Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement *

Xiaojian Yang Bo-Kyung Choi Majid Sarrafzadeh

Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095

xjyang,bkchoi,majid@cs.ucla.edu

ABSTRACT

The use of white space in fixed-die standard-cell placement is an effective way to improve routability. In this paper, we present a white space allocation approach that dynamically assigns white space according to the congestion distribution of the placement. In the top-down placement flow, white space is assigned to congested regions using a smooth allocating function. A post allocation optimization step is taken to further improve placement quality. Experimental results show that the proposed allocation approach, combined with a multilevel placement flow, significantly improves placement routability and layout quality.

In our experiments, we compared our placement tool with two other fixed-die placers using an industrial place and route flow. Placements created by all three tools have been routed with an industrial router (Warp Route of Cadence). Compared with a leading-edge industrial tool, our placer produces placements with similar or better routability and on average 8.8% shorter routed wirelength. Furthermore, our tool produces placement that runs faster through the Warp Route compared with the industrial tool. Compared with a state-of-the-art academic placement tool (Capo/MetaPlacer), our placer shows ability to produce more routable placements: for 15 out of all 16 benchmarks our placer's outputs are routable while Capo/MetaPlacer only creates 4 routable placements.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Experimentation

Keywords

Physical Design, placement, routability

*This work was supported by NSF under Grant #CCR-0090203

ISPD'02, April 7-10, 2002, San Diego, California, USA.

1. INTRODUCTION

Achieving auto-routability is one of the main goals in modern standard-cell placement. Total estimated wirelength, or bounding box wirelength, was widely used as an objective function to optimize routability. It is commonly believed that a shorter total wirelength implies better routability. Wirelength optimization has been extensively studied during the past two decades. Successful placement techniques include min-cut [1], simulated annealing [2], and analytical approach [3].

Congestion is another important indicator of routability in placement, and it has become dominant for large, tight designs. Previous works addressed the congestion problem using various methods. Mayrhofer and Lauther [4] combine congestion function in cut minimization. Cheng [5] employs a congestion model in simulated annealing approach. Wang et. al [6] propose a post-processing step to remove congestion for a wirelength optimized placement. These methods attempt to reduce congestion by obtaining a placement with less gathered wires and are successful for improving routability.

White space allocation is another way to alleviate congestion in placement orthogonal to above congestion management techniques. White space is a term associated with *fixed-die* placement, which is the common design style in current industry practice. For fixed-die designs, chip area, core area, rows and available sites are given before placement and routing. White space, or the empty space that is not occupied by the standard-cells, varies from 0.1% to 50% for real designs.

The appearance of the fixed-die style does not dramatically change the placement methodology — most previous placement techniques (cut minimization, quadratic approach, simulated annealing) are still applicable. However, the white space in fixed-die mode introduces new problems in placement. For instance, in fixed-die placement with large white space, purely minimizing wirelength tends to place all the cells close to each other. But the congestion of this "packed" placement is worse than a spread-out placement. Fixed-die placement tool has to take white space into consideration to improve routability.

In general, there are two ways to utilize white space in placement: (a) increasing the tolerance in cut minimization to achieve better partitioning quality [7], or (b) allocating white space to congested area to alleviate congestion. The latter one has not drawn enough research attention. One previous work is congestion aware region growing/shrinking by Parakh *et al* [8]. The idea of white space allocation is straightforward: since congestion originates from the discrepancy between routing demand and routing supply, increasing supply, as well as reducing demand, is a natural way to reduce congestion. However, the problem of allocating white space without much loss of placement quality (e.g., wirelength) is not trivial.

In this paper, we present a fixed-die white space allocation approach that significantly improves the routability of the placement. During the placement, white space is dynamically assigned to congested places according to the current congestion distribution. The proposed white space allocation method, combined with a multi-level placement flow, yields high quality placements for fixed-die designs. Experimental results show that, compared to a leading-edge industrial tool, our fixed-die placer produces placements with similar or better routability, shorter routed wirelength and less vias.

The remainder of the paper is organized as follows. In Section 2, we give the background and notations of white space allocation. We then propose the problem and present our solution in Section 3. In Section 4, the process of incorporating white space allocation into top-down placement is discussed. Experimental results compared with other fixed-die placers are shown in Section 5. We conclude in Section 6.

2. BACKGROUND AND DEFINITIONS

In *fixed-die* standard-cell placement, the *core area* consists of *rows* and spaces between rows[1]. Each row has a fixed number of available *sites*. The *total available sites* is the summation of the available sites for all the rows. After the placement stage, every standard-cell occupies an integer number of sites. The *row utilization* is the percentage of sites that are occupied by cells. The *relative white space*, or *white space* of the design is the percentage of sites that are not occupied by cells.

In this paper, we use *bin* based placement as in [9, 10]. During the placement, the core area is divided into rectangular bins and cells are placed at the centers of the bins with overlaps. Bin size varies during the *top-down* placement. Bins become smaller and contain less cells as the placement process goes deeper.

In top-down placement flow, white space is allocated at later levels, where the congestion information acquired from the current placement can be used to guide allocation. Basically, we tend to allocate more white space to congested areas.

In a $m \times n$ bin mesh, assuming that the congestion of each bin is known, we want to determine the white space of each bin. Let c_{ij} be the congestion of the bin at column i and row j, and w_{ij} be the white space to be assigned into this bin. We use W to denote the total (relative) white space of the design.

3. WHITE SPACE ALLOCATION

In this work, we propose a two-step white space allocation approach. We first allocate white space to each row of the bin mesh, then allocate white space to each bin within rows. We will describe them in the next two sections.

3.1 Row White Space Allocation

Assume that there are n rows in the design and the total congestion for row j is c_j. Let w_j be the white space to be allocated to row j. The capacity constraint is,

$$\sum_{j=1}^{n} w_j = W \tag{1}$$

[1]In this work, we assume that there is no space between rows. This is the case for most practical designs.

The total white space assigned to each row should be balanced, i.e., there is no row containing too much or too little white space[2]. Let w_{min} and w_{max} be the minimum and maximum white space for rows, respectively. We then have the following constraints:

$$w_{min} \leq w_j \leq w_{max} \qquad j = 1, ..., n \tag{2}$$

The problem of row white space allocation is to find a function f such that the white space of each row can be computed by its congestion ($w_j = f(c_j)$). The function should be monotone, i.e.,

$$\begin{aligned} w_i \leq w_j \quad if \quad c_i \leq c_j \\ 1 \leq i, j \leq n \end{aligned} \tag{3}$$

We first sort rows according to their congestions. After sorting the congestions of the rows are in non-decreasing order, i.e.,

$$c_1 \leq c_2 \leq \ldots \leq c_n$$

Hence,

$$w_{min} \leq w_1 \leq w_2 \leq \ldots \leq w_n \leq w_{max}$$

Ideally, we want to achieve $w_1 = w_{min}$ and $w_n = w_{max}$. A quadratic function may fit well in this situation. Let $f(x)$ be in the form $f(x) = a_1x^2 + a_2x + a_3$. According to the above constraints,

$$a_1c_1^2 + a_2c_1 + a_3 = w_{min} \tag{4}$$

$$a_1c_n^2 + a_2c_n + a_3 = w_{max} \tag{5}$$

$$a_1 \sum_{i=1}^{n} c_i^2 + a_2 \sum_{i=1}^{n} c_i + a_3 n = W \tag{6}$$

where $c_i, w_i, i = 1, ..., n$ are known and a_1,a_2 and a_3 are unknown parameters. The solution of equation (4) (5) and (6) determines function $f(x)$.

However, the function $f(x)$ may have extremum inside $[c_1, c_n]$, This happens when $c_1 < -a_2/(2a_1) < c_n$. In these cases, the function is no longer monotone within $[c_1, c_n]$. We need to relax either $w_1 = w_{min}$ or $w_n = w_{max}$ to satisfy the capacity constraint.

In the case $a_1 < 0$, we relax the constraint $w_1 = w_{min}$ to $w_1 \geq w_{min}$ and take the point (c_n, w_{max}) as the extremum of the quadratic function. We replace (5) by

$$c_n = -\frac{a_2}{2a_1} \tag{7}$$

The solution of (4), (6) and (7) determines the function that is monotone within $[c_1, c_n]$.

Similarly, in the case $a1 > 0$, we relax the constraint $w_n = w_{max}$ to $w_n \leq w_{max}$ and take the point (c_1, w_{min}) as the extremum of the quadratic function. We replace (4) by

$$c_1 = -\frac{a_2}{2a_1} \tag{8}$$

The solution of (5), (6) and (8) determines the function that is monotone within $[c_1, c_n]$.

The obtained quadratic function is used to compute white space for each row according to the congestion of this row. The white space of the row is then assigned to each bin.

[2]The reason of bouding white space in rows will be discussed in Section 4.2

3.2 Bin White Space Allocation

Unlike row white space allocation, there is no maximum or minimum white space limitation for bin white space allocation. The white space for a bin can be zero, if the bin is not congested. If a bin is highly congested, its neighbor bin is likely to be congested as well. This prevents one congested bin from being assigned too much white space.

For each bin, it is reasonable to allocate white space proportional to the ratio of the congestion to the total congestion, i.e., $w_{ij} = w_j c_{ij}/c_j$. Other ratios can be used, for instance, the ratio of the bin congestion square to the total square of the bin congestion. The specific model used to allocate bin white space varies and should take congestion model into consideration.

4. USING WHITE SPACE ALLOCATION IN PLACEMENT

Several factors need to be considered when incorporating white space allocation into top-down placement. Congestion estimation and detailed placement approach are two of them. Congestion estimation can be fast and rough [5, 11], or accurate but relatively slow [12]. Estimates can be obtained at the early placement stages [13]. White space allocation based on fast congestion estimation can be used frequently in the placement process since it takes little extra time, whereas allocation guided by accurate congestion estimation is computationally expensive and should be avoided as possible. The detailed placement approach also affects the usage of white space allocation. In general, placement quality (e.g. wirelength) degrades after white space allocation, since the locations of the cells are changed without considering the wirelength. Therefore, the allocation must not be the final step of the placement. A detailed placement optimization, usually a low temperature annealing step, is a good solution to the loss of quality in white space allocation.

In this section, we will first describe our placement flow used for our fixed-die placer in Section 4.1. Then we will discuss the white space usage in the context of this specific placement flow in Section 4.2. A post allocation optimization step will be presented in 4.3.

4.1 Placement Flow

We take a top-down placement flow which combines several traditional techniques. The placement process is mainly composed by partitioning (cut minimization) and simulated annealing (wirelength optimization). The given circuit is recursively bipartitioned into subcircuits/clusters in the top-down flow, and these clusters are placed into the bins. For bisection we use the state-of-the-art multilevel partitioner [14]. After every few partitionings, a cluster based low temperature simulated annealing is applied to improve the wirelength. The cluster size and the bin size become smaller as the placement process goes deeper. Once the average cluster size is small enough, a detailed placement process takes control. The steps in the detailed placement includes: adjusting the bins to match the rows, annealing to improve placement quality, removing overlap to obtain the legal placement, and local improvement.

For this specific placement flow, we use the white space allocation two times in the placement, both are in the detailed placement stage. The congestion information at this stage is more accurate. The first allocation is made after bin adjustment. At this moment, the number of rows in the bin mesh is the same as the number of standard-cell rows for the design. The second allocation is made after the annealing improvement step, and before the overlap removal. The entire placement flow with white space allocation steps is shown in Figure 1.

Input: Placement with overlapped cells at $m \times n$ bins
Output: White space $w_{ij} (1 \le i \le m, 1 \le j \le n)$ for each bin.

$W \leftarrow 1 - row\_utilizationn$
$w_{min} \leftarrow 0.01/n$
$w_{max} \leftarrow (1 - 0.9 \cdot row\_utilization)/n$
Estimate congestion using wire probability within bounding box.
Obtain congestion for bin (i, j) as c_{ij}.
Sum up congestion for each row: $c_j = \sum_{i=1}^{m} c_{ij}$
$c_1 \leftarrow minimum(c_j), 1 \le j \le n$
$c_n \leftarrow maximum(c_j), 1 \le j \le n$
Solve equation system (4) (5) and (6)
if $c_1 < -a_2/(2a_1) < c_n$ **then**
  if $a_1 < 0$ **then**
    Solve equation system (4) (6) and (7)
  else
    Solve equation system (5) (6) and (8)
  end if
end if
for each row j **do**
  $w_j \leftarrow a_1 c_j^2 + a_2 c_j + a_3$
  Assign white space to bins: $w_{ij} \leftarrow w_j \cdot c_{ij}/c_j, 1 \le i \le m$
end for

Figure 2: White space allocation algorithm

4.2 Use of White Space Allocation

In the top-down placement flow, we allocate white space only two times although the allocation introduces almost no overload for our placement approach. One reason is that we may replace it with a better approach which does good congestion estimation thus takes longer time.

Congestion estimation method also impacts on the performance of the white space allocation. Different congestion models, or different ways to calculate congestion for each bin, will lead to different white space distribution. In this work, we use a modified congestion model originally proposed in [5]. It is based on the wire probability within the bounding box of the net. Congestion for each bin is computed by the horizontal and vertical wires passing through this bin, and is scaled using the average congestion over all the bins. The estimation takes almost no overload for the placement process.

The maximum and minimum white space in row white space allocation (discussed in Section 3) affect the white space distribution as well. Their values relate to (a) the worst congestion of the entire core area, and (b) the row utilization of the design. For instance, a tight design with highly congested area needs a larger maximum row white space. The existence of maximum and minimum row white space is important. If the row white space is too large, it is difficult to make good allocation for multiple congested regions. Moreover, excessive row white space may considerably increase the wirelength of the placement. If the row white space is too small, it will degrade the quality of simulated annealing by imposing unnecessary row balance penalty. Generally, it is hard to find a direct function for maximum and minimum row white space. In this work, we experimentally set the minimum row white space to $0.01/n$, and set the maximum row white space to $(1 - 0.9 \cdot row\_utilization)/n$, where n is the number of rows.

Figure 2 shows the algorithm for white space allocation in our placement flow.

4.3 Post Allocation Optimization

The optimization steps after two allocations are crucial for the

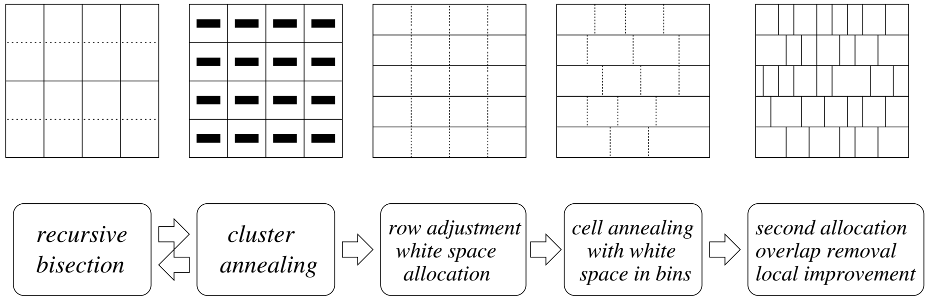
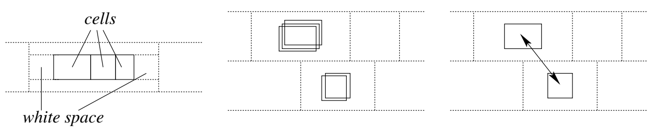


Figure 1: Our placement flow with two white space allocations.

successful allocation, as white space allocation changes the current placement and results in loss of placement quality. We use a simulated annealing based step after the first allocation, and a fast greedy algorithm after the second allocation. Both steps improve the total wirelength by swapping or moving cells. We describe the first approach in this section.



(a) Bin width is determined by cell width and white space (b) Cells are placed at the bin center with overlap (c) Swapping cells between two bins

Figure 3: Simulated annealing after white space allocation.

As shown in Figure 3, after the white space allocation, each bin is assigned some white space. The width of a bin is determined by the white space and the total widths of cells in this bin. Before simulated annealing, the locations of bins are computed based on the actual cell width in each bin. We name it *bin spreading* step. This step will be repeated periodically in simulated annealing. All the cells are placed at the centers of the bins with overlaps. For each move in annealing, cells are either swapped between two bins or are moved from one bin to another. The calculation of row overflow penalty should take the white space of this row into consideration, i.e., penalizing the rows for which the total cell width plus white space is greater than the row capacity.

5. EXPERIMENTAL RESULTS

We have implemented the white space allocation algorithm and have incorporated it into our previous standard-cell placement tool named Dragon [10]. To test our approach, we combined an industrial router — Cadence Warp Route (Silicon Ensemble version 5.3) — into our place and route flow. We compare our placer (Dragon) with Cadence QPlace (Silicon Ensemble version 5.3, running in congestion driven mode) and Capo/MetaPlacer[3] from UCLA PD tool set (September 2001 version) [7]. All three placers read the same LEF/DEF files and write DEF files as the placement output. The Warp Route then reads the placements and does both global and final routing[4].

To obtain appropriate benchmarks for our experiments, we investigate both MCNC and IBM-PLACE benchmarks [15]. MCNC benchmarks are old and most circuits in the suit are small. IBM-PLACE benchmarks are in reasonable size, but lack physical information. We scale the cells in IBM-PLACE according to the cell sizes in TSMC 0.18μm standard-cell library[5], add routing layer information, and translate the benchmarks into LEF/DEF format. We then use Cadence Silicon Ensemble to floorplan the circuits and create design instances using different row utilizations. Specifically, for each circuit we use two row utilizations in floorplanning and create a relatively easier instance and a relatively harder one. The row utilizations for the created test cases are ranging from 85% to 95%[6]. The statistics of the circuits are listed in Table 1.

circuits	*cells*	*nets*	*rows*	*core(row) utilization*	*white space*	*routing layers*
ibm01-easy	12,028	11,753	132	85.12%	14.88%	4
ibm01-hard			130	88.00%	12.00%	4
ibm02-easy	19,062	18,688	153	90.42%	9.58%	5
ibm02-hard			149	95.28%	4.72%	5
ibm07-easy	44,811	44,681	233	89.95%	10.05%	5
ibm07-hard			226	95.30%	4.70%	5
ibm08-easy	50,672	48,230	243	90.03%	9.97%	5
ibm08-hard			236	95.16%	4.84%	5
ibm09-easy	51,382	50,678	246	90.24%	9.76%	5
ibm09-hard			240	95.12%	4.88%	5
ibm10-easy	66,762	64,971	321	90.22%	9.78%	5
ibm10-hard			313	95.08%	4.92%	5
ibm11-easy	68,046	67,422	281	90.11%	9.89%	5
ibm11-hard			273	95.33%	4.67%	5
ibm12-easy	68,735	68,376	347	85.22%	14.78%	5
ibm12-hard			338	90.06%	9.94%	5

Table 1: Tested circuit statistics, including number of cells, number of nets, number of rows, row utilization, white space ratio, and number of routing layers. Core utilization and row utilization are the same since there is no space between rows. Each circuit is floorplanned twice using a higher row utilization and a lower one, resulting a relatively harder design instance and a relatively easier one.

Table 2 shows the comparison of routability using different white space allocation method. We compare (a) placement with the routability driven allocation approach proposed in this paper, (b) placement with no white space allocation (packed placement), and (c) placement with uniformly distributed white space. From the table

[3]It is the only academic fixed-die placer as we know.

[4]For Capo/MetaPlacer, there is a quick legalization step between placement and routing as described in [7]. This step usually takes a couple of seconds and does minor changes on the placement. The reason of this legalization step is probably that Capo/MetaPlacer does not place all the cells exactly on sites (according to personal communication with the author of Capo/MetaPlacer).

[5]Obtained from Artisan Components Inc.

[6]We failed to find "hard" test cases with 70% to 80% utilization (or less). All the circuits except ibm01 are not routable if we use four routing layers. Whereas in five layer cases, 70% or 80% utilization makes placement and routing so easy that it is meaningless to compare wirelength.

one can see that, although the packed placement have the shortest wirelength among three placements, its routability is deteriorated. Evenly distributing white space is slightly better than no white space at all, but it is of little help on alleviating congestion. Allocating white space according to the congestion produces the best routability among three approaches.

Table 3 shows the comparison between Cadence QPlace, our placer (Dragon) and Capo/MetaPlacer. For each placement, we report total bounding box wirelength and runtime. Note that the runtime of QPlace is on Sun workstation and runtime of Dragon or MetaPlacer is on PC/Linux[7]. The routing result can be one of the three types: *successful* routing without violation, *finished* routing with some violations, or *failed* routing (too many violations or routing takes too much time). We also report the total routed wirelength and number of vias if the routing is successful or finished. Due to the intensive runtime of our experiments, we run each placement and routing only once.

Compared with other two placers, our placer (Dragon) produces placements with similar or better routability on all the tested circuits. Out of 16 circuit instances, Dragon creates 15 routable (zero violation) placements while QPlace and MetaPlacer create 13 and 4, respectively. For the only instance that both QPlace and Dragon fail (ibm12-hard), the number of violations for our placement is less than that of QPlace's. Routing time is another metric of routability. For most test cases, the placements by our placer require less routing time than those by QPlace.

For almost all the test cases, our placer (Dragon) produces placements with shorter wirelength and less vias after final routing. Figure 4 illustrates the comparison on total routed wirelength between QPlace's outputs and Dragon's. On average Dragon improves the total routed wirelength by 8.8% and reduces the vias by 4.0%, compared with QPlace.

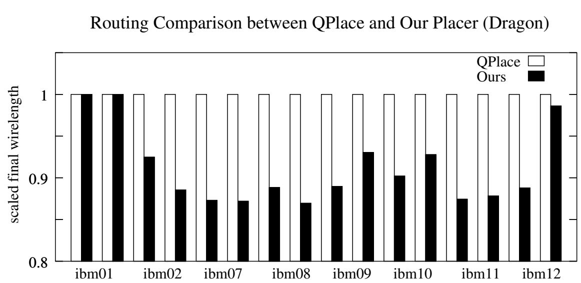


Figure 4: Routed wirelength comparison between QPlace and our placer. Wirelengths are scaled with QPlace's result as unity.

6. CONCLUSION

We have present an approach for routability driven white space allocation. Experimental results show that it is an effective way for fixed-die placement. The proposed method, combined with a multilevel placement flow, creates high quality placements compared with a leading-edge industrial tool and a state-of-the-art academic placement tool.

A promising aspect of our approach is that the routability is improved by white space allocation alone. Previous work on congestion optimization, such as moving highly connected cells out of congested area, can work with our approach concurrently. It is reasonable to expect that a combined congestion optimization would give better routability than merely using white space allocation.

[7]Our experiments show that for placement applications, our PC is usually 1.5-1.8 times faster than our Sun workstation.

The benchmarks and our placement tool are available through our research group webpage http://er.cs.ucla.edu.

7. ACKNOWLEDGMENTS

The authors are grateful to the reviewers and to Ryan Kastner, Helen Hu for their helpful comments.

8. REFERENCES

[1] M. A. Breuer. "A Class of Min-cut Placement Algorithms". In *Design Automation Conference*, pages 284–290. IEEE/ACM, 1977.

[2] C. Sechen and A. Sangiovanni-Vincentelli. "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package". In *Design Automation Conference*, pages 432–439. IEEE/ACM, 1986.

[3] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization". *IEEE Transactions on Computer Aided Design*, 10(3):365–365, 1991.

[4] S. Mayrhofer and U. Lauther. "Congestion-Driven Placement Using a New Multi-partitioning Heuristic". In *International Conference on Computer-Aided Design*, pages 332–335. IEEE, 1990.

[5] C. E. Cheng. "RISA: Accurate and Efficient Placement Routability Modeling". In *International Conference on Computer-Aided Design*, pages 690–695, 1994.

[6] M. Wang, X. Yang, and M. Sarrafzadeh. "Congestion Minimization During Placement". *IEEE Transactions on Computer Aided Design*, 19(10):1140–1148, 2000.

[7] A. E. Caldwell, A. B. Kahng, and I. L. Markov. "Can Recursive Bisection Alone Produce Routable Placements?". In *Design Automation Conference*, pages 477–482. IEEE/ACM, June 2000.

[8] P. N. Parakh, R. B. Brown, and K. A. Sakallah. "Congestion Driven Quadratic Placement". In *Design Automation Conference*, pages 275–278. IEEE/ACM, June 1998.

[9] M. Sarrafzadeh and M. Wang. "NRG: Global and Detailed Placement". In *International Conference on Computer-Aided Design*. IEEE, November 1997.

[10] M. Wang, X. Yang, and M. Sarrafzadeh. "Dragon2000: Fast Standard-cell Placement for Large Circuits". In *International Conference on Computer-Aided Design*, pages 260–263. IEEE, 2000.

[11] X. Yang, R. Kastner, and M. Sarrafzadeh. "Congestion Reduction During Placement Based on Integer Programming". In *International Conference on Computer-Aided Design*, pages 573–576. IEEE, 2001.

[12] J. Lou, S. Krishnamoorthy, and H. S. Sheng. "Estimating Routing Congestion using Probabilistic Analysis". In *International Symposium on Physical Design*, pages 112–117. ACM, April 2001.

[13] X. Yang, R. Kastner, and M. Sarrafzadeh. "Congestion Estimation During Top-down Placement". In *International Symposium on Physical Design*, pages 164–169. ACM, April 2001.

[14] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. "Multilevel Hypergraph Partitioning: Application in VLSI Domain". In *Design Automation Conference*, pages 526–529. IEEE/ACM, 1997.

[15] NuCAD. "IBM-PLACE benchmark". http://www.ece.nwu.edu/nucad/ibm-place.html.

circuit	RD allocation		packed placement		even allocation	
	wirelength	routing	wirelength	routing	wirelength	routing
ibm01-easy	0.576	*Success*	0.499	*Failure*	0.509	*Failure*
ibm02-easy	1.539	*Success*	1.420	*Failure*	1.446	*Failure*
ibm07-easy	3.548	*Success*	3.159	*Failure*	3.222	*Finished*(107)
ibm08-easy	3.659	*Success*	3.327	*Finished*(20)	3.508	*Finished*(2)

Table 2: Comparison between routability of placements by three white space allocation approaches: routability-driven (RD) allocation, no allocation (packed placement) and evenly distributed allocation. Routing results include successful, finished (with number of violations reported) and failure (because of too many violations or time out).

Circuits	Placer	Placement		Routing				
		wire length	*run time*	*result*	*viola-tions*	*wire length*	*vias*	*run time*
ibm01-easy	QPlace	0.59	3	*Success*	0	0.86	133478	20
	Dragon	0.58	16	*Success*	0	0.86	136734	34
	MetaPlacer	0.56	2	*Failure*	-	-	-	-
ibm01-hard	QPlace	0.59	3	*Finished*	12	0.87	150736	189
	Dragon	0.56	15	*Success*	0	0.87	138216	59
	MetaPlacer	0.56	2	*Failure*	-	-	-	-
ibm02-easy	QPlace	1.59	6	*Success*	0	2.26	305559	45
	Dragon	1.54	40	*Success*	0	2.09	301759	96
	MetaPlacer	1.55	5	*Finished*	13	2.23	308476	97
ibm02-hard	QPlace	1.57	6	*Success*	0	2.27	316706	110
	Dragon	1.44	63	*Success*	0	2.01	299451	89
	MetaPlacer	1.52	5	*Failure*	-	-	-	-
ibm07-easy	QPlace	3.79	11	*Success*	0	4.96	572734	91
	Dragon	3.55	42	*Success*	0	4.33	551659	79
	MetaPlacer	3.73	11	*Failure*	-	-	-	-
ibm07-hard	QPlace	3.66	12	*Finished*	3	4.92	621698	323
	Dragon	3.32	43	*Success*	0	4.29	573506	172
	MetaPlacer	3.60	11	*Failure*	-	-	-	-
ibm08-easy	QPlace	3.97	14	*Success*	0	5.29	706225	94
	Dragon	3.66	104	*Success*	0	4.70	663061	55
	MetaPlacer	3.94	11	*Finished*	73	5.32	750222	950
ibm08-hard	QPlace	3.78	14	*Success*	0	5.06	734843	244
	Dragon	3.41	112	*Success*	0	4.40	671800	80
	MetaPlacer	3.77	11	*Failure*	-	-	-	-
ibm09-easy	QPlace	3.45	13	*Success*	0	4.08	566603	37
	Dragon	3.10	85	*Success*	0	3.63	557879	36
	MetaPlacer	3.18	13	*Success*	0	3.65	551005	41
ibm09-hard	QPlace	3.25	12	*Success*	0	3.88	580501	55
	Dragon	3.07	80	*Success*	0	3.61	565941	54
	MetaPlacer	3.23	13	*Success*	0	3.74	570913	65
ibm10-easy	QPlace	6.47	19	*Success*	0	7.87	908325	105
	Dragon	6.00	125	*Success*	0	7.10	879314	100
	MetaPlacer	6.26	16	*Finished*	5	7.56	942912	428
ibm10-hard	QPlace	6.28	19	*Success*	0	7.61	925169	132
	Dragon	5.97	122	*Success*	0	7.06	900879	101
	MetaPlacer	6.35	16	*Failure*	-	-	-	-
ibm11-easy	QPlace	5.15	16	*Success*	0	6.21	742832	71
	Dragon	4.78	90	*Success*	0	5.43	723892	49
	MetaPlacer	4.99	16	*Success*	0	5.75	738919	88
ibm11-hard	QPlace	4.97	16	*Success*	0	6.00	777997	104
	Dragon	4.55	90	*Success*	0	5.27	738743	79
	MetaPlacer	4.99	16	*Success*	0	5.92	769981	157
ibm12-easy	QPlace	9.31	22	*Success*	0	11.69	1156874	247
	Dragon	8.54	152	*Success*	0	10.38	1099080	186
	MetaPlacer	8.65	20	*Finished*	86	10.88	1179568	1193
ibm12-hard	QPlace	8.53	22	*Finished*	39	10.87	1204868	588
	Dragon	8.46	154	*Finished*	1	10.72	1168918	572
	MetaPlacer	8.35	20	*Failure*	-	-	-	-

Table 3: Comparison between Cadence QPlace, our placer (Dragon), and MetaPlacer. Both bounding box wirelength and routed wirelength are in meters. Runtime for QPlace is in minutes on Sun Ultra10 workstation with 400MHz CPU. Runtime for our placer and MetaPlacer is in minutes on Pentium machine with 733MHz CPU. Runtime for each routing is in minutes on the same Sun workstation. Routing results include successful, finished (with violation) and failure (because of too many violations or time out).