% whitespace, comparing Kraftwerk to FDP at approximately the same amount of pre-legal overlap.

Circuit	Kraftwerk		FDP	Ratio
	Pre-Legal HPWL	Overlap (%)	Pre-Legal HPWL	FDP/Kraft.
ibm01	1.86	28.65	1.75	0.94
ibm02	4.51	30.28	4.04	0.90
ibm03	5.91	33.19	5.70	0.96
ibm04	7.16	30.37	6.38	0.89
ibm05	12.34	30.19	10.87	0.88
ibm06	5.71	32.29	5.61	0.98
ibm07	10.43	30.73	9.77	0.94
ibm08	10.53	31.16	9.77	0.93
ibm09	12.17	34.69	11.34	0.93
ibm10	19.92	35.13	19.19	0.96
ibm11	17.68	35.82	17.25	0.98
ibm12	26.60	33.54	24.46	0.92
ibm13	22.20	37.85	20.70	0.93
ibm14	38.65	34.40	35.48	0.92
ibm15	50.66	36.12	47.68	0.94
ibm16	52.72	36.80	47.97	0.91
ibm17	78.22	34.24	67.07	0.86
			Average	0.93

Mixed-Size Benchmarks

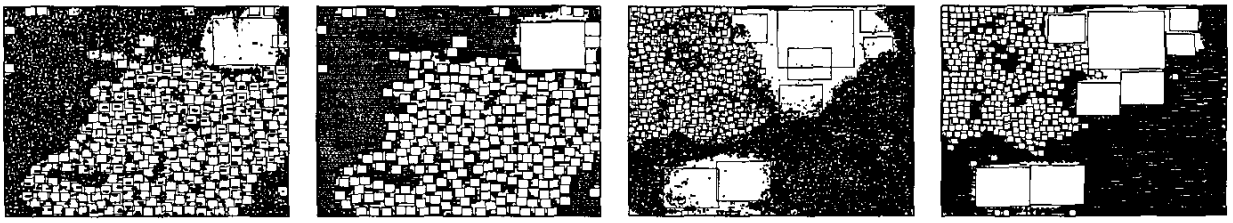
In the second set of experiments, we investigated FDP's handling of mixed-size problems. We compare to results recently published in [26], which include values from Capo, Kraftwerk, and mPG [30]. As our testing platform differs from those used in [26], we cannot compare the relative performance of each approach, but we can compare the quality of the final placements. Performance values are provided as a "reality-check".

We do not explicitly compare to the recent results of [5]. Our tool distributes whitespace throughout the placement area, whereas [5] appears to pack to the left. While the wire lengths of [5] are excellent, the differences in whitespace

[4]These modifi ed standard cell benchmarks are available on our website, at http://gibbon.uwaterloo.ca.

[5]Due to the method used to acquire the Kraftwerk results, a direct performance comparison with our placer was not possible.

[6]We note that, to facilitate indirect comparison with other tools, Capo 8.5 produces results that are, on average, 2% worse than Capo 8.7, and thus between 4 and 5% worse than FDP on these benchmarks. Space constraints, however, did not permit a full comparison with Capo 8.5 in this paper.



ibm01 before legalization ibm01 after legalization ibm04 before legalization ibm04 after legalization

Figure 5: Illustration of ibm01 and ibm04, before and after legalization.

Table 2: Standard cell (and unit-sized) benchmarks with an aspect ratio of 1.0 and 5% whitespace. Run times are observed on a 2.8 GHz Pentium, and reported in minutes. The reported wire length is HPWL.

Circuit	Standard Cell Height (No Macros)								Unit Size (No Macros)							
	Capo		Dragon		FDP		vs. Capo		Capo		Dragon		FDP		vs. Capo	
	WL	CPU	WL	CPU	WL	CPU	WL	CPU	WL	CPU	WL	CPU	WL	CPU	WL	CPU
ibm01	1.77	1	1.70	14	1.68	7	0.95	7.0	2.17	1	2.15	12	2.14	5	0.99	5.0
ibm02	3.87	2	n/a	n/a	3.78	11	0.98	5.5	5.22	2	4.97	19	5.21	12	1.00	6.0
ibm03	5.38	2	5.73	12	5.56	11	1.03	5.5	6.87	2	6.54	21	6.52	9	0.95	4.5
ibm04	6.49	3	6.39	25	6.21	15	0.96	5.0	7.88	3	7.42	37	7.84	11	0.99	3.7
ibm05	10.12	3	9.97	37	10.39	18	1.03	6.0	13.04	3	12.39	53	13.78	14	1.06	4.7
ibm06	5.59	3	5.71	26	5.38	14	0.96	4.7	8.64	3	7.49	45	7.54	16	0.87	5.3
ibm07	9.59	5	9.19	35	9.34	23	0.97	4.6	12.86	4	11.98	33	12.23	23	0.95	5.8
ibm08	9.98	5	9.10	79	9.42	24	0.94	4.8	13.64	5	12.92	78	13.37	26	0.99	5.2
ibm09	11.51	6	13.03	77	11.23	26	0.98	4.3	13.50	6	13.17	63	12.91	22	0.96	3.7
ibm10	18.92	8	n/a	n/a	18.36	29	0.97	3.6	22.56	7	21.63	85	21.74	56	0.96	8.0
ibm11	16.69	8	n/a	n/a	17.04	34	1.02	4.3	20.46	8	19.33	73	19.82	39	0.97	4.9
ibm12	24.58	9	n/a	n/a	23.24	33	0.95	3.7	28.09	8	26.23	93	27.04	67	0.96	8.4
ibm13	20.87	11	n/a	n/a	20.51	41	0.98	3.7	24.86	10	24.36	94	23.97	38	0.96	3.8
ibm14	35.44	18	35.24	151	34.08	67	0.96	3.7	46.32	18	n/a	n/a	43.80	76	0.95	4.2
ibm15	46.68	25	49.88	208	46.57	89	1.00	3.6	57.37	22	n/a	n/a	57.67	100	1.01	4.6
ibm16	49.02	25	46.52	232	46.38	94	0.95	3.8	61.82	27	n/a	n/a	58.62	103	0.95	3.8
ibm17	66.93	30	69.26	505	63.60	120	0.95	4.0	81.53	28	n/a	n/a	77.84	127	0.95	4.5
ibm18	45.58	28	46.07	454	45.36	142	1.00	5.1	60.43	30	n/a	n/a	58.11	127	0.96	4.2
						Average	0.98	4.6						Average	0.97	5.0

allocation appear to make comparisons difficult and potentially misleading.

The results for the macro cell benchmarks are shown in Table 3. Note that Capo I corresponds to the "Improved Flow 1 (C)" of [26], while Capo II corresponds to "Flow 2" in [26]. We emphasize that Capo II and Kraftwerk results are *not* legal whereas the results for our tool *are* legalized, and that the ratios comparing our placer to Capo uses the best value from either of the Capo flows for each problem.

Relative to the best of the Capo flows, FDP produced very comparable results. Moreover, FDP achieved results that were 9% better than Kraftwerk and 1% better than mPG. Our run times for these problems also appear to be quite reasonable. These numbers confirm that FDP not only implements but extends the techniques presented in [10], rendering placements that are on-par with current state-of-the-art methods.

CONCLUSIONS

In this paper, we discussed the implementation of a force-directed analytic placer. Several engineering details, such as methods for force computation and spreading assessment, were investigated, and various new methods for improving quality were presented. The quality of our approach was compared to other leading-edge tools and the results were found to be favorable, with performance of most designs usually within 4 times that of Capo 8.7 and results up to 3% better, on average, for unit-sized standard cells.

While top-down partitioning strategies have received considerable attention over the past five years, relatively few details have surfaced about force-directed placers. For instance, we know of no freely-available frameworks for investigations into force-based placement. We feel that the enhancements introduced by our tool have brought it in-line with state-of-the-art technologies, and that by releasing its source code, we may stimulate further investigation in this milieu.[7]

All told, we believe that there is still considerable room to improve the quality of results achieved from force-based techniques. Both the placer and spread metrics can be extended to consider additional objectives, such as routing, congestion, timing, variable whitespace allocation, and 3D placement simply by replacing the quad-tree with an oct-tree. Moreover, we feel that partitioning one or two levels prior to analytic placement could improve results. Cutlines added by a partitioning method would not only have the effect of cre-

[7]The C++ source code for our placer is available for free download at http://gibbon.uwaterloo.ca.

Table 3: Macro cell benchmark results. Run times for Capo and Kraftwerk are observed on a 2 GHz Pentium. Run times for mPG are observed on a Sun Blade 1000 running at 750 MHz. Run times for FDP are observed on a 2.8 GHz Pentium.

Circuit	Capo Flow I		Capo Flow II		Kraftwerk		mPG		FDP		vs.		
	WL	CPU	WL	CPU	WL	CPU	WL	CPU	WL	CPU	Capo	Kraftwerk	mPG
ibm01	3.36	13	2.92	5	3.01	2	3.01	18	2.62	8	0.90	0.87	0.87
ibm02	8.23	240	6.5	11	7.58	9	7.42	32	6.59	19	1.01	0.87	0.89
ibm03	11.53	22	9.63	14	11.4	10	11.2	32	11.17	11	1.16	0.98	1.00
ibm04	11.93	25	11.2	14	12.1	12	10.5	42	10.05	17	0.90	0.83	0.96
ibm06	9.63	19	7.9	17	10.2	12	9.2	45	8.89	17	1.13	0.87	0.97
ibm07	15.80	39	13.6	55	17.1	19	13.7	68	14.09	21	1.04	0.82	1.03
ibm08	18.85	111	17.2	22	18.2	21	16.4	82	15.93	23	0.93	0.88	0.97
ibm09	17.52	178	17.8	31	19.1	28	18.6	84	17.89	31	1.02	0.94	0.96
ibm10	53.58	490	47.5	68	51.5	35	43.6	172	45.44	45	0.96	0.88	1.04
ibm11	26.47	69	25.1	41	26.6	36	26.5	112	26.60	33	1.06	1.00	1.00
ibm12	55.12	119	47.5	51	52.6	43	44.3	153	49.72	39	1.05	0.95	1.12
ibm13	33.56	88	33.4	68	35.9	55	37.7	151	31.83	37	0.95	0.89	0.84
ibm14	52.67	333	47.9	117	47.4	74	43.5	276	47.72	58	1.00	1.01	1.10
ibm15	64.69	264	66.8	122	73.7	93	65.5	285	69.00	115	1.07	0.94	1.05
ibm16	83.14	580	86.7	166	82.4	94	72.4	436	66.82	120	0.80	0.81	0.92
ibm17	91.50	249	87.6	136	92.2	107	78.5	606	80.35	115	0.92	0.87	1.02
ibm18	54.11	397	57.2	158	54.9	110	50.7	437	55.56	160	1.03	1.01	1.10
										Average	1.00	0.91	0.99

ating a more initially-spread design, they would also render the Hessians more diagonally dominant and therefore faster to solve with a conjugate gradient technique. Lastly, we believe that there is still considerable room to improve upon our run times through parameter tweaking and a stretching technique along the lines of [18].

We expect that further research on this placement methodology will continue to be very rewarding, and expect to investigate these ideas for inclusion in the not-too-distant future.

REFERENCES

[1] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits," in *Proc. of DAC*, pp. 211–215, 1995.

[2] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Field-Programmable Logic and Applications* (W. Luk, P. Y. Cheung, and M. Glesner, eds.), pp. 213–222, Springer-Verlag, Berlin, 1997.

[3] A. E. Dunlop and B. W. Kernighan, "A placement procedure for standard-cell VLSI circuits," *Trans. on CAD*, vol. 4, pp. 92–98, January 1985.

[4] A. E. Caldwell, A. B. Kahng, and I. Markov, "Can recursive bisection alone produce routable placements?," in *Proc. of DAC*, pp. 477–482, ACM Press, 2000.

[5] 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. of ISPD*, April 2004.

[6] J. Vygen, "Algorithms for large-scale flat placement," in *Proc. of DAC*, pp. 746–751, ACM Press, 1997.

[7] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *Trans. on CAD*, vol. 10, pp. 356–365, March 1991.

[8] H. Etawil, S. Areibi, and A. Vannelli, "Attractor-repeller approach for global placement," in *Proc. of ICCAD*, pp. 20–24, 1999.

[9] G. Sigl, K. Doll, and F. Johannes, "Analytical placement: A linear or a quadratic objective function?," in *Proc. of DAC*, pp. 427–432, June 1991.

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

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

[12] F. Mo, A. Tabbara, and R. K. Brayton, "A timing-driven macro-cell placement algorithm," in *Proc. of ICCAD*, pp. 322–327, 2001.

[13] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," in *Proc. of FPGA*, pp. 29–36, ACM Press, 2001.

[14] B. Goplen and S. S. Sapatnekar, "Efficient thermal placement of standard cells in 3D ICs using a force directed approach," in *Proc. of ICCAD*, pp. 86–89, IEEE Press, 2003.

[15] "The Boost C++ library." `http://www.boost.org`, Current July 2004.

[16] U. T. Mello and I. Khabibrakhmanov, "On the reusability and numeric efficiency of C++ packages in scientific computing," in *Proc. of the ClusterWorld Conference and Expo*, June 2003.

[17] J. George, *Computer Implementation of the Finite-Element Method.* Ph. D. thesis, Stanford University, 1971.

[18] N. Viswanathan and C. C.-N. Chu, "Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. of ISPD*, April 2004.

[19] J. Barnes and P. Hut, "A hierarchical $O(n \log n)$ force calculation algorithm," *Nature*, vol. 324, 1986.

[20] M. H. Overmars and C.-K. Yap, "New upper bounds in Klee's measure problem," *SIAM Journal on Computing*, vol. 20, no. 6, pp. 1034–1045, 1991.

[21] J. Bentley, "Algorithms for Klee's rectangle problems." Unpublished Manuscript, 1977.

[22] S.-W. Hur, T. Cao, K. Rajagopal, Y. Parasuram, A. Chowdhary, V. Tiourin, and B. Halpin, "Force directed Mongrel with physical net constraints," in *Proc. of DAC*, pp. 214–219, ACM Press, 2003.

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

[24] G. Karypis, *Multilevel Optimization and VLSICAD*, ch. 3. Boston: Kluwer Academic Publishers, 2002.

[25] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *Trans. on VLSI*, vol. 11, pp. 1120–1135, December 2003.

[26] S. Adya and I. Markov, "Combinatorial techniques for mixed-size placement," *Trans. on DAES*, 2004. To appear.

[27] S. Adya and I. Markov, "Consistent placement of macro-blocks using floorplanning and standard-cell placement," in *Proc. of ISPD*, ACM Press, 2002.

[28] A. Saurabh and I. Markov, "ISPD02 mixed-size placement benchmarks." `http://vlsicad.eecs.umich.edu/BK/ISPD02bench`, Current July 2004.

[29] X. Y. M. Wang and M. Sarrafzadeh, "Dragon2000: Standard-cell placement tool for large industry circuits," in *Proc. of ICCAD*, November 2000.

[30] C.-C. Chang, J. Cong, and X. Yuan, "Multi-level placement for large-scale IC designs," in *Proc. of ASPDAC*, pp. 325–330, 2003.