# FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control *

Natarajan Viswanathan, Min Pan and Chris Chu
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011-3060, USA
email: {nataraj, panmin, cnchu}@iastate.edu

**Abstract— In this paper, we present *FastPlace 3.0* – an efficient and scalable multilevel quadratic placement algorithm for large-scale mixed-size designs. The main contributions of our work are: (1) A multilevel global placement framework, by incorporating a two-level clustering scheme within the flat analytical placer *FastPlace* [27, 28]. (2) An efficient and improved Iterative Local Refinement technique that can handle placement blockages and placement congestion constraints. (3) A congestion aware standard-cell legalization technique in the presence of blockages. On the ISPD-2005 placement benchmarks [19], our algorithm is $5.12\times$, $11.52\times$ and $16.92\times$ faster than *mPL6*, *Capo10.2* and *APlace2.0* respectively. In terms of wirelength, we are on average, $2\%$ higher as compared to *mPL6* and $9\%$ and $3\%$ better as compared to *Capo10.2* and *APlace2.0* respectively. We also achieve competitive results compared to a number of academic placers on the placement congestion constrained ISPD-2006 placement benchmarks [20].**

## I. Introduction

In recent years, it has become common to interleave placement with logic synthesis and timing-optimization transforms to create a physical synthesis design flow. As a result, placement needs to be run repeatedly during the early design stages. In addition, circuits today often contain over a million objects that need to be placed. Hence, it is necessary to have efficient and scalable placement algorithms that produce good-quality results satisfying various design objectives including congestion, routability and timing.

Existing placement algorithms employ various approaches including *simulated annealing* [24, 25], *partitioning* [1, 2, 7, 29] and *analytical placement* [4, 9–11, 16, 17, 21, 27, 28]. Analytical placement algorithms based on the quadratic objective funtion (also called quadratic placers) are very popular as they are quite efficient and also give good quality of results. They typically employ a flat placement methodology [9–11, 17, 27, 28] so as to maintain a global view of the placement problem.

But, with circuit sizes steadily increasing towards tens of millions of objects, a flat placement methodology may not be effective in handling the large problem size. Hence, for better scalability and solution quality, a hierarchical placement approach is beneficial. To this effect many modern placers follow a hierarchical or multilevel approach [3, 4, 13, 15, 21, 26].

An essential constraint that needs to be handled by current placers is that of placement congestion. Designers often run placement algorithms with specific *target_density* values. To determine the placement density, a pre-defined bin structure is imposed over the placement region. The *density* of a bin is then defined as the ratio of the total area of the movable objects to the total available free-space within the bin. The *target_density* basically specifies the maximum possible occupation for any bin in the placement region. Satisfying the *target_density* constraint means that the *density* of all the bins in the placement region should be less than or equal to the *target_density* value. The purpose of the *target_density* is to allow for more room within a bin for the subsequent routing step. It also creates space to perform subsequent timing optimization transforms like buffer insertion, gate-sizing etc.

In this paper we address the two issues of scalability and placement congestion. We present *FastPlace 3.0* - an efficient multilevel quadratic placement algorithm with placement congestion control for large-scale mixed-size designs. The main contributions of our work are:

- Incorporating a multilevel framework within the global placement stage of the flat quadratic placer *FastPlace* [27, 28]. This is done by employing two levels of clustering: an intial netlist based fine-grain clustering followed by a netlist and location based coarse-grain clustering.
- An improved Iterative Local Refinement Technique to reduce the wirelength based on the half-perimeter measure. This technique is very effective in simultaneously reducing the wirelength while spreading the objects around the placement region. It can also effectively handle placement blockages and placement congestion constraints.
- A density-aware standard-cell legalization technique. This technique operates on the segments created in the placement region due to the presence of blockages. It satisfies segment capacities and congestion constraints and legalizes the standard-cells within the segments.

The rest of this paper is organized as follows: Section II gives an overview of the multilevel global placement framework and an outline of our algorithm. Section III describes the two-level clustering scheme used during global placement. Section IV describes the improved Iterative Local Refinement technique and its use in placement congestion control. Section V describes the density aware legalization and detailed placement techniques. Experimental results are provided in Section VI followed by the conclusions in Section VII.

*This work was partially supported by the Semiconductor Research Corporation under Task ID 1206 and NSF under grant CCF-0540998.

## II. Overview of the Algorithm

Our multilevel placement framework is summarized in Fig. 1 and follows the classical hierarchical flow that has been used in many existing placement algorithms [3, 4, 6, 13, 15, 21].

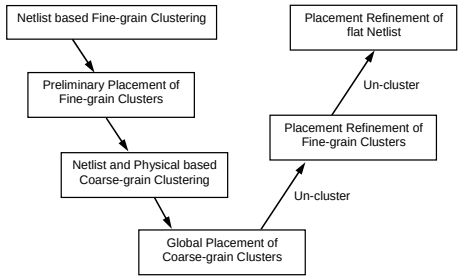


Fig. 1. Multilevel Global Placement Framework.

In Step 1 of the multilevel flow, we create fine-grain clusters of about 2-3 objects per cluster based on the connectivity information of the original flat netlist. In Step 2 we perform a fast initial placement of the fine-grain clusters. In Step 3 we create coarse-grain clusters by performing a second level of clustering. This step considers the connectivity information between the clusters and their physical locations as obtained from the initial placement. This creates a good-quality clustering solution for the subsequent global placement step. In Step 4 we perform global placement on the coarse-grain clustered netlist until the clusters are evenly distributed over the placement region. We then perform a series of un-clustering and placement refinements in Steps 5 and 6, finally yielding a global placement solution of the original flat netlist.

The entire flow of our placement algorithm is summarized in Fig. 2. It consists of three stages: (a) global placement using a multilevel framework, (b) legalization of macro blocks using the Iterative Clustering Algorithm of [28] followed by a density aware standard-cell legalization scheme and (c) an effective detailed placement algorithm [22]. The individual components of the flow are described in more detail in the subsequent sections.

## III. Clustering for Placement

Circuit clustering is an attractive method to reduce the placement problem size for large- scale VLSI designs. If clustering is performed in a careful manner, it can also yield better wirelength along with faster runtime as compared to flat placement approaches. In our multilevel framework we use clustering in a *persistent* context as defined in [21]. As in, we use clustering at the beginning of placement to pre-process the flat netlist so as to reduce the placement problem size.

In our multilevel framework, we follow a two-level clustering scheme as shown in Fig. 1. In the first level of clustering we create fine-grain clusters of about 2-3 objects per cluster. This clustering is solely based on the connectivity information between the objects in the original flat netlist. Since this clustering is performed before any placement, we restrict it to fine-grain clustering to minimize any loss in placement quality due to incorrect clustering. In fact, it was demonstrated in [12] that building fine-grain clusters can improve placement efficiency with negligible loss in placement quality.

We then perform a fast, initial placement of the fine-grain clusters. The purpose of this step is to get some placement information for the next clustering level. Since each cluster in the first level has only around 2-3 objects, the initial placement of the clusters closely resembles an initial placement of the flat netlist. We then create coarse-grain clusters by performing a second level of clustering. In this level, we consider both, the connectivity information between the clusters and their physical locations as obtained from the initial placement. We believe that generating coarse-grain clusters based on actual placement information, is better than generating them by a solely netlist based approach. Also, such an approach would further minimize any loss in (or even improve) the final wirelength.

**Stage 1: Global Placement**
- Level 1: Initial Placement
  1. Construct fine-grain clusters using netlist based clustering
  2. Solve initial quadratic program
  3. Repeat
     a. Perform *regular Iterative Local Refinement* on fine-grain clusters
  4. Until the placement is roughly even
- Level 2: Coarse Global Placement
  5. Construct coarse-grain clusters using netlist and physical based clustering
  6. Repeat
     a. Solve the convex quadratic program
     b. Perform *cell-shifting* on coarse-grain clusters and add spreading forces
  7. Until the placement is roughly even
  8. Repeat
     a. Perform *density-based Iterative Local Refinement* on coarse-grain clusters
     b. Perform *regular Iterative Local Refinement* on coarse-grain clusters
     c. Perform *cell-shifting* on coarse-grain clusters
  9. Until the placement is quite even
- Level 3: Refinement of fine-grain clusters
  10. Un-cluster coarse-grain clusters
  11. Perform *density-based Iterative Local Refinement* on fine-grain clusters
  12. Perform *regular Iterative Local Refinement* on fine-grain clusters
- Level 4: Refinement of flat netlist
  13. Un-cluster fine-grain clusters
  14. Perform *density-based Iterative Local Refinement* on flat netlist
  15. Perform *regular Iterative Local Refinement* on flat netlist

**Stage 2: Legalization**
16. Legalize and fix movable macro-blocks using *Iterative Clustering Algorithm*
17. Move standard-cells among segments to satisfy segment capacities
18. Legalize standard-cells within segments

**Stage 3: Detailed Placement**

Fig. 2. Outline of Our Placement Flow.

The key difference between our clustering scheme and the ones followed in [3, 5, 15, 21] is that we use actual placement information while forming coarse-grain clusters, whereas the other approaches generate coarse-grain clusters solely based on netlist information. Our approach closely resembles that of [13]. The difference being that [13] uses two-levels of netlist based clustering followed by physical clustering, whereas we only use one level of fine-grain netlist based clustering.

For both levels of clustering, we use the *Best-Choice* clustering algorithm described in [21]. In Fig. 3 we summarize the modified version of the *Best-Choice* clustering algorithm using Lazy-Update speed-up technique to consider our two-level clustering scheme. From Fig. 3 there are four key parameters within our clustering scheme:

- $clustering\_ratio$: Ratio of the number of objects before and after clustering.
- $s(j, k)$: The netlist based clustering score between two objects $j$ and $k$.
- *max_cluster_area*: The upper-bound on the cluster area.
- *distance_threshold*: The distance threshold used for the physical clustering.

Within our clustering scheme, for each level of clustering we use a $clustering\_ratio$ of 2 resulting in a $4\times$ reduction in

**Algorithm Clustering**

**Phase 1: Construct Initial Priority-queue (PQ)**

For each object *j*

1. Find closest object *k* and clustering score *s(j, k)*
2. Insert triple *(j, k, s)* into PQ with *s* as the key

**Phase 2: Form Clusters**

while (*number_of_objects* > *target_number_of_objects*)

1. Pick top triple *(j, k, s)* from PQ
2. if *j* is marked invalid
3. Re-calculate closest object *k'* and clustering score *s'(j, k')*
4. Insert triple *(j, k', s')* into PQ
5. else
6. if *fine-grain clustering*
7. if (*a(j)* + *a(k)* < *max_cluster_size*) cluster *j* and *k* into new object *j'*
8. if *netlist + physical clustering*
9. Calculate *d(j, k)* the distance between *j* and *k*
10. if (*d(j, k)* < *distance_threshold* and *a(j)* + *a(k)* < *max_cluster_size*) cluster *j* and *k* into new object *j'*
11. Update netlist based on the clustering
12. For object *j'* find closest object *k'* and clustering score *s'(j', k')*
13. Insert triple *(j', k', s')* into PQ with *s'* as the key
14. Mark neighbours of *j'* as invalid

Fig. 3. Best-Choice Clustering Algorithm with Placement Information.

the number of objects in the final coarse-grain netlist. For the netlist based clustering score between objects $j$ and $k$ we use:

$$s(j,k) = \frac{\Sigma_{\nu \in N} w_\nu}{a_j + a_k}$$

where $N$ is the set of nets connecting the two objects and $w_\nu = 1/k$ where $k$ is the degree of net $\nu$. To strictly control the area of the clusters, we set the *max_cluster_area* to $5\times$ *average_cluster_area*. This results in the formation of balanced clusters. Finally, we experimentally set the *distance_threshold* to $10\%$ of the maximum chip dimension.

## IV. Congestion Aware Iterative Local Refinement

The Iterative Local Refinement (ILR) technique is a key component of our placement flow. It is highly effective in minimizing the wirelength while simultaneously distributing the cells over the placement region. We separate the ILR technique into two components: a density-based ILR *d-ILR* and the regular ILR *r-ILR*. The core algorithm is the same within both the components and hence we only describe it in the context of the *r-ILR*. We first give an overview of the ILR technique of [27], followed by the enhancements. We then describe the top level flow for ILR based placement congestion control.

### A. Description of the Technique

During ILR the placement region is binned and the utilization of all the bins is determined, following which, the respective *source* bins of all the cells is determined. For every cell present in a bin, $8$ scores are computed that correspond to moving it to the $8$ neighboring bins. For calculating the score, it is assumed that a cell is moving from its current position in a *source* bin to the same relative position in the *target* bin. The score for each move is a weighted sum of two components: (a) the half-perimeter wirelength reduction for the move and (b) a function of the utilization of the source and target bins. For each cell and bin, a fixed weight is used to compute the score. The cell is then moved to the bin with the highest positive score.

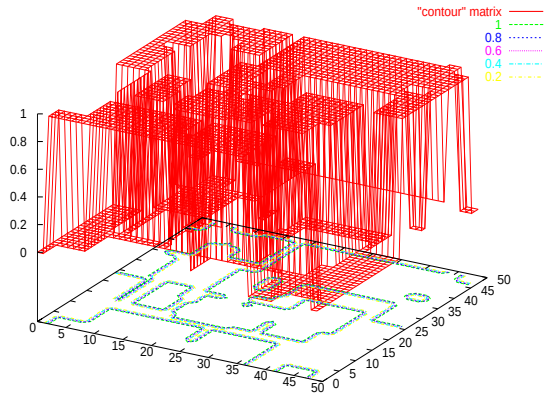


Fig. 4. Initial Contour Map Depicting Placement Blockages.

During one iteration the above steps are followed for all the cells in the placement region. It is then repeated until there is no significant improvement in the wirelength. For the first loop of ILR, the width and height of the bins are set to $5\times$ that of the bin used during Cell Shifting. The bin dimensions are then gradually brought down to the values used during Cell Shifting over subsequent iterations of the global placement.

### B. Enhancements to the ILR Technique

A major drawback with the ILR is that every bin in the placement region, irrespective of if it being sparse or dense, will have the same weight for the utilization component. This does not accurately reflect the placement density. A sparse bin should have a lesser utilization weight so that more cells can be moved into it, whereas, the weight for a dense bin should be higher to enable movement of cells out of this bin. In the enhanced version of ILR each bin has its associated utilization weight that is constantly updated based on the placement distribution.

Another extension to the ILR is in handling placement blockages. ASIC circuits contain many placement blockages in the form of fixed macros. Quadratic placers often place a lot of movable objects on top of the fixed macros. These objects have to be moved out of the fixed macros in an effective manner with minimal increase in the wirelength. To handle fixed macros during placement, we construct a contour map of the placement region. Based on the fixed macros, each bin in the contour map has a value of either $1$ in case it overlaps with a fixed macro or $0$ otherwise. The initial contour map for one of the placement benchmarks is shown in Fig. 4. We then use a $3 \times 3$ Laplacian matrix as a smoothing filter and run it for a specified number of iterations on the entire map. This removes the sharp edges in the original contour map creating a smoothed version as shown in Fig. 5. This smoothing is basically done so that cells can easily move over and cross a fixed macro if required or slide down the slope for it to be moved out of the macro.

Based on the above enhancements, for cell $i$ in bin $m$, if:

- $\alpha$: Weight for the wirelength component.
- $\beta_m$: Weight of the utilization component for bin $m$.
- $\beta_n$: Weight of the utilization component for bin $n$.
- $\gamma$: Weight for the contour component.
- $wl_i(m)$: Half-perimeter wirelength when $i$ is in bin $m$

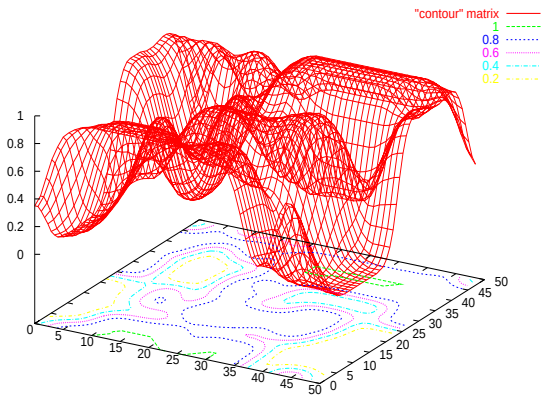


Fig. 5. Contour Map after Smoothing Transform.

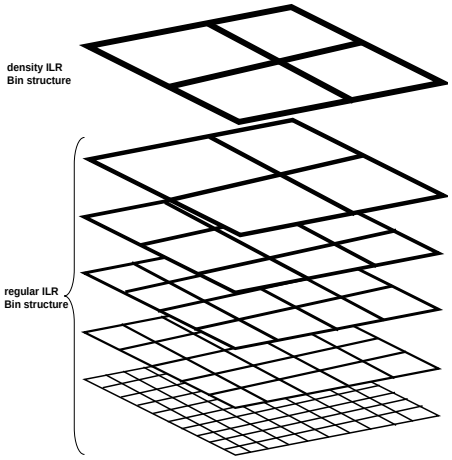


Fig. 6. Bin Structure for Iterative Local Refinement.

- $wl_i(n)$: Half-perimeter wirelength when $i$ is in bin $n$
- $U(m)$: Utilization function for bin $m$
- $U(n)$: Utilization function for bin $n$
- $C(m)$: Contour height of bin $m$
- $C(n)$: Contour height of bin $n$

Then, the score for the move from bin $m$ to bin $n$ is given by:

$$s_i(m,n) =$$

$$\alpha(wl_i(m) - wl_i(n)) + (\beta_m U(m) - \beta_n U(n)) + \gamma(C(m) - C(n))$$

### C. ILR for Placement Congestion Control

For placement congestion control, the ILR is divided into 2 components. The *d-ILR* uses the global pre-defined bin structure used for placement *density* computation. It then calculates the utilization and contour height for these bins. Cells are then moved from *source* to *target* bins of the global bin structure.

Once the *d-ILR* is performed, we then run the *r-ILR* as before in which the bin sizes are initially set to a large value and then decreased over subsequent placement iterations. Fig. 6 depicts the interaction between the *d-ILR* and the *r-ILR* and shows the decrease in the size of the bins from the *d-ILR* stage to the end of the *r-ILR* stage.

## V. Legalization and Detailed Placement

The aim of the legalization stage is to resolve module overlaps, present after global placement, and yield a legal non-overlapping placement. Our legalization stage is divided into two steps: we first ignore all the standard-cells and resolve overlaps among the macro blocks; we then fix the macros and legalize the standard-cells. This is followed by detailed placement. These steps are described in more detail below.

### A. Macro Block Legalization

During legalization, we do not want to move the macros by a significant amount from their global placement positions. Hence, the goal of the macro block legalization algorithm is to resolve overlaps among the macros by perturbing them by the minimum possible distance from their global placement positions. This is achieved by using the *Iterative Clustering Algorithm* [28] for macro block legalization. Due to space constraints, we refer the reader to [28] for more details.

### B. Density Aware Selective Bin-based Cell Legalization

After macro block legalization, we fix their positions and treat them as placement blockages for all subsequent steps. Each row in the placement region is then fragmented into segments based on the overlap of the row with the placement blockages. The aim of the density aware standard-cell legalizer is to satisfy segment capacities as well as placement congestion constraints and legalize the standard-cells within the segments.

To perform legalization, we create a Regular Bin Structure (RBS) over the entire placement region. The height of each bin is equal to the cell row height and its width is equal to around $4\times$ the average cell width. We then determine the utilization of every bin and segment in the placement region. The utilization of a segment is defined as the total width of all the cells within the segment. If the total width is greater than the segment width, the segment is considered to be above capacity.

Based on the segment utilizations and placement blockages, we construct a *move map* of the entire placement region. For each bin in the RBS, this map has a value of either $1$ for allowing movement of cells into or out of this bin, or $0$ otherwise. For bins that completely overlap blockages we assign a value of $0$ as we do not want cells to be moved on top of the blockage. If the utilization of a particular segment is greater than the *target_density*, then a small region of bins in and around the current segment is assigned a value of $1$. This is to allow for move based legalization to be performed only on these bins. This is depicted in Fig. 7 where there are two segments that are above capacity (shown by the diagonal lines). Then, we turn on move based legalization for only a small set of bins around the segments (shown by the shaded regions).

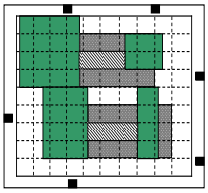

Fig. 7. Selective Bin-based Standard Cell Movement.

For moving the cells among the bins we use a technique similar to the ILR. The difference being that the score for a move during legalization is a weighted sum of three components: (a) the half-perimeter wirelength reduction for the move, (b) a

function of the utilization of the source and target bins and (c) a weighted difference of the *move map* values for the source and target bins. Since the legalization technique is mainly used to even out the placement and satisfy segment capacities, a higher weight is assigned to the second and third components. Once all the segments are brought within capacity, we assign the cells to legal positions within each segment.

The key advantages of the selective bin-based legalizer is that it does not significantly perturb the global placement solution. Secondly, it distributes the cells evenly within the segments. This helps to satisfy placement congestion constraints.

### C. Detailed Placement

To further reduce the wirelength of the placement, we adopt a modified version of the *FastDP* [22] detailed placer that can handle placement congestion constraints.

## VI. Experimental Results

*FastPlace3.0* was tested on the ISPD-2005 Placement Benchmarks [19] and the ISPD-2006 Placement Benchmarks [20]. These benchmarks have been derived from industrial ASIC designs with circuit sizes ranging from 211K to 2.50M objects. In addition, the ISPD-2006 benchmark suite has a specific *target_density* assigned to each circuit.

In Table I, we compare *FastPlace3.0* with the latest available versions of the academic placers *mPL6* [4, 5, 8], *Capo10.2* [23] and *APlace 2.0* [15, 16] on the ISPD-2005 Placement Benchmarks. All the placers were run in their default mode and all experiments were run on a $2.6$ GHZ AMD Opteron $252$ machine with $8$ GB RAM.

From Table I, we have on average, $2\%$ higher wirelength as compared to *mPL6* and $9\%$ and $3\%$ better wirelength as compared to *Capo10.2* and *APlace2.0* respectively. In terms of runtime we are $5.12\times$, $11.52\times$ and $16.92\times$ faster than *mPL6*, *Capo10.2* and *APlace2.0* respectively.

In Table II we compare our results with that of other placers reported during the ISPD 2005 placement contest. It should be noted that for the contest, all the placers were given the benchmarks in advance and there was no limit on the CPU time required to get the best possible results on the individual circuits. From Table II, the contest version of *APlace* is on average $4.5\%$ better than our placer in terms of wirelength. In [15] the authors report that the entire benchmark set takes $113.2$ hrs on a $1.6$ GHZ machine and that they are on average $3\times$ slower than *Capo*. Based on these results our placer is roughly $34\times$ faster than the contest version of *APlace*. It can also be seen that our results are better than the reported results of all the other placers during the ISPD 2005 placement contest.

In Table III we compare our results with that of other placers reported during the ISPD 2006 placement contest. We use the same scoring function as the contest which is a weighted function of wirelength, placement congestion and runtime. On average, we have only $1\%$ higher score than the best reported results during the contest. Looking at individual results, on $4$ of the $8$ benchmarks we are better than the best reported results during the contest.

Table IV gives the runtime comparison of our placer with other placers in the ISPD 2006 placement contest. This is a direct comparison of the runtime, as the machine specifications for the contest are the same as the one on which we ran our experiments. On average, the runtime of our placer is the least among all the placers.

## VII. Conclusions

In this paper we describe *FastPlace 3.0* an efficient and scalable quadratic placer for large-scale mixed-size circuits. It is based on a multilevel global placement framework and incorporates an improved Iterative Local Refinement Technique that can handle placement blockages as well as placement congestion constraints. We also describe an efficient density aware standard-cell legalization scheme.

The current implementation produces competitive results compared to other state-of-the-art academic placers on various benchmark circuits but in a significantly lesser runtime. Such an ultra-fast placer is very much needed in present day iterative physical synthesis flows to achieve timing closure without a significant runtime overhead.

## References


[1] A. R. Agnihotri, S. Ono, C. Li, M. C. Yildiz, A. Khatkhate C.-K. Koh, and P. H. Madden. Mixed block placement via fractional cut recursive bisection. *TCAD*, 24(5):748–761, May 2005.

[2] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection produce routable placements. In *Proc. DAC*, pages 477–482, 2000.

[3] T. Chan, J. Cong, T. Kong, and J. Shinnerl. Multilevel optimization for large-scale circuit placement. In *Proc. ICCAD*, pages 171–176, 2000.

[4] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. ISPD*, pages 185–192, 2005.

[5] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie. mPL6: Enhanced multilevel mixed-size placement. In *Proc. ISPD*, pages 212–214, 2006.

[6] C. C. Chang, J. Cong, and X. Yuan. Multi-level placement for large-scale mixed-size IC designs. In *Proc. ASPDAC.*, pages 325–330, 2003.

[7] T.-C. Chen, T.-C. Hsu, Z.-W. Jiang, and Y.-W. Chang. NTUplace: A ratio partitioning based placement algorithm for large-scale mixed-size designs. In *Proc. ISPD*, pages 236–238, 2005.

[8] J. Cong and M. Xie. A robust detailed placement for mixed-size ic designs. In *Proc. ASPDAC*, pages 188–194, 2006.

[9] H. Eisenmann and F. Johannes. Generic global placement and floorplanning. In *Proc. DAC*, pages 269–274, 1998.

[10] H. Etawil, S. Arebi, and A. Vannelli. Attractor-repeller approach for global placement. In *Proc. ICCAD*, pages 20–24, 1999.

[11] B. Hu and M. Marek-Sadowska. FAR: Fixed-points addition and relaxation based placement. In *Proc. ISPD*, pages 161–166, 2002.

[12] B. Hu and M. Marek-Sadowska. Fine granularity clustering for large scale placement problems. In *Proc. ISPD*, pages 67–74, 2003.

[13] B. Hu and M. Marek-Sadowska. Multilevel fixed-point-addition-based VLSI placement. *TCAD*, 24(8):1188–1203, August 2005.

[14] A. B. Kahng, S. Reda, and Q. Wang. APlace: A general analytic placement framework. In *Proc. ISPD*, pages 233–235, 2005.

[15] A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In *Proc. ICCAD*, pages 890–897, 2005.

[16] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *TCAD*, 24(5):734–747, May 2005.

[17] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *TCAD*, 10(3):356–365, March 1991.

[18] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence pair. *TCAD*, 15(12):1518–1524, December 1996.

[19] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz. The ISPD2005 placement contest and benchmark suite. In *Proc. ISPD*, pages 216–220, 2005.

[20] G.-J. Nam. ISPD 2006 placement contest: Benchmark suite and results. In *Proc. ISPD*, pages 167–167, 2006.

TABLE I
WIRELENGTH AND RUNTIME COMPARISON OF *FastPlace3.0* WITH *mPL6*, *Capo10.2* AND *APlace2.0* ON THE ISPD-2005 BENCHMARK SUITE.

| **Circuit** | **Half-Perimeter Wirelength** | | | | **Runtime (sec)** | | | |
|---|---|---|---|---|---|---|---|---|
| | FastPlace3.0 | $\frac{mPL6}{FP3.0}$ | $\frac{Capo10.2}{FP3.0}$ | $\frac{APlace2.0}{FP3.0}$ | FastPlace3.0 | $\frac{mPL6}{FP3.0}$ | $\frac{Capo10.2}{FP3.0}$ | $\frac{APlace2.0}{FP3.0}$ |
| adaptec1 | 79383680 | 0.98 | 1.15 | 0.99 | 294 | 7.42 | 15.12 | 21.66 |
| adaptec2 | 93084248 | 0.99 | 1.08 | 1.03 | 466 | 4.84 | 12.13 | 19.68 |
| adaptec3 | 217804128 | 0.98 | 1.05 | 1.00 | 1896 | 3.79 | 6.67 | 11.75 |
| adaptec4 | 201358944 | 0.96 | 1.03 | 1.04 | 1176 | 5.75 | 9.80 | 21.37 |
| bigblue1 | 95679992 | 1.01 | 1.14 | 1.05 | 503 | 5.44 | 13.31 | 16.92 |
| bigblue2 | 155101744 | 0.98 | 1.05 | 0.99 | 1150 | 6.78 | 11.56 | 17.42 |
| bigblue3 | 379882464 | 0.91 | 1.05 | 1.08 | 3868 | 2.72 | 9.83 | 9.75 |
| bigblue4 | 832879872 | 1.00 | 1.16 | 1.05 | 5718 | 4.22 | 13.78 | 16.82 |
| **Average** | | **0.98** | **1.09** | **1.03** | | **5.12**× | **11.52**× | **16.92**× |

TABLE II
HALF-PERIMETER WIRELENGTH COMPARISON OF *FastPlace3.0* WITH OTHER ACADEMIC PLACERS ON THE ISPD-2005 BENCHMARK SUITE.

| **Placer** | **Circuit** | | | | | | **Average** |
|---|---|---|---|---|---|---|---|
| | adaptec2 | adaptec4 | bigblue1 | bigblue2 | bigblue3 | bigblue4 | |
| APlace | 0.94 | 0.93 | 0.99 | 0.93 | 0.94 | 1.00 | 0.955 |
| **FastPlace3.0** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.000 |
| mFAR | 0.98 | 0.95 | 1.02 | 1.09 | 1.00 | 1.05 | 1.015 |
| Dragon | 1.02 | 1.00 | 1.07 | 1.03 | 1.00 | 1.09 | 1.034 |
| mPL | 1.04 | 1.00 | 1.03 | 1.12 | 0.97 | 1.09 | 1.041 |
| Capo | 1.07 | 1.05 | 1.13 | 1.11 | 1.01 | 1.32 | 1.115 |
| NTUplace | 1.08 | 1.03 | 1.11 | 1.23 | 1.08 | 1.39 | 1.153 |
| Fengshui | 1.32 | 1.67 | 1.20 | 1.84 | 1.24 | 1.25 | 1.420 |
| Kraftwerk | 1.69 | 1.75 | 1.56 | 2.08 | 1.73 | 1.69 | 1.749 |

TABLE III
*FastPlace3.0* COMPARED TO OTHER ACADEMIC PLACERS ON THE ISPD-2006 BENCHMARK SUITE USING THE ISPD-2006 PLACEMENT CONTEST SCORING FUNCTION.

| **Placer** | **Circuit** | | | | | | | | **Avg** |
|---|---|---|---|---|---|---|---|---|---|
| | adaptec5 | newblue1 | newblue2 | newblue3 | newblue4 | newblue5 | newblue6 | newblue7 | |
| Kraftwerk | 1.01 | 1.19 | 1.00 | 1.00 | 1.01 | 1.04 | 1.00 | 1.00 | 1.03 |
| mPL6 | 1.00 | 1.06 | 1.07 | 1.17 | 1.00 | 1.02 | 1.00 | 1.00 | 1.04 |
| **FastPlace3.0** | 1.12 | 1.15 | 0.96 | 1.09 | 0.98 | 1.11 | 0.96 | 0.93 | 1.04 |
| NTUplace2 | 1.02 | 1.00 | 1.07 | 1.16 | 1.03 | 1.00 | 1.04 | 1.07 | 1.05 |
| mFAR | 1.09 | 1.23 | 1.09 | 1.16 | 1.09 | 1.13 | 1.03 | 1.04 | 1.11 |
| APlace3 | 1.26 | 1.20 | 1.05 | 1.13 | 1.35 | 1.21 | 1.06 | 1.05 | 1.16 |
| Dragon | 1.08 | 1.21 | 1.29 | 1.90 | 1.05 | 1.13 | 1.03 | 1.23 | 1.24 |
| DPlace | 1.26 | 1.55 | 1.77 | 1.36 | 1.14 | 1.35 | 1.23 | 1.25 | 1.36 |
| Capo | 1.16 | 1.57 | 1.64 | 1.44 | 1.22 | 1.28 | 1.32 | 1.46 | 1.39 |

TABLE IV
RUNTIME RESULTS OF *FastPlace3.0* COMPARED TO OTHER ACADEMIC PLACERS ON THE ISPD-2006 BENCHMARK SUITE.

| **Placer** | **Circuit** | | | | | | | | **Avg** |
|---|---|---|---|---|---|---|---|---|---|
| | adaptec5 | newblue1 | newblue2 | newblue3 | newblue4 | newblue5 | newblue6 | newblue7 | |
| **FP3.0 (sec)** | **1973** | **609** | **816** | **1619** | **878** | **3156** | **2519** | **3279** | |
| FastPlace3.0 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00× |
| Kraftwerk | 1.67 | 1.86 | 1.23 | 0.56 | 3.16 | 2.35 | 2.12 | 2.28 | 1.91× |
| mPL6 | 4.19 | 3.70 | 7.47 | 5.99 | 6.62 | 3.91 | 4.78 | 8.66 | 5.66× |
| NTUplace2 | 5.32 | 3.55 | 5.43 | 4.10 | 8.51 | 6.48 | 5.50 | 6.55 | 5.68× |
| mFAR | 3.48 | 4.17 | 3.55 | 1.83 | 7.25 | 3.62 | 4.82 | 5.94 | 4.33× |
| APlace3 | 10.27 | 7.07 | 6.78 | 7.72 | 17.07 | 10.39 | 11.56 | 16.73 | 10.95× |
| Dragon | 1.14 | 1.62 | 2.00 | 0.72 | 1.69 | 1.12 | 1.53 | 3.02 | 1.61× |
| DPlace | 1.46 | 1.69 | 7.84 | 0.64 | 1.88 | 1.44 | 1.60 | 2.90 | 2.43× |
| Capo | 4.93 | 4.21 | 6.92 | 3.75 | 7.89 | 6.61 | 7.34 | 16.76 | 7.30× |


[21] G.-J. Nam, S. Reda, C. J. Alpert, P. G. Villarrubia, and A. B. Kahng. A fast hierarchical quadratic placement algorithm. *TCAD*, 25(4):678–691, April 2006.

[22] M. Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *Proc. ICCAD*, pages 48–55, 2005.

[23] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov. Min-cut Floorplacement. *TCAD*, 25(7):1313–1326, Jul 2006.

[24] C. Sechen and A. L. Sangiovanni-Vincentelli. TimberWolf 3.2: A new standard cell placement and global routing package. In *Proc. DAC*, pages 432–439, 1986.

[25] W.-J. Sun and C. Sechen. Efficient and effective placement for very large circuits. *TCAD*, 14(5):349–359, 1995.

[26] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M. Sarrafzadeh. Dragon2005: Large-scale mixed-size placement tool. In *Proc. ISPD*, pages 245–247, 2005.

[27] N. Viswanathan and C. C.-N. Chu. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. *TCAD*, 24(5):722–733, May 2005.

[28] N. Viswanathan, M. Pan, and C. Chu. Fastplace 2.0: An efficient analytical placer for mixed-mode designs. In *Proc. ASPDAC*, pages 195–200, 2006.

[29] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proc. ICCAD*, pages 260–263, 2000.