A High-Quality Mixed-Size Analytical Placer Considering Preplaced Blocks and Density Constraints *

Tung-Chieh Chen[1], Zhe-Wei Jiang[1], Tien-Chang Hsu[1], Hsin-Chen Chen[2], and Yao-Wen Chang[1,2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
{donnie, crazying, tchsu, indark}@eda.ee.ntu.edu.tw; ywchang@cc.ee.ntu.edu.tw

ABSTRACT

In addition to wirelength, modern placers need to consider various constraints such as preplaced blocks and density. We propose a high-quality analytical placement algorithm considering wirelength, preplaced blocks, and density based on the log-sum-exp wirelength model proposed by Naylor et al. [20] and the multilevel framework. To handle preplaced blocks, we use a two-stage smoothing technique, Gaussian smoothing followed by level smoothing, to facilitate block spreading during global placement. The density is controlled by white-space re-allocation using partitioning and cut-line shifting during global placement and cell sliding during detailed placement. We further use the conjugate gradient method with dynamic step-size control to speed up the global placement and macro shifting to find better macro positions. Experimental results show that our placer obtains the best published results.

1. INTRODUCTION

High-performance IC designs usually require significant white space for further performance optimization, such as buffer insertion and gate sizing. Therefore, density control and white-space allocation become very important. A wirelength-driven placer without considering placement density tends to pack blocks together to minimize wirelength. However, an over congested region may not have enough white space for buffer insertion, and thus degrade the chip performance. Although some congestion-aware placement algorithms were proposed [18, 23], these algorithms intend to minimize the routing congestion, which is different from the density control since density can still be high for some regions as long as no routing overflows occur in those regions.

Further, modern chip designs often consist of many preplaced blocks, such as analog blocks, memory blocks, and/or I/O buffers, which are fixed in the chip and cannot overlap with other blocks. These preplaced blocks impose more constraints on the placement problem. A placement algorithm without considering preplaced blocks may generate illegal placement or inferior solutions.

Based on three sets of state-of-the-art benchmark suites, most of the recently proposed placement algorithms can handle the mixed-size constraints [4–6, 8, 12, 15, 24]. However, very few modern mixed-size placement algorithms can handle preplaced blocks and the chip density well. In this paper, we present a high-quality mixed-size analytical placement algorithm considering preplaced blocks and density constraints. Our placer has the following distinguished features:

- Based on the log-sum-exponential wirelength model [1] proposed by Naylor et al. [20] and the multilevel framework, our placer consistently generates high-quality mixed-size placement results.
- To solve the unconstrained minimization placement objective function, we use the conjugate gradient method with a dynamic step size. Experimental results show that the method leads to significant run-time speedup.
- Our placer handles preplaced blocks by a two-stage smoothing technique. The preplaced block potential is first smoothed by a Gaussian function to remove the rugged potential regions, and then the potential levels are smoothed so that the blocks can spread to the whole placement region effectively.
- Density constraints are considered during both global and detailed placement. We re-allocate white space using partitioning and cut-line shifting to remove density overflows between different levels of global placement. In detailed placement, a cell-sliding technique is applied to remove the density overflow.
- A macro-shifting technique is used between levels of global placement to find better macro positions that are easier for legalization.
- A look-ahead legalization scheme during global placement is used to obtain a better legal placement result.

Table 1 summarizes the comparisons between our placer and two state-of-the-art analytical placers, APlace 2.0/3.0 [14, 16] and mPL5/6 [6, 7], which are also based on the log-sum-exp wirelength model. In the table, "Unknown" denotes that the corresponding method is not available in the literature.

The remainder of this paper is organized as follows. Section 2 gives the analytical model used in our placer. Our core placement techniques are explained in Section 3. Section 4 reports the experimental results. Finally, the conclusions are given in Section 5.

2. ANALYTICAL PLACEMENT MODEL

The circuit placement problem can be formulated as a hypergraph $H = (V, E)$ placement problem. Let vertices $V = \{v_1, v_2, ..., v_n\}$ represent blocks and hyperedges $E = \{e_1, e_2, ..., e_n\}$ represent nets. Let x_i and y_i be the x and y coordinates of the center of block v_i, and a_i be the area

*This work was partially supported by MediaTek Inc., National Science Council of Taiwan under Grant No's. NSC 94-2215-E-002-030 and NSC 94-2752-E-002-008-PAE, and RealTek Semiconductor Corp.

ICCAD'06 November 5–9, 2006, San Jose, CA

[1]The log-sum-exponential wirelength model is a patented technology [20] and use requires a license from Synopsys.

Table 1: Comparisons between our placer and APlace and mPL; all the placers are based on the analytical technique and the log-sum-exp wirelength model. Unknown: not mentioned in the corresponding work.

	APlace 2.0/3.0	mPL5/mPL6	Ours
Global Placement Framework	V-cycle multilevel framework	W-cycle multilevel framework (mPL5) V-cycle multilevel framework (mPL6)	V-cycle multilevel framework
Clustering	Best-choice clustering	First-choice clustering (mPL5) Best-choice clustering (mPL6)	First-choice clustering
Wirelength Model	Log-sum-exp	Log-sum-exp	Log-sum-exp
Spreading Force	Bell-shaped potential	Poisson smoothed potential	Bell-shaped potential
Nonlinear Objective Solver	Conjugate gradient method w/ golden section line search	Explicit Euler method	Conjugate gradient method w/ dynamic step-size control
Preplaced Block Handling	Level smoothing	Poisson equation smoothing	Gaussion and level smoothing
Density Handling	Unknown	Network-flow-based cell redistribution	White-space allocation, Cell sliding
Macro Block Handling	Unknown	Linear programming based macro legalization	Macro shifting
Look-Ahead Legalization	No	No	Yes

of the block v_i. The circuit may contain some *preplaced blocks* which have fixed x and y coordinates and cannot be moved. We intend to determine the optimal positions of movable blocks so that the total wirelength is minimized and there is no overlap among blocks. The placement problem is usually solved in three stages, (1) global placement, (2) legalization, and (3) detailed placement. Global placement evenly distributes the blocks and finds the best position for each block to minimize the target cost (e.g., wirelength). Then, legalization removes all overlaps. Finally, detailed placement refines the solution.

Figure 1 gives the notation used in this paper.

x_i, y_i	center coordinate of block v_i
w_i, h_i	width and height of block v_i
w_b, h_b	width and height of bin b
M_b	the maximum area of movable blocks in bin b
D_b	potential (area of movable blocks) in bin b
P_b	base potential (preplaced block area) in bin b
$t_{density}$	target placement density

Figure 1: Notation used in this paper.

To evenly distribute the blocks, we divide the placement region into uniform non-overlapping bin grids. Then, the global placement problem can be formulated as a constrained minimization problem as follows:

$$\begin{array}{ll} \min & W(\mathbf{x},\mathbf{y}) \\ \text{s.t.} & D_b(\mathbf{x},\mathbf{y}) \le M_b, \quad \text{for each bin b,} \end{array} \tag{1}$$

where $W(\mathbf{x},\mathbf{y})$ is the wirelength function, $D_b(\mathbf{x},\mathbf{y})$ is the potential function that is the total area of movable blocks in bin b, and M_b is the maximum area of movable blocks in bin b. M_b can be computed by $M_b = t_{density}(w_b h_b - P_b)$, where $t_{density}$ is a user-specified target density value for each bin, w_b (h_b) is the width (height) of bin b, and P_b is the *base potential* that equals the preplaced block area in bin b. Note that M_b is a fixed value as long as all preplaced block positions are given and the bin size is determined.

The wirelength $W(\mathbf{x},\mathbf{y})$ is defined as the total half-perimeter wirelength (HPWL). Since $W(\mathbf{x},\mathbf{y})$ is non-convex, it is hard to minimize it directly. Thus, several smooth wirelength approximation functions are proposed, such as quadratic wirelength [9, 17], L_p-norm wirelength [7, 16], and log-sum-exp wirelength [6, 15, 20]. The log-sum-exp wirelength model,

$$\gamma \sum_{e\in E}(\log \sum_{v_k\in e} \exp(x_k/\gamma) + \log \sum_{v_k\in e} \exp(-x_k/\gamma) + \log \sum_{v_k\in e} \exp(y_k/\gamma) + \log \sum_{v_k\in e} \exp(-y_k/\gamma)), \tag{2}$$

proposed in [20], achieves the best result among these three models [7]. When γ is small, log-sum-exp wirelength is close to the HPWL [20]. However, due to the computer precision, we can only choose a reasonably small γ, say, 1% length of the chip width, so that it will not cause any arithmetic overflow.

Since density $D_b(\mathbf{x},\mathbf{y})$ is neither smooth nor differentiable, mPL [7] uses inverse Laplace transformation to smooth the density, while APlace [15] uses a bell-shaped function for each block to smooth the density. We express the function $D_b(\mathbf{x},\mathbf{y})$ as $D_b(\mathbf{x},\mathbf{y}) = \sum_{v\in V}^{n} P_x(b,v)P_y(b,v)$, where P_x and P_y are the overlap functions between bin b and block v along the x and y directions. In [15], a bell-shaped potential function p_x provides the smoothed version of P_x. By doing so, the non-smooth function $D_b(\mathbf{x},\mathbf{y})$ can be replaced by a smooth one, $D'_b(\mathbf{x},\mathbf{y}) = \sum_{v\in V}^{n} c_v p_x(b,v)p_y(b,v)$, where c_v is a normalization factor so that the total potential of a block equals its area.

The quadratic penalty method is used to solve Equation (1), implying that we solve a sequence of unconstrained minimization problems of the form

$$\min \quad W(\mathbf{x},\mathbf{y}) + \lambda \sum_b (D'_b(\mathbf{x},\mathbf{y}) - M_b)^2 \tag{3}$$

with increasing λ's. The solution of the previous problem is used as the initial solution for the next one. We solve the unconstrained problem in Equation (3) by the conjugate gradient (CG) method. We use CG with a dynamic step size to minimize Equation (3).

3. PROPOSED ALGORITHM

Our placement algorithm consists of three stages: (1) global placement, (2) legalization, and (3) detailed placement.

3.1 Global Placement

3.1.1 Multilevel Framework

We use the multilevel framework for global placement to improve the scalability. Our algorithm is summarized in Figure 2. Lines 1–4 are the coarsening stage. The initial placement is generated in line 5. Lines 6–23 are uncoarsening stages. The details of each step are explained in the following.

During the coarsening stage, we cluster blocks to reduce the number of movable blocks. The hierarchy of clusters is built by the first-choice (FC) clustering algorithm [7].

After clustering, the initial placement for the coarsest level is generated by minimizing the quadratic wirelength using the conjugate gradient method, the same method in quadratic placement.

Then, we solve the placement problem from the coarsest level to the finest level. The placement for the current level

Algorithm: **Multilevel Global Placement**
Input:
hypergraph H_0: mixed-size circuit
n_{max}: the maximum block number in the coarsest level
Output:
(x^*, y^*): optimal block positions

01. $level = 0$;
02. **while** $(BlockNumber(H_{level}) > n_{max})$
03. $level{+}{+}$;
04. $H_{level} = FirstChoiceClustering(H_{level-1})$;
05. initialize block positions by $SolveQP(H_{level})$;
06. **for** $currentLevel = level$ **to** 0
07. initialize bin grid size $n_{bin} \propto \sqrt{n_x}$;
08. initialize base potential for each bin;
09. initialize $\lambda_0 = \frac{\sum |\partial W(\mathbf{x},\mathbf{y})|}{\sum |\partial D'_b(\mathbf{x},\mathbf{y})|}$; $m = 0$;
10. **do**
11. solve min $W(\mathbf{x},\mathbf{y}) + \lambda_m \sum (D'_b - M_b)^2$;
12. $m{+}{+}$;
13. $\lambda_m = 2\lambda_{m-1}$;
14. **if** $(currentLevel == 0$ & $overflow\_ratio < 10\%)$
15. call $LookAheadLegalization()$ and save the best result;
16. compute $overflow\_ratio$;
17. **until** (spreading enough or no further reduction in $overflow\_ratio$)
18. **if** $(currentLevel == 0)$
19. restore the best look-ahead result;
20. **else**
21. call $MacroShifting()$;
22. call $WhiteSpaceAllocation()$;
23. decluster and update block positions.

Figure 2: Our global placement algorithm.

provides the initial placement for the next level. In each level, the bin grid size is set according to the number of clusters, the base potential P_b for each bin is computed, and the maximum area of movable blocks M_b is updated accordingly. Then, the value of λ is initialized according to the strength of wirelength and density gradients, $\lambda = \frac{\sum |\partial W(\mathbf{x},\mathbf{y})|}{\sum |\partial D'_b(\mathbf{x},\mathbf{y})|}$, and a conjugate gradient solver with dynamic step-size control is used to solve the constrained minimization problem in Equation (1) (in lines 10–17).

Macro shifting and *white-space allocation for density control* are applied between uncoarsening levels. We will explain them in Section 3.1.4 and Section 3.1.5, respectively. Then, blocks are declustered, providing the initial placement for the next level.

We define the *overflow ratio* as the total overflow area in each bin over the area of total movable blocks as follows:

$$overflow\_ratio = \frac{\sum_{\text{Bin } b} \max(D_b(\mathbf{x},\mathbf{y}) - M_b, 0)}{\sum \text{total movable area}}, \quad (4)$$

where $overflow\_ratio \geq 0$.

Our placer uses the overflow ratio to measure the evenness of block distribution, instead of the *discrepancy* as in [15]. The overflow ratio has a more global view since it considers all overflow areas in the placement region while discrepancy only considers the maximum density of a window in the placement region. The global placement stage stops when the overflow ratio is less than a user-specified target value, which is 0 by default.

3.1.2 Base Potential Smoothing

Preplaced blocks pre-define the *base potential*, which significantly affects block spreading. Since the base potential P_b is not smooth, it forms mountains that prevent movable blocks from passing through these regions. Therefore, we shall smooth the base potential to facilitate block spreading. We first use the Gaussian function to smooth the base potential change, remove the rugged regions in the base potential, and then smooth the base potential level so that blocks can spread to the whole placement region.

The base potential of each block can be calculated by

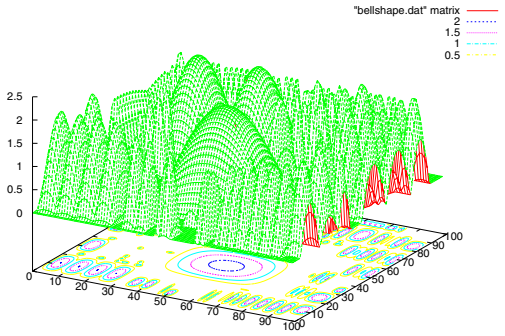

Figure 3: Base potential using the bell-shaped function. The z-coordinate is the value of $P_b/(w_b h_b)$. Note for a region with potential level > 1.0, it means that the base potential in the region is larger than the bin area.

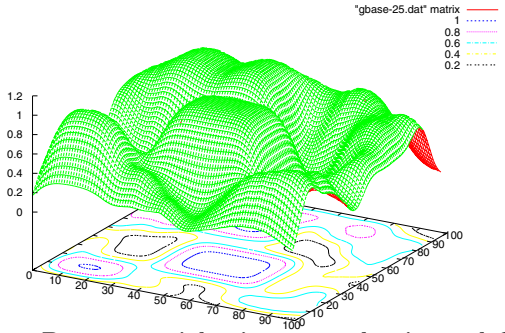

Figure 4: Base potential using exact density and Gaussian smoothing, resulting a better smoothing potential.

the bell-shaped function. However, we observe that the potential generated by the bell-shaped function has "valleys" between the adjacent regions of blocks. Figure 3 shows the base potential generated by the bell-shaped function. The z-coordinate is the value of $P_b/(w_b h_b)$. If a bin has $z > 1$, it means that the potential in the bin is larger than the bin area. There are several valleys in the bottom-left regions as shown in the figure, and these regions do not have free space but their potentials are so low that a large number of blocks may spread to these regions. To avoid this problem, we calculate the exact density as the base potential, and then use the Gaussian function to smooth the base potential. The two-dimensional Gaussian has the form

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad (5)$$

where σ is the standard deviation of the distribution. Applying convolution to the Gaussian function G with the base potential P, $P'(x,y) = G(x,y) * P(x,y)$, we can obtain a smoother base potential P'. Gaussian smoothing works as a low-pass filter, which can smooth the local density change, and the value σ defines the smoothing range. A larger σ leads to a more smooth potential. In global placement, the smoothing range gradually decreases so that the smoothed potential approaches the exact density gradually. Figure 4 shows the resulting potential by using σ being 0.25 times of the chip width.

After the Gaussian smoothing, we apply another landscape smoothing function [10, 13] to reduce the potential levels. The smoothing function $P''(x,y)$ is defined as follows:

$$P''(x,y) = \begin{cases} \overline{P'} + (P'(x,y) - \overline{P'})^\delta & \text{if } P'(x,y) \geq \overline{P'} \\ \overline{P'} - (\overline{P'} - P'(x,y))^\delta & \text{if } P'(x,y) \leq \overline{P'}, \end{cases} \quad (6)$$

Algorithm: Conjugate Gradient Algorithm with Dynamic Step-Size Control

Input:
$f(x)$: objective function
x_0: initial solution
s: step size
Output:
optimal x^*

01. initialize $g_0 = 0$ and $d_0 = 0$;
02. **do**
03. compute gradient directions $g_k = \nabla f(x_k)$;
04. compute the Polak-Ribiere parameter $\beta_k = \frac{g_k^T(g_k - g_{k-1})}{||g_{k-1}||^2}$;
05. compute the conjugate directions $d_k = -g_k + \beta_k d_{k-1}$;
06. compute the step size $\alpha_k = s/||d_k||_2$;
07. update the solution $x_k = x_{k-1} + \alpha_k d_k$;
08. **until** $(f(x_k) > f(x_{k-1}))$

Figure 5: Our nonlinear placement objective solver.

where $\delta \geq 1$. δ decreases from a large number (say 5) to 1, and a series of level-smoothed potential is generated. Smoothing potential levels reduces "mountain" (high potential regions) heights so that blocks can spread to the whole placement area smoothly.

3.1.3 The Conjugate Gradient Algorithm with Dynamic Step Sizes

We use the conjugate gradient (CG) algorithm to minimize Equation 3. APlace uses the golden section line search to find the optimal step size, which takes most portion of its runtime during the minimization process. Instead, our step size is computed by a more efficient method. After computing the conjugate gradient direction d_k, the step size α_k is computed by $\alpha_k = s/||d_k||_2$, where s is a user-specified scaling factor. By doing so, we can limit the step size of block spreading since the total quadratic Euclidean movement is fixed,

$$\sum_{v_i \in V} (\Delta x_i^2 + \Delta y_i^2) = ||\alpha_k d_k||_2^2 = s^2, \quad (7)$$

where Δx_i and Δy_i are the amount of the movement along the x and y directions for the block v_i in each iteration.

The value of s affects the precision of objective minimization; smaller s values lead to better results but longer runtime. In our implementation, we set s between 0.2 and 0.3 times of the bin width to obtain a good tradeoff between runtime and quality. Figure 5 gives our conjugate gradient algorithm for minimizing the placement objective during global placement.

3.1.4 Macro Shifting

In the global placement stage, it is important to preserve legal macro positions since illegal macro positions may make the task of legalization much more difficult. To avoid this, we apply macro shifting at each declustering level of the global placement stage. Macro shifting moves macros to the closest legal positions.

Integrating with our multilevel framework, only macros with sizes larger than the average cluster size of the current level are processed. Then, the legal macro positions provide a better initial solution for the next declustering level, and those macros are still allowed to spread at subsequent declustering levels.

3.1.5 White-Space Allocation for Density Control

After block spreading, some regions may still have overflows. We reduce the overflows by assigning appropriate amount of white space. Unlike the method proposed in [18] that applies white-space allocation to reduce the routing congestion, we use white-space allocation to remove overflow regions. We recursively partition the placement region and construct a slicing tree to record the cut directions and blocks inside the partition until the partitioned area is similar to that of a global placement bin. To prevent from generating sub-partitions with large aspect ratios, we choose the larger side to divide the partition into two sub-partitions evenly. The process is similar to a partitioning-based global placement flow, and the difference is that we divide the partition based on geometric locations of blocks instead of the cut size minimization.

After the construction of the partitions and the slicing tree, we compute the white space in each partition, and update the data structures for the leaf nodes of the slicing tree. A negative white space value $w < 0$ means that the partition has an overflow area of $|w|$. Then, the white space of an internal node can be computed by summing up the white space of its two child nodes.

After the white space calculation, the white spaces are distributed to the two child nodes in a top-down process according to the following rules:

- If one child node has white space $w < 0$, we allocate white space of $|w|$ to this child node, and allocate the remaining white space to the other child node.
- If two child nodes both have white spaces greater than or equal to 0, we allocate the white space proportional to their original white space amount.

The new partition area a' can be computed by $a' = a + w' - w$, where a is the old partitioned area, w is the old white space, and w' is the new white space. The cut-line adjustment is also performed in a top-down fashion. We can know the desired areas of the two sub-partitions from the data structure of the two child nodes, and then the cut line is shifted accordingly.

Finally, the new block positions can be computed by linear interpolation of the coordinates of the old partition and the new one.

3.1.6 Look-Ahead Legalization

It is often hard to determine when to stop the block spreading during global placement. If blocks do not spread enough, the wirelength may significantly be increased after legalization since blocks are over congested. If blocks spread too much, the wirelength before legalization may not be good even the legalization step only increases wirelength a little. This situation becomes even worse when the density is also considered, since the placement objective is more complex.

We use a look-ahead legalization technique to find a desired solution. At the finest level, we apply legalization after minimizing nonlinear objective in each iteration and record the best result that has the minimum cost (wirelength and density penalty). Although look-ahead legalization may take longer runtime due to more iterations of legalization, we can ensure that blocks do not over spread and thus obtain a better legal placement.

3.2 Legalization

To obtain a better solution from the global placement result, the legalization stage removes all overlaps with minimal total displacement. We extend the standard-cell legalization method in [11] to solve the mixed-size legalization problem. In our legalization stage, the legalization order of macros and cells are determined by their x coordinates and sizes. We legalize macros earlier. Then, in the legalization order, cells are packed into rows while macros are placed to their nearest available positions. We find this macro/cell legalization strategy works well on all benchmarks.

3.3 Detailed Placement

3.3.1 Wirelength Minimization

We extend the window-based detailed placement (WDP) algorithm [12] and name our approach *cell matching* here. The WDP algorithm finds a group of exchangeable cells inside a given window, and formulates a bipartite matching problem by matching the cells to the empty slots in the window. The cost is given by the HPWL difference of a cell

in each empty slot. The bipartite matching problem can be solved optimally in polynomial time, but the optimal assignment cannot guarantee the optimal HPWL result because the HPWL cost of a cell to each empty slot depends on the positions of the other connected cells. Our cell matching algorithm remedies this drawback by selecting *independent cells* at one time to perform bipartite matching. Here by independent cells, we mean that there is no common net between any pairs of the selected cells.

3.3.2 Density Optimization

In addition to wirelength minimization during the detailed placement, we optimize the chip density by cell sliding. The objective of density optimization is to reduce the density overflow in the congested area. In this stage all macro blocks are fixed, and we consider standard cells only. We divide the placement region into uniform non-overlapping bins, and then our algorithm iteratively reduces the densities of overflowed bins by sliding the cells horizontally from denser bins to sparser bins while the cell order is preserved. Each iteration consists of two phases: left sliding and right sliding. In each phase, we calculate the density of each bin and then compute the area flow $f_{bb'}$ between bin b and its left or right neighboring bin b'. $f_{bb'}$ denotes the desired amount of cell area to move from bin b to b'. Recall that we define D_b as the total area of the movable cells in bin b, and M_b as the maximum area of movable blocks in bin b. If bin b does not have any area overflow or the area overflow ratio of b is smaller than b', that is $D_b \leq M_b$ or $D_b/M_b \leq D_{b'}/M_{b'}$, we set $f_{bb'} = 0$. Otherwise we calculate $f_{bb'}$ according to the capacity of b'. If bin b' has enough free space, we move the overflow area of bin b to b'. Otherwise we evenly distribute the overflow area between b and b'. Therefore, $f_{bb'}$ is defined by

$$f_{bb'} = \begin{cases} D_b - M_b, & if\ (M_{b'} - D_{b'}) \geq (D_b - M_b) \\ \frac{D_b M_{b'} - D_{b'} M_b}{M_b + M_{b'}}, & otherwise, \end{cases} \tag{8}$$

where the second condition of Equation (8) is derived from

$$D_b - \left(M_b + \frac{(D_b - M_b + D_{b'} - M_{b'})M_b}{M_b + M_{b'}} \right) = \frac{D_b M_{b'} - D_{b'} M_b}{M_b + M_{b'}}. \tag{9}$$

After the area flow $f_{bb'}$ is computed, we sequentially slide the cells across the boundary between b and b' until the amount of sliding area reaches $f_{bb'}$ or there is no more area for cell sliding. Then we update D_b and $D_{b'}$. In the right sliding phase, we start from the left-most bin of the placement region, and b' is right to b. In the left sliding phase, we start from the right-most bin, and b' is left to b, accordingly. We iterative slide the cells from the area overflow region to a sparser region until no significant improvement can be obtained.

4. EXPERIMENTAL RESULTS

We compared our placer with APlace 2.0 and mPL5, which achieved best published results among all publicly available placers, based on the ICCAD'04 IBM mixed-size [1] and the ISPD'05 placement contest [2] benchmark suites. All results were generated on the same PC workstation with an Opteron 2.4GHz CPU based on the default parameters given in each placer, and no manual parameter tuning for individual circuits is allowed for fair comparison.

We also compared with other eight state-of-the-art academic placers, such as APlace 3.0 and mPL6, based on the ISPD'06 placement contest benchmark suite [3]. Since the eight academic placers are not available to us, we reported the results given in [3, 19].

4.1 ICCAD'04 IBM Mixed-Size Benchmarks

In the first experiment, we evaluated the performance of our placer on the ICCAD'04 IBM mixed-size benchmark suite. Table 2 lists the HPWLs and CPU times for our placer, APlace 2.0, and mPL5, where APlace 2.0 and mPL5 were both performed in the default mode. The last row in Table 2 shows the average normalized wirelength and CPU time ratio based on our results. Compared with APlace 2.0, our placer achieves 1% shorter wirelength and is 7.87X faster. Compared with mPL5, our placer obtains 3% shorter wirelength and is 3.31X faster. On average, our placer produces the best solution quality in smaller runtime.

Table 2: Comparison among our placer (NTUplace3), APlace 2.0, and mPL5 on the ICCAD'04 IBM mixed-size benchmarks.

	NTUplace3		APlace 2.0		mPL5	
	HPWL (×e6)	CPU (sec)	HPWL (×e6)	CPU (sec)	HPWL (×e6)	CPU (sec)
ibm01	2.17	30	2.14	346	2.22	83
ibm02	4.63	57	4.65	793	4.68	240
ibm03	6.65	65	6.71	923	6.86	273
ibm04	7.21	81	7.57	888	7.69	237
ibm05	9.66	145	9.69	696	10.09	118
ibm06	5.94	86	6.02	879	6.16	473
ibm07	9.90	199	10.00	1178	9.96	629
ibm08	12.29	214	12.50	1349	11.92	1030
ibm09	12.00	194	12.13	1670	13.15	1239
ibm10	28.49	319	28.83	2408	29.36	1504
ibm11	17.54	305	18.67	3467	17.87	974
ibm12	32.07	302	33.42	3330	33.43	1290
ibm13	22.16	487	22.80	3495	22.52	981
ibm14	35.36	1158	35.92	4294	34.99	1444
ibm15	45.38	1337	46.81	4926	50.88	4535
ibm16	57.59	1450	54.53	5554	55.21	5636
ibm17	66.73	1930	65.67	6032	66.96	1937
ibm18	41.58	2613	41.99	9932	43.99	2252
average	1.00	1.00	1.01	7.87	1.03	3.30

Table 3: Comparison among our placer (NTUplace3), APlace 2.0, and mPL5 on the ISPD'05 placement contest benchmarks.

	NTUplace3		APlace 2.0		mPL5	
	HPWL (×e6)	CPU (sec)	HPWL (×e6)	CPU (sec)	HPWL (×e6)	CPU (sec)
adaptec1	80.93	803	78.35	7001	87.40	4419
adaptec2	89.85	824	95.70	9793	105.41	4389
adaptec3	214.20	1767	218.52	24849	263.03	5020
adaptec4	193.74	2114	209.28	29377	232.03	5080
bigblue1	97.28	1523	100.01	10318	112.04	2832
bigblue2	152.20	3047	153.75	24789	201.57	5715
bigblue3	348.48	5687	411.59	44805	432.41	23653
bigblue4	829.16	10280	871.29	115363	956.17	25414
average	1.00	1.00	1.05	10.32	1.19	3.31

4.2 ISPD'05 Placement Contest Benchmarks

Table 3 lists the results of ours, APlace 2.0, and mPL5. As shown in the table, our placer achieves the best average wirelength in the shortest CPU time. On average, our resulting HPWL is smaller than that of APlace 2.0 by 5% and mPL5's by 19%, and our placer is 10.32X and 3.31X faster than APlace 2.0 and mPL5, respectively.

4.3 ISPD'06 Placement Contest Benchmarks

In the third experiment, we reported the results on the ISPD'06 placement contest benchmark suite [3]. The results of other placers were taken from [3, 19]. Table 4, Table 5, and Table 6 compare the HPWL, density HPWL, and CPU time of the placers on the ISPD'06 benchmarks, respectively. The density HPWL (DHPWL) is defined as follows [3, 19]:

$$DHPWL = HPWL \times (1 + density\_penalty). \tag{10}$$

To compute *density_penalty*, we made the bin grid width and height equal to 10 circuit row height, and *density_penalty* is

Table 4: HPWL (×e6) comparison based on the ISPD'06 benchmarks.

HPWL	NTUplace3	APlace3 [16]	Capo [21]	DPlace	Dragon [22]	FastPlace	Kraftwerk	mFAR	mPL6 [6]	NTUplace2 [12]
adaptec5	378.56	449.61	491.60	463.95	500.24	478.47	444.07	448.43	425.12	404.98
newblue1	60.74	73.26	98.35	102.37	80.76	84.49	78.29	77.36	66.90	62.40
newblue2	198.76	197.42	308.64	324.07	259.95	209.73	205.87	211.65	197.53	201.95
newblue3	278.87	273.63	361.21	379.19	524.41	361.05	279.94	303.58	283.80	291.14
newblue4	274.48	377.55	358.28	305.78	340.70	319.08	311.09	307.73	294.43	284.99
newblue5	474.84	545.90	657.40	600.11	613.34	601.45	555.48	567.65	530.67	494.57
newblue6	484.81	522.58	668.33	674.39	572.19	539.16	537.32	527.36	510.40	504.39
newblue7	1056.78	1098.26	1518.49	1398.85	1408.97	1173.15	1139.17	1135.80	1070.33	1116.86
average	1.00	1.13	1.41	1.37	1.36	1.21	1.12	1.14	1.06	1.04

Table 5: Density HPWL (×e6) comparison based on the ISPD'06 benchmarks.

DHPWL	NTUplace3	APlace3 [16]	Capo [21]	DPlace	Dragon [22]	FastPlace	Kraftwerk	mFAR	mPL6 [6]	NTUplace2 [12]
adaptec5	448.58	520.97	494.64	572.98	500.74	805.63	457.92	476.28	431.14	432.58
newblue1	61.08	73.31	98.48	102.75	80.77	84.55	78.60	77.54	67.02	63.49
newblue2	203.39	198.24	309.53	329.92	260.83	212.30	208.41	212.90	200.93	203.68
newblue3	278.89	273.64	361.25	380.14	524.58	362.99	280.93	303.91	287.05	291.15
newblue4	301.19	384.12	362.40	364.45	341.16	429.78	315.53	324.40	299.66	305.79
newblue5	509.54	613.86	659.57	752.07	614.23	962.06	569.36	601.27	540.67	517.63
newblue6	521.65	522.73	668.66	682.87	572.53	574.18	545.94	535.96	518.70	532.79
newblue7	1099.66	1098.88	1518.75	1438.99	1410.54	1236.34	1170.85	1153.76	1082.91	1181.30
average	1.00	1.10	1.34	1.41	1.29	1.38	1.08	1.10	1.01	1.02

Table 6: CPU time (sec) comparison based on the ISPD'06 benchmarks. Our CPU time is measured on an Opteron 2.4GHz machine, while others are on an Opteron 2.6GHz machine.

CPU Time	NTUplace3	APlace3 [16]	Capo [21]	DPlace	Dragon [22]	FastPlace	Kraftwerk	mFAR	mPL6 [6]	NTUplace2 [12]
adaptec5	4718	20267	9718	2877	2257	4055	3293	6875	8265	10494
newblue1	1168	4303	2562	1026	989	516	1135	2538	2252	2163
newblue2	2750	5533	5642	6393	1631	1033	1007	2892	6089	4425
newblue3	1670	12503	6076	1028	1170	2437	912	2958	9696	6646
newblue4	3627	14982	6926	1647	1486	1388	2772	6362	5815	7468
newblue5	17955	32799	20854	4550	3529	6224	7423	11426	12349	20441
newblue6	9679	29124	18485	4032	3860	4157	5350	12154	12035	13849
newblue7	20436	54852	54962	9508	9902	6624	7465	19484	28385	21464
average	1.00	3.64	2.20	0.75	0.51	0.58	0.59	1.38	2.08	1.92

defined by

$$density\_penalty = (overflow\_ratio \times bin\_area \times density\_target)^2, \quad (11)$$

and $overflow\_ratio$ is defined by Equation (4).

Among all placers, we obtained both the best average HPWL and the best average DHPWL. Further, according to the scoring function in the 2006 ISPD Placement Contest [3, 19], placers with 2X (4X) CPU time incur about 4% (8%) penalty. Therefore, our overall result considering (1) HPWL, (2) density penalty, and (3) the CPU factor, is the best among all participating placers, and is about 4%, 5%, and 6% better than the three leading placers, Kraftwerk, mPL6, and NTUplace2, respectively.

5. CONCLUSION

We have proposed in this paper a high-quality mixed-size analytical placer considering preplaced blocks and density constraints. Experimental results have shown that our placer achieves very high-quality placement results and is very efficient.

6. REFERENCES

[1] *ICCAD04 Mixed-Size Placement Benchmarks.* http://vlsicad.eecs.umich.edu/BK/ICCAD04bench/.

[2] *ISPD 2005 Placement Contest.* http://www.sigda.org/ispd2005/contest.htm.

[3] *ISPD 2006 Program.* http://www.ispd.cc/program.html.

[4] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. Unification of partitioning, placement and floorplanning. In *Proc. of ICCAD*, pages 550–557, 2004.

[5] A. R. Agnihotri, S. Ono, and P. H. Madden. Recursive bisection placement: Feng Shui 5.0 implementation details. In *Proc. of ISPD*, pages 230–232, 2005.

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

[7] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. of ISPD*, pages 185–192, April 2005. Best paper award at ISPD'2005.

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

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

[10] J. Gu and X. Huang. Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). *IEEE Trans. on Systems, Man and Cybernetics*, 24(5):728–735, 1994.

[11] D. Hill. US patent 6,370,673: Method and system for high speed detailed placement of cells within an intergrated circuit design. 2002.

[12] Z.-W. Jiang, T.-C. Chen, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. NTUplace2: A hybrid placer using partitioning and analytical techniques. In *Proc. of ISPD*, pages 215–217, 2006.

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

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

[15] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *IEEE Trans. on CAD*, 24(5), May 2005.

[16] A. B. Kahng and Q. Wang. A faster implementation of APlace. In *Proc. of ISPD*, pages 218–220, 2006.

[17] M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. on CAD*, 10(3):356–365, 1991.

[18] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden. Routability-driven placement and white space allocation. In *Proc. of ICCAD*, pages 394–401, 2004.

[19] G.-J. Nam, C. J. Aplert, and P. G. Villarrubia. The ISPD 2006 placement contest and benchmark suite. In *Slides presented at ISPD'06*, 2006.

[20] W. C. Naylor, R. Donelly, and L. Sha. US patent 6,301,693: Non-linear optimization system and method for wire length and dealy optimization for an automatic electric circuit placer. 2001.

[21] J. Roy, D. Papa, A. Ng, and I. Markov. Satisfying whitespace requirements in top-down placement. In *Proc. of ISPD*, pages 206–208, 2006.

[22] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M. Sarrafzadeh. Dragon2006: Blockage-aware congestion-controlling mixed-size placer. In *Proc. of ISPD*, pages 209–211, 2006.

[23] X. Yang, B.-K. Choi, and M. Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. In *Proc. of ISPD*, pages 42–47, 2002.

[24] B. Yao, H. Chen, C.-K. Cheng, N.-C. Chou, L.-T. Liu, and P. Suaris. Unified quadratic programming approach for mixed mode placement. In *Proc. of ISPD*, pages 193–199, 2005.