Hindawi Publishing Corporation
VLSI Design
Volume 2008, Article ID 517919, 10 pages
doi:10.1155/2008/517919

Research Article

Fine Control of Local Whitespace in Placement

Jarrod A. Roy,[1] David A. Papa,[1,2] and Igor L. Markov[1]

[1] *Department of EECS, University of Michigan, 2260 Hayward Avenue, Ann Arbor, MI 48109-2121, USA*
[2] *IBM Austin Research Lab, IBM Corporation, 11501 Burnet Road, Austin, TX 78758, USA*

Correspondence should be addressed to Jarrod A. Roy, royj@eecs.umich.edu

Received 1 November 2007; Revised 16 May 2008; Accepted 4 August 2008

Recommended by Spyros Tragoudas

In modern design methodologies, a large fraction of chip area during placement is left unused by standard cells and allocated as "whitespace." This is done for a variety of reasons including the need for subsequent buffer insertion, as a means to ensure routability, signal integrity, and low coupling capacitance between wires, and to improve yield through DFM optimizations. To this end, layout constraints often require a certain minimum fraction of whitespace in each region of the chip. Our work introduces several techniques for allocation of whitespace in global, detail, and incremental placement. Our experiments show how to efficiently improve wirelength by reallocating whitespace in legal placements at the large scale. Additionally, for the first time in the literature, we empirically demonstrate highprecision control of whitespace in designs with macros and obstacles. Our techniques consistently improve the quality of whitespace allocation of top-down as well as analytical placement methods and achieve low penalties on designs from the ISPD 2006 placement contest with minimal interconnect increase.

1. INTRODUCTION

At the 65 nm technology node and below, many systematic manufacturing problems that arise can only be effectively mitigated in the physical design portion of the computer-aided design (CAD) flow [1, 2]. Issues such as parasitics variability induced by chemical mechanical polishing (CMP) yield loss due to increased shorts, via failures, forbidden pitches, and forbidden polygonal shapes greatly affect yield. Techniques to handle these problems are known collectively as design for manufacturing (DFM) and are important to routing tools targeting 65 nm designs [3].

An important factor in many DFM issues is design density, determined by local whitespace (also known as free space). Wire density is critical as too much wiring congestion has notable performance impact due to (i) longer wires resulting from routing detours, (ii) increased crosstalk which reduces reliability and degrades timing, and (iii) increased via counts which lengthen signal propagation time and can decrease yield [4]. Conversely, too little wiring density will increase the likelihood that CMP will erode parts of wires, greatly increasing their resistance or leaving their connection open entirely [3]. While metal fill can mitigate harmful CMP effects, it does so at the cost of negative performance impact of additional crosstalk. Achieving the right balance of wire density through whitespace management avoids the performance impact of too much congestion while reducing the need for metal fill.

In addition to wire density concerns, poor whitespace allocation during physical synthesis can increase total cell area due to buffer insertion and gate sizing [5]. One must reserve space locally to accommodate these operations to meet timing constraints, but reserving too much is wasteful [6]. In particular, a design that is placed too densely may have increased wirelength due to routing detours, and may be unable to close timing by inserting buffers into full regions of the chip. However, a design that is placed too sparsely will also have increased wirelength and suffers both timing degradation and increased power consumption from more buffers. Therefore, accurate modeling of whitespace and precise cell density control are important concerns during the global and detail placement phases of a physical synthesis flow.

The literature includes several techniques to optimize whitespace distributions [6–9]. A natural scheme for managing whitespace in top-down placement, uniform whitespace allocation, was rigorously analyzed in [8]. The authors derived expressions for the tolerance to be given to a

```
    ▷ Input: placement bin B, whitespace target targetWS
    ▷ Output: partitioned child bins
(1) SetTentativeCutline(B)
(2) if (BinWhitespace(B) > targetWS and targetWS ≥ SafeWS)
(3)   then SetPartToleranceSafe(B, targetWS)
(4)     CallPartitioner(B)
(5)     ShiftCutlineToMaintainTargetWS(B, targetWS)
(6)   else if (BinWhitespace(B) > targetWS
                and targetWS ≥ minLocalWS)
(7)       then SetPartToleranceMinLocal(B, targetWS)
(8)         CallPartitioner(B)
(9)         ShiftCutlineToMaintainTargetWS(B, targetWS)
(10)      else SetPartToleranceUniform(B)
(11)        CallPartitioner(B)
(12)        ShiftCutlineToEqualizeWS(B)
(13) FinalizeCutline(B)
(14) CreateChildBins(B)
```

Algorithm 1: Allocating whitespace in top-down placement to satisfy density constraints using uniform, minimum local, and safe whitespace allocation.

min-cut partitioner such that whitespace would be allocated as uniformly as possible given the discrete nature of the problem.

A technique for nonuniform whitespace allocation presented in [7] adds disconnected standard cells to the design before placement using uniform whitespace allocation and removes them immediately after. Care must be taken not to add too many cells to the design which complicates the work of many placement algorithms, increasing interconnect length or leading to overlapping circuit modules [10]. In [6], analytical methods are used to allocate whitespace in sparse designs for min-cut placement. Before calls to partitioning, the design is placed quickly with an analytical algorithm. Cell area that is placed on either side of a proposed cutline is used as an area target for min-cut partitioning. After floorplanning, [9] provides min-cost network-flow formulations to optimally redistribute floorplan whitespace to reduce interconnect length.

There are relatively few techniques in the literature for respecting whitespace constraints imposed by a designer while still optimizing interconnect. Such constraints are helpful as they are typically imposed to improve routability, allow for effective buffer insertion, and so forth. In many cases, these constraints come in the form of cell density restrictions. One trivial way to ensure sparser cell densities in a placement is by artificially increasing cell sizes before placement (*bloating*) and shrinking them back to normal size afterward [11]. For the bloating to be effective, the majority of the original whitespace of the design must be consumed. This reduces the amount of whitespace available to the placer which is undesirable for reasons stated above. Bloating also makes density control in detail placement more difficult as standard cells can only be bloated in discrete steps. Widening a standard cell by a single cell site often increases cell width by 20% or more whereas density control requires much finer precision.

In this paper, we propose several methods for top-down whitespace allocation to satisfy whitespace constraints. Our key contributions are the following.

(i) We introduce three user-controlled whitespace allocation techniques which allow for fine control of whitespace allocation in top-down global placement.

(ii) We outline several detail placement techniques which enforce density constraints while simultaneously improving interconnect length.

(iii) We quantify the difficulty in satisfying density constraints and show why so many of the best solutions to the ISPD 2006 contest benchmarks [12] did not satisfy these constraints.

This paper is organized as follows. Whitespace terminology and previous work are covered in Section 2. Section 3 describes top-down whitespace allocation techniques and illustrates using them to satisfy density constraints in global placement. Whitespace allocation in detail placement is discussed in Section 4. Section 5 provides empirical evaluation of our whitespace allocation techniques, and we conclude in Section 6.

2. BACKGROUND AND PREVIOUS WORK

Below, we review fundamental whitespace terminology and techniques for top-down placement as well as existing work on allocating whitespace for goals such as improved mixed-size placement and routability. Our choice of the top-down placement framework is motivated by the fact that a top-down technique has been used to improve the routability of analytical placements [13], and the top-down placer Capo is currently producing the smallest via counts for the IBMv2 and IWLS 2005 benchmark suites [14].

```
    ▷ Input: initial placement P, grid of density targets G
    ▷ Output: optimized placement P
(1) bestImprovement ← 0, cellToMove ← −1
(2) foreach cell C
(3)   do Search G for a legal location of C that best
         improves density violations and does not increase HPWL
         by more than HpwlLimit% (break ties using HPWL)
(4)      Store the best location of C
(5)      if (density improvement of C > bestImprovement)
(6)        then bestImprovement ← density improvment of C
(7)             cellToMove ← C
(8) while (cellToMove ≠ −1)
(9)   do Move cellToMove to its best location
(10)     Fix cellToMove
(11)     bestImprovement ← 0, newCellToMove ← −1
(12)     foreach cell C≠ cellToMove
(13)       do if (best location of C overlaps with cellToMove)
(14)            then Search G for the best location of C
(15)     else if (C is connected to cellToMove)
(16)            then Recalculate the density improvement of moving C
                  to its best location
(17)              if (density improvement of C ≤ 0)
(18)                 then Search G for the best location of C
(19)       if (density improvement of C > bestImprovement)
(20)       then bestImprovement ← density improvment of C
(21)            newCellToMove ← C
(22)     cellToMove ← newCellToMove
(23)     if (density improvement of cellToMove < improvementCutoff)
(24)       then cellToMove ← −1
(25)     if (moving cellToMove to its best location increases HPWL
           beyond HpwlCap)
(26)       then cellToMove ← −1
```

Algorithm 2: Greedy cell movement to reduce density violations while also taking into account HPWL.

2.1. Whitespace fundamentals

Optimal uniform whitespace allocation techniques for top-down bisection-based placement were derived in [8]. Let a placement bin which is going to be partitioned have *site area* S, *cell area* C, *absolute whitespace* $W = \max\{S - C, 0\}$, and *relative whitespace* $w = W/S$. A bi-partitioning divides the bin into two-child bins with *site areas* S_0 and S_1 such that $S_0 + S_1 = S$ and *cell areas* C_0 and C_1 such that $C_0 + C_1 = C$. A partitioner is given cell area targets T_0 and T_1 as well as a tolerance τ for a particular bi-partitioning instance. In many cases of bi-partitioning, $T_0 = T_1 = C/2$, but this is not always true [6]. τ defines the maximum percentage by which C_0 and C_1 are allowed to differ from T_0 and T_1, respectively.

The work in [8] bases its whitespace allocation techniques on *whitespace deterioration*: the phenomenon that discreteness in partitioning and placement does not allow for exact uniform whitespace distribution. The whitespace deterioration for a bi-partitioning is the largest α, such that each child bin has at least αw relative whitespace. Assuming nonzero relative whitespace in the placement bin, α should be restricted such that $0 \leq \alpha \leq 1$ [8]. The authors note that $\alpha = 1$ may be overly restrictive in practice because it induces zero tolerance on the partitioning instance but $\alpha = 0$ may not be restrictive enough as it allows for child bins with zero whitespace, which can improve wirelength but impair routability [8].

For a given block, feasible ranges for partition capacities are uniquely determined by α. The partitioning tolerance τ for splitting a block with relative whitespace w is $(1 - \alpha)w/(1 - w)$ [8]. The challenge is to determine a proper value for α. First assume that a bin is to be partitioned horizontally n times more during the placement process. n can be calculated as $\lceil \log_2 R \rceil$, where R is the number of rows in the placement bin [8]. Assuming end-case bins have $\alpha = 0$ since they are not further partitioned, $\overline{w}$, the relative whitespace of an end-case bin, is determined to be $\overline{\tau}/(\overline{\tau} + 1)$, where $\overline{\tau}$ is the tolerance of partitioning in the end-case bin [8].

Making the practical assumption that τ remains constant over all partitionings, we find $\tau = \sqrt[n]{(1 - \overline{w})/(1 - w)} - 1$ [8]. $\overline{w}$ can be eliminated from the equation for τ and a closed form for α using only w, and n is derived to be $\alpha = (\sqrt[n+1]{1 - w} - (1 - w))/w\sqrt[n+1]{1 - w}$ [8].

2.2. Whitespace in mixed-size placement

Industrial floorplacement problems are increasingly difficult due to factors such as an increasing number of movable modules and a wide variation of module sizes. To address these issues, SCalable Advanced Macro Placement Improvements (SCAMPIs) [15] modify the top-down partitioning flow to selectively place large macros, while smaller macros are clustered into soft modules that will be placed later. The robustness of the flow is also improved by employing fast *look-ahead* simulated annealing on large macros of newly created bins. This allows early detection of bins difficult to floorplan and alerts the placer to backtrack and seek a different partitioning solution.

There is also insufficient cohesion for whitespace allocation between top-down methods and macroplacement algorithms. For example, a partitioner may approximate the area required by a set of macros and incorrectly allocate whitespace. While uniform whitespace distribution is sufficient in many cases, the authors of SCAMPI observe that in certain cases, one of two-child bins may require more whitespace than its sibling. By redistributing whitespace from easy-to-pack child bins to those hard to pack, a floorplacer becomes more robust and runtime improves [15].

2.3. Whitespace allocation for routability

It is well known that a placement with small HPWL may be unroutable due to uneven routing demand and ensuing wiring congestion. For this reason, modern placers must explicitly account for routing congestion in order to produce routable placements. In [16], congestion maps are built after global placement, and annealing moves are applied to minimize a congestion metric. Another technique known as WSA [17] is applied after detail placement. WSA uses congestion maps to identify areas with high congestion and injects whitespace into these areas in a top-down fashion. After whitespace allocation, cells typically overlap each other, and legalization is required. After legalization, window-based detail placement techniques are applied to reduce wirelength that was increased during whitespace allocation and legalization. ROOSTER [14] builds congestion maps during top-down placement. Based on the congestion estimates, whitespace is allocated preferentially to areas of high congestion through cutline shifting. Unlike WSA, no additional legalization is necessary in ROOSTER as whitespace allocation is done before detail placement. Cell bloating [11] and cell spreading [17] have been used to tie whitespace to specific cells, rather than to fixed regions as in techniques based on congestion maps. Bloating reduces the amount of whitespace available to the placer which can impair solution quality significantly [10]. Bloating also makes fine density control in detail placement more difficult as standard cells can only be bloated in discrete steps that correspond to site widths and row heights. Widening a standard cell by one cell site often increases cell width by more than 20% whereas density control requires precision on the order of single percents.

3. WHITESPACE ALLOCATION

Top-down min-cut placement proceeds by successively dividing *placement bins*, the first of which contains the entire core area and all movable objects, until the bins are small enough to be optimally placed. Whitespace allocation is done per placement bin and in this section we describe three techniques: *uniform*, *minimum local*, and *safe* whitespace allocation. Any one of these options can be chosen per bin based on the bin's whitespace and user-configurable options. Pseudocode in Algorithm 1 shows how these three techniques are used together to satisfy whitespace constraints.

3.1. Uniform whitespace

If a bin has a user-defined "small" amount of whitespace or less, partitioning attempts to divide the cell area approximately in half, within a given tolerance. The appropriate partitioning tolerance is chosen based on whitespace deterioration and is calculated as described in Section II above. After a partitionnement (i.e., a partitioning solution) is computed, the geometric cut-line for the bin is positioned so that each side of the cutline has an equal percentage of whitespace. As tolerance is calculated assuming a fixed cutline, the cutline is shifted to make whitespace more uniform. Such whitespace allocation generally produces routable placements, at the cost of increased wirelength.

3.2. Minimum local whitespace

If a bin has more than a user-defined minimum local whitespace (`minLocalWS`), partitioning will define a tentative cutline that divides the bin's placement area in half. Partitioning targets an equal division of cell area, but is given more freedom to deviate from its target. Tolerance is computed so that with whitespace deterioration, each descendant bin of the current bin will have at least `minLocalWS`.

The assumption that the whitespace deterioration α in end-case bins is 0 made in [8] and presented in Section 2 no longer applies, so the calculation of α must change. Since we want all child bins of the current bin to have `minLocalWS` relative whitespace, in particular end case bins must have at least `minLocalWS` and thus we may set $\overline{w}$ = `minLocalWS`, instead of a function of τ. With the assumption that τ remains constant, τ is calculated as a function of $\overline{w}$, w, and n as shown in Section 2. Knowing τ, $\alpha = (\tau + 1) - \tau/w$.

After a partitionnement is calculated, the cutline is shifted to ensure that `minLocalWS` is preserved on both sides of the cutline. If the minimum local whitespace is chosen to be small, the placer can produce tightly packed placements which greatly improves wirelength.

3.3. Safe whitespace

The last whitespace allocation mode is designed for bins with "large" quantities of whitespace. In safe whitespace allocation, as with minimum local whitespace allocation, a tentative geometric cutline of the bin is chosen, and

the target of partitioning is an equal bisection of the cell area. The difference in safe whitespace allocation mode is that the partitioning tolerance is much higher. Essentially, any partitioning solution that leaves at least `safeWS` on either side of the cutline is considered legal. This allows for very tight packing and reduces wirelength, but is not recommended for congestion-driven placement.

4. WHITESPACE IN DETAIL PLACEMENT

Placement tools use several techniques to further reduce HPWL after global placement, such as the sliding window optimizer RowIroning [18], but these techniques usually do not respect density constraints. To have finer control of whitespace than the sliding-window scheme, we present two-detail placement techniques that focus on whitespace allocation in addition to improvement of HPWL: a greedy cell-movement scheme and optimal whitespace allocation that preserves relative cell ordering by solving min-cost network-flow problems [9, 19].

4.1. Greedy cell movement

A gridded greedy movement technique can improve both wirelength and whitespace distribution. Pseudocode for our technique is shown in Algorithm 2. An arbitrary grid is imposed on the placement region to analyze local placement density. Density targets are set for each of the grid regions individually and can be nonuniform. For standard cells that are in regions with density violations, location candidates are found in areas of lower density violation. Candidate moves are ranked by how well they alleviate violations as well as how they affect wirelength. We allow moves that increase HPWL, but only a fixed amount per move. Moves are made until a threshold of density improvement or a limit on increased HPWL is reached.

A similar greedy movement technique can reclaim HPWL while leaving whitespace distribution unchanged. In this technique, pairs and triples of cells of approximately the same size are examined. The number of pairs and triples of cells in any modern design is intractable, so to keep runtime feasible our technique only considers pairs and triples of cells that are directly connected to each other by 2-pin nets. After pairs and triples are collected, the HPWL gain is evaluated for swapping pairs of cells and the five nontrivial permutations of triples of cells. As the cells are of approximately the same size, no overlap is produced by these swapping moves and whitespace distribution is largely unaffected. These cell-swapping moves are applied until a HPWL improvement threshold is reached.

4.2. Optimal whitespace redistribution

Optimal whitespace allocation in row-based placement [19] and floorplanning [9] given a fixed cell-ordering has previously been described in the literature. Unfortunately, [19] only considered optimal whitespace allocation for the x-direction of a single row of a placement at a time while [9] was limited to relatively small floorplanning solutions

Let the set of cells and macros be denoted
$$C = c_1, c_2, \ldots, c_i,$$
and let the set of nets be denoted
$$N = n_1, n_2, \ldots, n_j.$$
Using the variables:
$$x_k \quad \text{for } 1 \le k \le i,$$
$$L_k^x, U_k^x \quad \text{for } 1 \le k \le j,$$
and constants:
$$left\_boundary,\ right\_boundary,$$
$$placement(k).x \text{ for } 1 \le k \le i,$$
$$width(k) \quad \text{for } 1 \le k \le i,$$
we minimize
$$\sum_{k=1}^{j} (U_k^x - L_k^x)$$
subject to constraints:
$$L_n^x \le x_k \le U_n^x \quad \text{where cell } k \text{ is on net } n,$$
$$left\_boundary \le x_k \quad \text{for } 1 \le k \le i,$$
$$x_k + width(k) \le right\_boundary \quad \text{for } 1 \le k \le i,$$
$$x_k = placement(k).x \text{ where cell } k \text{ is fixed},$$
$$x_k + width(k) \le x_l \quad \text{where cell } k \text{ is directly to the left of cell } l$$

Algorithm 3: Linear programming formulation for the horizontal direction to optimize HPWL of an existing placement. Further simplification is possible for 2- and 3-pin nets.

generated using sequence-pairs. We extract the best of these techniques and generate min-cost network-flow problems for generic floorplacement instances whose solutions optimally redistribute whitespace and snap cells to row/site boundaries.

Our technique builds upon the well-known linear programming formulations used, for example, in [9, 20] in that we impose linear constraints for movable objects based on their relative positions with respect to core boundaries and other movable objects. We include additional linear inequalities to account for fixed obstacles and region constraints. One major difference from previous work is that we guarantee that the x and y locations found align to legal sites and rows, as explained below.

We handle reallocation of whitespace separately for the horizontal and vertical directions and preserve local relative ordering of movables in each direction. In other words, movable objects may not jump over each other or any fixed obstacles when whitespace is being reallocated. Unlike global-placement [20], we start with legal or nearly-legal locations. This simplifies our selection of relative constraints to include into the LP formulation as follows. In the horizontal case, we examine each row individually. For each cell or macro that intersects the row, we determine its immediate neighbors to the left and to the right (those objects with which the current object could feasibly overlap if it would slide to the left or right). These neighbors include movable objects, row, or region boundaries as well as fixed obstacles. After the neighborhood relations are determined, we constrain an object to lie between its left- and right-hand neighbors. Construction of constraints for the vertical case is

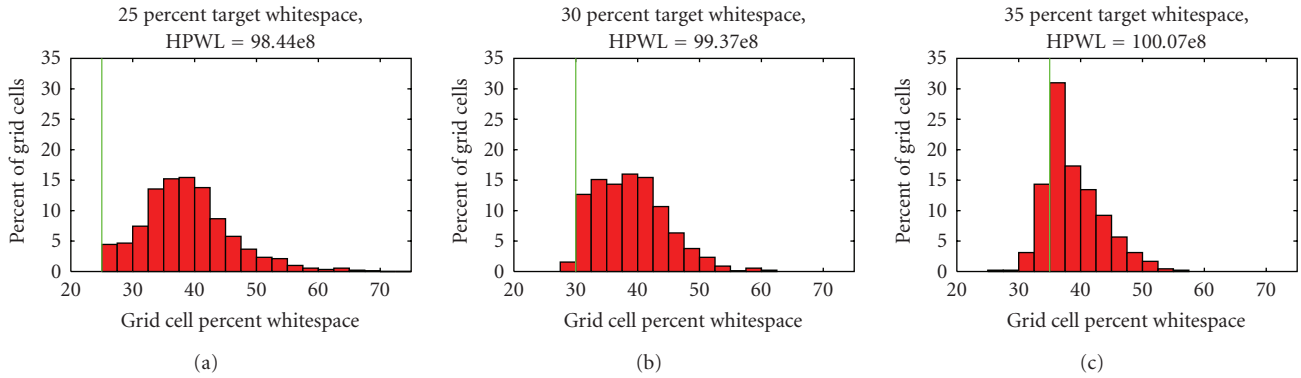

FIGURE 1: Controlling whitespace distribution on the `ethernet` benchmark from the IWLS 2005 benchmark suite [22], which has approximately 38% whitespace. We divide the placement area into a regular grid and report whitespace distribution across grid cells when targeting (a) 25%, (b) 30%, and (c) 35% minimum local whitespace. As the minimum whitespace requested approaches the amount of total whitespace, the constraint becomes more difficult to satisfy, but our techniques are successful in producing solutions that are legal or nearly legal for the vast majority of grid cells.

analogous where rows are replaced with columns, and site width is replaced by row height. Lastly, to preserve global whitespace allocation characteristics, we add constraints to limit the amount of movement of any individual cell from its initial position. Unlike the formulation from [20], ours guarantees an overlap-free placement and needs to be solved only once. In contrast with [9], we include only several constraints per movable object rather than a quadratic number of constraints read from a sequence-pair. This significantly improves scalability and allows one to pack more tightly.

In addition to the constraints above, we minimize HPWL. This is done by adding L^x, U^x, L^y, U^y variables for each net, and the terms $(U^x - L^x)$ and $(U^y - L^y)$ to the objective function. The LP formulation for the horizontal case is enumerated in Algorithm 3. To solve the entire LP efficiently, we dualize it as in [9] and cast the dual as a min-cost max-flow instance. The latter is solved using the scaling push-relabeling algorithm of Goldberg [21]. Nets from the original LP formulation become directed edges with unit capacity and zero cost in the dualized flow instance, and distance constraints become directed edges with costs and unlimited capacity. Goldberg's implementation of push-relabeling in C uses integer variables for both costs and capacities. Thus, the algorithm naturally produces integer solutions when the input is encoded in integers. We use this integrality to produce solutions that are row- and site-aligned—we scale coordinates so that integer x values correspond to legal sites and integer y values correspond to standard-cell rows. Thus, our solutions need no further legalization.

Empirically, this technique is extremely fast and provides nontrivial interconnect length improvement. Entire placements of up to 50 000 cells can have their whitespace reallocated in 60 seconds or less. We have found that 50 000 cells is a good tradeoff between quality and runtime, so we break the placement area into a regular grid with a target of 50 000 cells per grid cell and allocate whitespace in each region separately. We generally see interconnect length improvement of 2-3% with a runtime cost less than 10% of placement runtime.

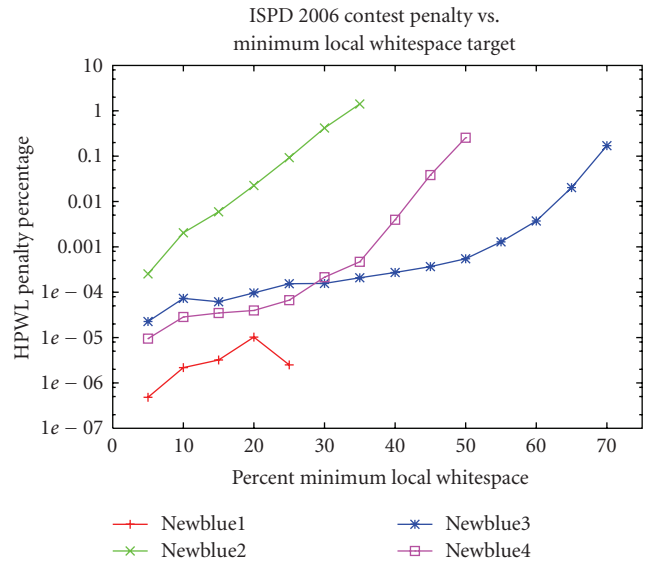

FIGURE 2: ISPD 2006 placement contest penalty for requested amounts of minimum local whitespace. We test on benchmarks from the ISPD 2006 placement contest suite [12]. These benchmarks have 29%, 38%, 74%, and 54% whitespace, respectively. Usually the penalty is very small when using our techniques (always less than 1.5%), but the penalty grows significantly as the requested whitespace approaches the amount of whitespace available in the design.

5. EMPIRICAL RESULTS

We have implemented these whitespace allocation techniques in the open-source academic placer Capo 10.5 [24]. Our use of Capo as an implementation platform is justified by Capo's competitive results on difficult mixed-size instances [15] and all routability-driven placement benchmarks reported

Table 1: Real location of whitespace in mPL6 [23] placements of selected ISPD 2006 contest benchmarks [12]. Local whitespace targets are the same as from the ISPD 2006 placement contest. Density violations are measured as the percentage of total cell area that violates density constraints. Using Capo 10.5 in ECO-system mode [13] in combination with our whitespace allocation techniques, we are able to significantly reduce the density violations of the mPL6 placements.

Benchmark	Local WS target	HPWL (e6)		Density violation	
		Before	After	Before	After
newblue1	20%	66.61	74.37	1.039%	0.065%
newblue2	10%	199.05	223.87	3.701%	0.964%
newblue3	20%	283.40	297.97	2.813%	0.126%
newblue4	50%	293.22	304.99	7.238%	4.105%

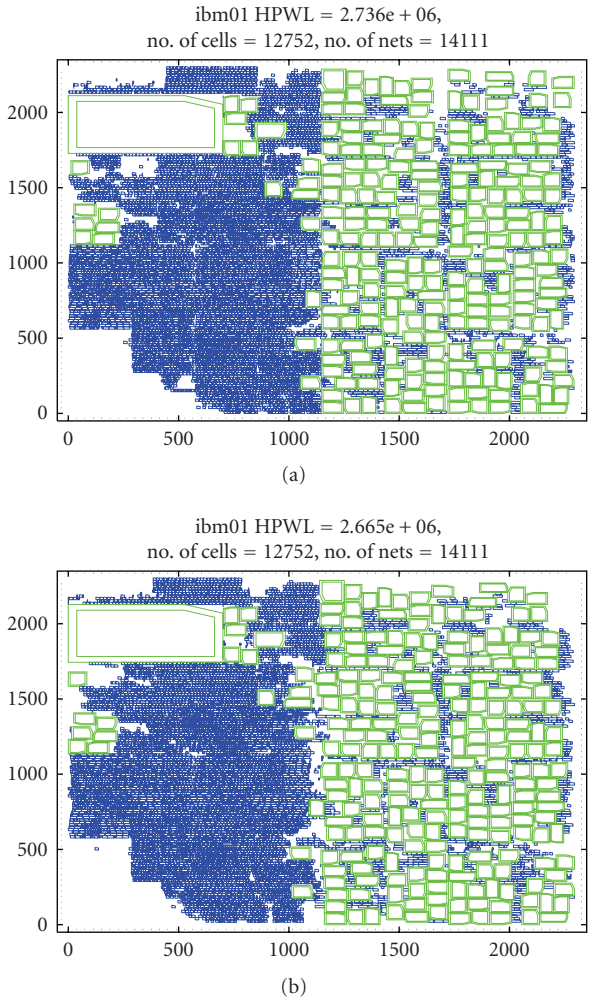

Figure 3: ICCAD 2004 IBM-MSwPins benchmark [27] ibm01 before (left) and after (right) optimal whitespace allocation via network flows. The HPWL improvement for this placement is 2.61% and takes only 10 CPU seconds.

in the literature. In particular, Capo's routed solutions have best published via counts [14], which is very important for DFM and yield. Vias also significantly impact timing and may complicate routing by blocking routing tracks [4]. In this section, we evaluate our techniques in the contexts of satisfying density constraints and optimizing HPWL on a wide variety of publicly available benchmarks.

Table 2: Characteristics of selected benchmarks from the IWLS 2005 suite [22]. Grid size represents the size of the grid used for greedy cell movement on these benchmarks.

Benchmark	# Movables	Grid size
aes_core	20795	14×14
ethernet	46771	30×30
mem_ctrl	11440	12×12
pci_bridge32	16816	17×17
usb_funct	12808	14×14
vga_lcd	124031	41×41

5.1. Whitespace reallocation

We combine our whitespace allocation techniques with the ECO-system [13] mode of Capo 10.5 to reallocate whitespace in ISPD 2006 contest solutions from the mPL6 placer [23]. mPL6 uses a multilevel analytical technique for global placement with cell bloating to help meeting the target densities [25] and the XDP detail placer [26] which legalizes and applies sliding window techniques to recover wirelength. At the ISPD 2006 placement contest, mPL6 produced the best solutions when not considering runtime, but as shown in Table 1 the solutions did not satisfy the density constraints imposed by the competition. These density violations can be significantly improved using our technique, but only at the cost of significantly increased wirelength. It is important to note that the coefficients in the ISPD 2006 penalty formula were chosen rather arbitrarily, while the effective cost of violations greatly depends on the types of problems caused by violations, such as increased crosstalk noise and need for DFM fix ups. In the smaller benchmarks, newblue1 and newblue2, the cost in HPWL is approximately 12%. On newblue3 and newblue4, the increase in HPWL is much lower at 5% and 4%, respectively. This shows, especially on the larger benchmarks, that density violations can be improved dramatically with a reasonable increase to HPWL. These reallocated placements outperform all but one placer on one benchmark from the ISPD 2006 contest (only Dragon's placement of newblue4 has a lower density penalty) and have extremely competitive HPWL.

Table 3: Correction of local density violations by greedy cell movement techniques. Benchmarks are selected from the IWLS 2005 benchmark suite and each has 38% total whitespace [22]. Density violations are measured as the percentage of total cell area violates density constraints. Greedy cell movement corrects all density violations when requested local whitespace is 25% or less and in many cases improves HPWL as well.

Benchmark	Local WS target	Density violation		HPWL impact	Runtime (s)
		Before	After		
aes_core	25%	0.042%	**0**%	−**1.699**%	43
aes_core	30%	0.208%	0.017%	+0.513%	4
aes_core	35%	0.572%	0.246%	+0.530%	8
ethernet	20%	0.002%	**0**%	−**0.534**%	163
ethernet	25%	0.036%	**0**%	+0.501%	174
ethernet	30%	0.113%	0.015%	+0.503%	111
ethernet	35%	0.892%	0.459%	+0.522%	128
mem_ctrl	30%	0.008%	**0**%	−**0.023**%	3
mem_ctrl	35%	0.586%	0.110%	+0.518%	4
pci_bridge32	20%	0.002%	**0**%	−**1.167**%	25
pci_bridge32	25%	0.061%	**0**%	−**1.182**%	31
pci_bridge32	30%	0.010%	0.003%	+0.530%	11
pci_bridge32	35%	0.823%	0.331%	+0.506%	17
usb_funct	30%	0.030%	**0**%	−**0.547**%	16
usb_funct	35%	0.356%	0.068%	+0.578%	6
vga_lcd	20%	0.0002%	**0**%	−**0.440**%	622
vga_lcd	25%	0.045%	**0**%	+0.132%	743
vga_lcd	30%	0.264%	0.101%	+0.500%	585
vga_lcd	35%	1.288%	0.758%	+0.500%	740

Table 4: HPWL improvement due to flow-based whitespace redistribution on the ICCAD 2004 IBM-MSwPins mixed-size benchmarks [27]. On average, the flows are able to reallocate whitespace and improve HPWL by nearly 3% while scaling well with increasing quantities of movable objects.

Benchmark	# Movables	Runtime (s)	Improvement
ibm01	12506	11	2.41%
ibm02	19342	23	2.60%
ibm03	22853	27	3.81%
ibm04	27220	38	2.96%
ibm05	28146	46	2.27%
ibm06	32332	44	2.87%
ibm07	45639	69	3.17%
ibm08	51023	64	2.82%
ibm09	53110	69	4.23%
ibm10	68685	129	1.91%
ibm11	70152	102	4.42%
ibm12	70439	135	1.87%
ibm13	83709	127	3.83%
ibm14	147088	262	2.64%
ibm15	161187	342	3.67%
ibm16	182980	404	2.62%
ibm17	184752	446	2.54%
ibm18	210341	338	2.47%
Average			2.86%

5.2. Density constraint satisfaction

We implemented all of our proposed whitespace allocation techniques in the Capo 10.5 framework and test uniform and nonuniform whitespace allocation on the ISPD 2005 contest benchmark adaptec1 (57.34% utilization) with 60% and 90% target whitespace densities. The HPWL for the uniform and nonuniform placements is 10.69e7 and 9.03e7, respectively. Uniform whitespace produces almost no violations when the target is 90% and relatively few when the target is 60%. The nonuniform placement has more violations when compared to the uniform placement, especially when the target is 60%, but remains largely legal with a 90% target density. This shows that uniform whitespace allocation is appropriate when target density is near the total amount of whitespace in a design, otherwise nonuniform allocation can be used to improve wirelength.

In Figure 1, we show histograms of grid cell densities across the ethernet benchmark [22] when given local whitespace constraints of (a) 25%, (b) 30%, and (c) 35%. The ethernet design has 38% total whitespace, and our techniques are able to achieve completely legal solutions when 25% local whitespace is requested, but the constraints become more difficult to satisfy at 30% and 35%. Despite the difficulty, our techniques are successful in producing solutions that are legal or nearly-legal for the vast majority of grid cells.

The inherent difficulty in satisfying minimum whitespace constraints as the requested whitespace approaches the total amount of whitespace in the design is also apparent from Figure 2. Here, we place selected benchmarks from the ISPD 2006 placement contest with a wide range of requested local whitespace values. We use our whitespace allocation methods to match the requested amount of local whitespace for each of the instances and evaluate the legality of our solutions with the ISPD 2006 placement contest density penalty function [12]. The penalty is calculated based on the total amount of density constraint violations in the placement. Our solutions generally have very small penalties (always less than 1.5%) suggesting that our techniques satisfy density constraints well. Note how the density penalty grows more quickly as the amount of requested whitespace approaches the total amount of whitespace in the design.

5.3. Greedy cell movement

Table 3 shows the effectiveness of greedy movement techniques in removing density violations. Benchmarks are selected from the IWLS 2005 benchmark suite and each has 38% total whitespace [22]. Size characteristics of these benchmarks are shown in Table 2. Density violations are reported as the percentage of total cell area that violates density constraints. Greedy cell movement corrects all density violations when requested local whitespace is 25% or less and in many cases improves HPWL as well. As the requested local whitespace approaches the total whitespace, greedy movement is not able to remove all of the density violations without making HPWL increase more than 0.5%. With a higher limit on HPWL increase, greedy movement can apply more moves and further reduce density violations.

5.4. Flow-based whitespace redistribution

We test optimal whitespace redistribution based on min-cost network flows on the ICCAD 2004 IBM-MSwPins benchmark suite [27]. Table 4 gives detailed runtime and HPWL improvement results for each of the IBM-MSwPins benchmarks. We do not provide overflow statistics on these designs since our flow-based whitespace redistribution maintains global whitespace characteristics. On average, HPWL on these benchmarks is improved by 2.86% and runtimes scale nearly linearly with benchmark size. Figure 3 depicts a placement of the ibm01 mixed-size design before (left) and after (right) whitespace optimization with flows.

6. CONCLUSIONS

In this paper, we have introduced methods for satisfying whitespace constraints in top-down placement while also optimizing interconnect. These constraints take the form of cell density limits on a placement. A followup to the Rooster work on routability-driven placement [14] has found that cell density limits can be extremely useful for promoting routability, decreasing metal fill, improving yield, and so forth [28]. Our techniques consistently improve the quality of whitespace allocation of top-down as well as analytical placement methods and achieve low penalties on designs from the ISPD 2006 placement contest with minimal interconnect increase.

All of our techniques for whitespace allocation in both global and detail placement are implemented in the Capo 10.5 academic min-cut placer [14, 15, 24]. Capo 10.5 is freely available for all uses as part of the UMpack (http://vlsicad.eecs.umich.edu/BK/PDtools/).

ACKNOWLEDGMENTS

This work was partially supported by the DARPA/MARCO Gigascale Systems Research Center, the National Science Foundation, Synplicity, Inc., Calif, USA, as well as equipment donations from Intel, Calif, USA.

REFERENCES

[1] P. Azzoni, M. Bertoletti, N. Dragone, F. Fummi, C. Guardiani, and W. Vendraminetto, "Yield-aware placement optimization," in *Proceedings of Design, Automation & Test in Europe (DATE '07)*, pp. 1232–1237, Nice, France, April 2007.

[2] S. Hu and J. Hu, "Pattern sensitive placement for manufacturability," in *Proceedings of International Symposium on Physical Design (ISPD '07)*, pp. 27–34, Austin, Tex, USA, March 2007.

[3] M. Cho, D. Z. Pan, H. Xiang, and R. Puri, "Wire density driven global routing for CMP variation and timing," in *Proceedings of International Conference on Computer-Aided Design (ICCAD '06)*, pp. 487–492, San Jose, Calif, USA, November 2006.

[4] "TSMC: Silicon Success," http://www.tsmc.com/download/enliterature/html-newsletter/September03/InDepth/index.html.

[5] C. J. Alpert, S. K. Karandikar, Z. Li, et al., "Techniques for fast physical synthesis," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 573–599, 2007.

[6] C. J. Alpert, G.-J. Nam, and P. G. Villarrubia, "Effective free space management for cut-based placement via analytical constraint generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1343–1353, 2003.

[7] S. N. Adya, I. L. Markov, and P. G. Villarrubia, "On whitespace and stability in mixed-size placement and physical synthesis," in *Proceedings of International Conference on Computer-Aided Design (ICCAD '03)*, pp. 311–318, San Jose, Calif, USA, November 2003.

[8] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hierarchical whitespace allocation in top-down placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 11, pp. 716–724, 2003.

[9] X. Tang, R. Tian, and M. D. R. Wong, "Optimal redistribution of white space for wire length minimization," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC '05)*, vol. 1, pp. 412–417, Shanghai, China, January 2005.

[10] J. Cong, M. Romesis, and J. R. Shinnerl, "Robust mixed-size placement under tight white space constraints," in *Proceedings of International Conference on Computer-Aided Design (ICCAD '05)*, pp. 165–172, San Jose, Calif, USA, November 2005.

[11] N. Selvakkumaran, P. N. Parakh, and G. Karypis, "Perimeter-degree: a priori metric for directly measuring and homogenizing interconnection complexity in multilevel placement," in *Proceedings of the 5th International Workshop on System-Level Interconnect Prediction (SLIP '03)*, pp. 53–59, Monterey, Calif, USA, April 2003.

[12] G.-J. Nam, "ISPD 2006 placement contest: benchmark suite and results," in *Proceedings of International Symposium on Physical Design (ISPD '06)*, p. 167, San Jose, Calif, USA, April 2006.

[13] J. A. Roy and I. L. Markov, "ECO-system: embracing the change in placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 12, pp. 2173–2185, 2007.

[14] J. A. Roy and I. L. Markov, "Seeing the forest and the trees: steiner wirelength optimization in placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 4, pp. 632–644, 2007.

[15] A. N. Ng, I. L. Markov, R. Aggarwal, and V. Ramachandran, "Solving hard instances of floorplacement," in *Proceedings of International Symposium on Physical Design (ISPD '06)*, pp. 170–177, San Jose, Calif, USA, April 2006.

[16] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "Routability driven white space allocation for fixed-die standard-cell placement," in *Proceedings of International Symposium on Physical Design (ISPD '02)*, pp. 42–47, Del Mar, Calif, USA, April 2002.

[17] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," in *Proceedings of International Conference on Computer-Aided Design (ICCAD '04)*, pp. 394–401, San Jose, Calif, USA, November 2004.

[18] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Optimal partitioners and end-case placers for standard-cell layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 11, pp. 1304–1313, 2000.

[19] U. Brenner and J. Vygen, "Faster optimal single-row placement with fixed ordering," in *Proceedings of Design, Automation & Test in Europe (DATE '00)*, pp. 117–121, Paris, France, March 2000.

[20] S. Reda and A. Chowdhary, "Effective linear programming based placement methods," in *Proceedings of International Symposium on Physical Design (ISPD '06)*, pp. 186–191, San Jose, Calif, USA, April 2006.

[21] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *Journal of Algorithms*, vol. 22, no. 1, pp. 1–29, 1997.

[22] C. Albrecht, "IWLS 2005 Benchmarks," June 2005, http://iwls.org/iwls2005/benchmarks.html.

[23] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie, "mPL6: enhanced multilevel mixed-size placement," in *Proceedings of International Symposium on Physical Design (ISPD '06)*, pp. 212–214, San Jose, Calif, USA, April 2006.

[24] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov, "Min-cut floorplacement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1313–1326, 2006.

[25] J. Cong and G. Luo, "Highly efficient gradient computation for density-constrained analytical placement methods," in *Proceedings of International Symposium on Physical Design (ISPD '08)*, pp. 39–46, Portland, Ore, USA, April 2008.

[26] J. Cong and M. Xie, "A robust detailed placement for mixed-size IC designs," in *Proceedings 11th Asia and South Pacific Design Automation Conference (ASP-DAC '06)*, pp. 188–194, Yokohama, Japan, January 2006.

[27] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov, "Unification of partitioning, placement and floorplanning," in *Proceedings of International Conference on Computer-Aided Design (ICCAD '04)*, pp. 550–557, San Jose, Calif, USA, November 2004.

[28] T.-C. Chen, M. Cho, D. Z. Pan, and Y.-W. Chang, "Metal-density driven placement for CMP variation and routability," in *Proceedings of International Symposium on Physical Design (ISPD '08)*, pp. 31–38, Portland, Ore, USA, April 2008.