Robust Mixed-Size Placement by Recursive Legalized Bipartitioning *

Jason Cong, Michail Romesis and Joseph R. Shinnerl
UCLA Computer Science Department
Los Angeles, CA 90095-1596
{cong,romesis,shinnerl}@cs.ucla.edu

Technical Report 040057
January 21, 2005

Abstract

A novel and very simple correct-by-construction top-down methodology for mixed-size placement is presented. The PolarBear algorithm combines recursive cutsize-driven partitioning with fast and scalable legalization of every placement subproblem generated by every partitioning. The feedback provided by the legalizer at all stages of partitioning improves final placement quality significantly on standard IBM benchmarks and dramatically on low-white-space adaptations of them. Compared to Feng Shui 2.6 and Capo 9.0, PolarBear is the only tool that can consistently find high-quality placements for benchmarks with less than 5% white space. With white space at 12%, PolarBear beats Capo 9.0 by 6% in average total wirelength while Feng Shui 2.6 frequently fails to find legal placements altogether. With 20% white space, PolarBear still beats Capo 9.0 by 10% and FengShui 2.6 by 2% in average total wirelength, using very similar run time.

Key Words: Mixed-Size Placement, Legalization, Recursive Bipartitioning, White Space

1 Introduction

Advancing IC technology has brought increased use of large intellectual-property (IP) blocks in multi-million-gate ASICs and SOC designs. Most modern designs consist of a very large number of standard cells mixed with many big macros, such as ROMs, RAMs, and IP blocks. The placement of mixed-size cells has thus become a very important topic in physical design. The problem can be defined as follows. Given a fixed rectangular region $\mathcal{R}$ divided into rows of uniform height, arrange a set of rectangular standard cells and large macros in $\mathcal{R}$ such that adjacent cells and macros do not overlap. Standard cells all have a common height equal to the row height. Macros are significantly bigger and often span multiple rows. The objective is usually the minimization of total wirelength, expressed as the sum of the half-perimeters of the bounding boxes of the nets. However, other objectives — routed wirelength, timing — can be used instead, with various constraints — power, temperature, etc.

Compared to standard-cell placement, most of the increased difficulty in mixed-size placement is attributable to overlap removal, or *legalization*. Although in general legalization is NP-complete,

*Financial Support from Semiconductor Research Consortium Contract 2003-TJ-1091 and National Science Foundation Award CCF-0430077 is gratefully acknowledged.

legalization of a standard-cell placement is typically easy, because all standard cells have the same height and differ only in their widths. Most placement tools are able to produce legal standard-cell solutions, even when little white space is available, without sacrificing much wirelength. However, when large multi-row blocks are added to the design, placement becomes similar to floorplanning in complexity. In this context, it is often possible that even a good legalization algorithm can fail to find an overlap-free placement which retains the basic structure of a given global placement. Moreover, in designs of high row utilization, i.e., low white space, experiments show that publicly available state-of-the-art software may fail to find a legal solution altogether, even when a given global placement is known to be good in both wirelength and block density distribution.

Frequent failures by leading academic tools on designs with white space below 15% present a serious limitation on their applicability. In practice, the fabrication cost of an IC increases rapidly with the area of its layout. Designers therefore tend to pack their layouts as tightly as possible. Designs with white space between 5 and 10% white space are common; designs with less than 5% white space are not unusual [21, e.g.]. The ability to find high-quality overlap-free mixed-size placements scalably and reliably in a low-white-space setting is clearly important; yet, as discussed below, this ability has yet to be satisfactorily achieved, despite recent progress [2].

The work presented here demonstrates that mixed-size placement by recursive bipartitioning can be done very successfully in a scalable way that removes any need for post-hoc legalization *or backtracking*. A non-overlapping solution is obtained during global placement through the explicit construction of strictly legal layouts for every partition block at every level of the top-down hierarchy. These legal layouts are computed for all placement subproblems as soon as they are generated by min-cut bipartitioning. Targeting area as its primary objective, the legalizer has linear complexity, runs extremely fast, and finds legal solutions with a very high success rate, even with row utilizations over 99%. Its role is best understood not as predictor, but as *guarantor*. Most legal layouts for larger subproblems near the top of the bipartitioning hierarchy are not actually used. However, should legalization fail on a given subproblem, the existence of a known legal layout for its parent subproblem ensures that legality can be restored quickly and locally, without a potentially costly backtracking through multiple generations of ever larger ancestor regions. In this situation, feedback from the legalizer implicitly guides subsequent partitioning, significantly improving the quality of the final placement [12].

Dramatically improved performance on low-white-space designs is achieved by this approach. By scalably incorporating legalization into the hierarchical flow, better partitioning decisions are made during global placement, and placements of superior quality are obtained in extremely fast $\mathcal{O}(N \log N)$ run time. The implementation of our algorithm, PolarBear ("Placement by Legalized Recursive Bipartioning is more Robust"), consistently obtains high-quality placements on standard mixed-size benchmarks, even as the amount of white space is decreased to just 1%. In contrast Feng Shui [15] cannot consistently obtain legal solutions on these benchmarks with less than 16% white space, and Capo 9.0 fails at or near 5–10%. All three tools use comparable run time and show similar scalability. For the 20% white space value given in the Dragon benchmarks [20] PolarBear's average total wirelength is 2% less than that of Feng Shui 2.6 and 10% less than that of Capo 9.0.

The remainder of the paper is organized as follows. Section 2 briefly describes recent work on mixed-size placement. Section 3 presents the main elements of the PolarBear algorithm, including RBP. Experiments and results are presented in Section 4, and conclusions and future work are discussed in Section 5.

2 Related Work

Mixed-size placement has recently drawn considerable attention, with several recent papers reporting large improvement. We divide this work into two categories: methods requiring legalization after global placement, and methods whose global placements are overlap-free by construction.

As of November 2004, the best published wirelength results are obtained by methods requiring legalization after global placement. FengShui 2.6 [15] uses recursive-bisection with iterative deletion, iterative repartitioning, relaxed rows not aligned with standard cell rows ("fractional cut"), and a simple Tetris-style approach to legalization. A-place [13] employs a multiscale, force-directed formulation and nonlinear conjugate-gradient iterations. A-place's run time is approximately 8× that of Feng-Shui on the first 10 test IBM/ISPD98 test cases; results are not available for the 8 larger test cases.

Previously published correct-by-construction algorithms for mixed-size placement all rely on simulated annealing in some crucial way. mPG [7] builds a cluster hierarchy for multiscale optimization in a physical-hierarchy-generation framework. mPG uses simulated annealing (SA) on the Sequence-Pair [17] floorplanning representation over nested grids at every level of the cluster hierarchy for legalization. Reliance on annealing slows mPG down considerably.

Correctness by construction is a relatively recent addition to Capo [6, 3, 1, 2]. Capo 9.0 proceeds top down by cutsize-driven recursive bipartitioning until certain ad-hoc tests suggest that newly generated subproblems may be difficult to legalize. At that point, standard cells in each subproblem are clustered, and these clusters are treated as soft macros. SA-based fixed-outline floorplanning is then attempted on the hard macros and soft clusters for the given subregion. If it succeeds, the locations of the macros are then fixed, and further refinement proceeds on the declustered soft macros. If it fails, then the subproblem is merged with its sibling, the previous partition of the parent subproblem is discarded, and floorplanning is attempted for the parent subproblem. In priniciple, this backtracking may continue indefinitely until some ancestor is successfully floorplanned or until failure at the top level occurs. In practice, the ad-hoc tests used to determine when to commence floorplanning are observed to be good enough that backtracking is only rarely needed. However, when white space is particularly scarce, e.g., less than 4%, Capo 9.0 reports failures, presumably because its ad-hoc tests are insufficient to prevent floorplanning on subproblems that are too large for its SA-based floorplanner to solve scalably. Moreover, clustering standard cells to form rectangular blocks may prevent a hierarchical method from finding an optimal or near-optimal solution in terms of wirelength and delay minimization [9].

Although PolarBear has some superficial similarity to Capo 9.0 [2], its differences are quite significant, as evidenced by its superior performance and robustness. First, PolarBear guarantees legality by look-ahead rather than backtracking. Failures in its intermediate-level legalizations are used to provide important feedback to its cutsize-driven partitioner. Ultimately, this feedback allows its recursive bipartitioning to continue to single-cell end cases. Second, all its core algorithms are scalable and deterministic. Rather than use floorplanning to enforce legality, it uses extremely fast and simple row-oriented block packing (RBP). Because this legalization technique is so fast and scalable, it can be applied to every subproblem at every level, even on the flat problem at the very top level. In this way, backtracking is avoided, and run time is used very consistently and predictably.

Recent work has also drawn attention to the connection between mixed-size placement and floorplanning [2]. Recent advances in fast floorplanning by recursive bipartitioning [18, 10] are also related to the work in this paper. In particular, the PATOMA floorplanner [10] also constructs explicit floorplans for each of the floorplanning subproblems generated by recursive bipartitioning.

3 The Polar Bear Algorithm

PolarBear employs top-down, cutsize-driven, recursive bipartitioning in combination with explicit area legalization of every placement subproblem generated by each partitioning. Cut-size driven area bipartitioning is done by hMetis [14]. Subproblem legalization is performed by a simple, scalable heuristic which we call row-oriented block packing (RBP). The main contribution of PolarBear is not in either of these components, but in how they are combined in the recursive-bipartitioning flow. The feedback from RBP guides post-partitioning modification of the hMetis solution used

to legalize the two generated subproblems. The recursive partitioning process continues until every block is assigned to its own subregion. Placement of blocks in their assigned subregions automatically produces a legal layout. This layout is further improved by greedy swapping of neighboring cells in detailed placement.

3.1 Flow Overview

Pseudocode for PolarBear is shown in Figure 1. By an instance, we mean a rectangular placement region $\mathcal{R}$; a set of cells and macros $\mathcal{V}$, each $v \in \mathcal{V}$ of prescribed area and shape; and a hypergraph netlist $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where each net $e \in \mathcal{E}$ is a subset of $\mathcal{V}$. For brevity, we refer to cells and macros collectively as "blocks." Initially, a completely legal overlap-free placement to the given instance is computed by RBP without regard to wirelength. Because RBP is scalable, it cannot in general *guarantee* that it will find a legal solution, if one exists. However, by ignoring wirelength, the RBP heuristic is typically able to find tightly packed placements quickly, even with total white space below 1%. We have yet to observe an RBP failure at this initial stage on any available benchmark circuit. Henceforth, we assume that this initial application of RBP succeeds; therefore, every subproblem is descended from a parent subproblem for which an explicit, legal, overlap-free solution is known. Under this assumption, PolarBear is guaranteed to terminate with a legal placement.

Given an RBP placement of the given instance, the algorithm proceeds recursively in the same way on that instance as on every subproblem it subsequently generates. Cells and macros are partitioned by hMetis into two subsets, the ratio ρ of whose total areas satisfying $2/3 \leq \rho \leq 3/2$ (balance factor 10%). The partitioning attempts to minimize the total number of nets connecting blocks in both subsets. Connections to fixed terminals along the placement region or in other subregions are modeled by terminal propagation. The placement region $\mathcal{R}$ is then sliced to create two new placement subproblems, one for each of the two block subsets. Initially, the cutline is placed in proportion to the areas of the block subsets, and its initial orientation minimizes the aspect ratios of the subregions it creates. As described below, its position and orientation may subsequently be changed.

The key feature distinguishing PolarBear from other min-cut-based placement algorithms is its incorporation of legalization into its top-down flow. Before recurring with hMetis on the two new subproblems, PolarBear first calls RBP in order to construct legal, overlap-free placements of those two subproblems. Although the initial application of RBP may fail to legalize one or both of them, one of four separate correction strategies described in Section 3.3 below is guaranteed to succeed, given the legal layout of the parent subproblem. These explicitly constructed legal subproblem solutions then allow recursive, legalized, cutsize-driven bipartitioning to continue separately on each of them. The algorithm continues in this fashion down to end cases of subregions containing one block each.

3.2 Prelegalization by Block Packing (RBP)

Because legalization in PolarBear is always its *first* step for computing a placement, it is also called "prelegalization." The purpose of prelegalization is to find a legal, non-overlapping configuration of a given set of blocks in a given region as quickly as possible. This legal configuration is used to guide actual placements only if one of its child subproblems defined by cutsize-driven partitioning cannot also be legalized. Otherwise, once legal placements are obtained for both its child placements, the prelegalized solution is discarded. For speed, simplicity, and modularity, wirelength is ignored during prelegalization.[1] In order that the prelegalizer may serve as guarantor of the legalizability of subsequent placement subproblems contained within its subregion, each of its solutions is required

[1] Though some wirelength gain might be obtained by incorporating network flows into prelegalization, such techniques were not attempted in the current implementation.

PolarBear Mixed-Size Placement

input: Set of hard blocks $\mathcal{V} = \{v_1, \ldots, v_n\}$; netlist $\mathcal{H} = (\mathcal{V}, \mathcal{E})$; rectangular region $\mathcal{R}$ of fixed dimensions.
remark: Each node of the bipartitioning tree is a triple: (i) a set of blocks V, (ii) a rectangular subregion R, and (iii) a legalized placement $P(V, R)$ of V in R.

Create an initially empty queue Q of prelegalized placement subproblems. Apply RBP to $\mathcal{V}$ in $\mathcal{R}$.

if (RBP fails to prelegalize $\mathcal{V}$ in $\mathcal{R}$)
  Report a failure of PolarBear to the caller.
else Denote RBP's legal placement of $\mathcal{V}$ in $\mathcal{R}$ by $\mathcal{P}$. Set the root node to $(\mathcal{V}, \mathcal{R}, \mathcal{P})$.
enqueue the prelegalized root in Q.

while (Q is nonempty) **do**
  dequeue a prelegalized subproblem $S = (V, R, P)$.
  Partition V into disjoint subsets V_1, V_2 by hMetis with terminal propagation. Slice R into subregions R_1, R_2, and assign V_1, V_2 to them.
  Let $P_1 :=$ RBP(V_1, R_1) and $P_2 :=$ RBP(V_2, R_2).
  notation: P_i is **true** if and only if P_i is legal.
  if (**not** (P_1 **and** P_2))
    if (cutline search legalizes P_1 and P_2)
      continue
    else if (repartitioning legalizes P_1 and P_2)
      continue
    else if (block swapping legalizes P_1 and P_2)
      continue
    else refine $P =$ RBP(V, R) to reconstruct legal P_1 and P_2.
  end if
  remark. P_1 and P_2 are now legal.
  if ($|V_1| > 1$) **enqueue** (V_1, R_1, P_1) in Q.
  if ($|V_2| > 1$) **enqueue** (V_2, R_2, P_2) in Q.
end do
output: a legal placement of $\mathcal{V}$ inside region $\mathcal{R}$

Figure 1: Overview of the PolarBear algorithm.

to contain at least one straight-line *slice*, either horizontal or vertical, which can be used as a cutline. This process is described in Section 3.3 below.

Prelegalization in PolarBear is an extremely simple form of row-oriented block packing (RBP). Macros and cells ("blocks") are taken in nonincreasing-height order and placed in consecutive rows in the subregion. Each block is placed in the first row in which it fits in a way that preserves at least one slice. Individual rows are filled from left to right. Macros typically span multiple rows. Therefore, stacks of smaller blocks may appear to the right of larger blocks (Figure 2). If at any point, a macro or a standard cell cannot fit in the specified region, the algorithm reports failure. The row-oriented structure ensures that either (i) a horizontal slice along a row boundary exists; or, (ii) the tallest macro spans all rows, creating a vertical slice. A small sample RBP layout is shown

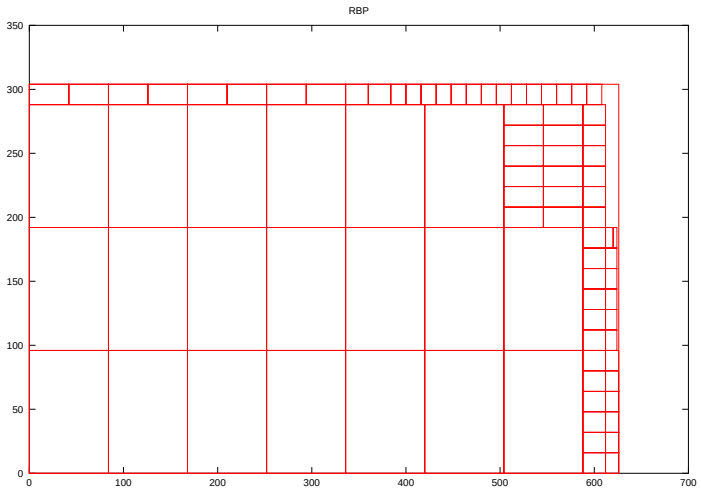

Figure 2: Sample RBP Layout

in Figure 2.

3.3 RBP Feedback to Bipartitioning

If RBP initially fails to find a legal solution to a given subproblem, four separate corrective measures are attempted in sequence. The first three measures — cutline repositioning, repartitioning, and iterated block swapping — are not guaranteed to legalize. When they succeed, however, they preserve a given cutsize-driven partitioning as closely as possible. If all three fail, then cutsize-driven partitioning of the parent subproblem is abandoned, and the prelegal RBP solution to the parent subproblem is instead adopted and refined.

Cutline Search

When RBP finds a legal solution to one of the subproblems but not the other, the cutline can be moved away from the failed case and toward the solved one. A limited number of iterations (3–12) of binary search on the cutline position is performed. The block subsets of the subregions are held fixed, and for each candidate cutline position, RBP is attempted anew on the same block subsets in the new candidate subregions.

Repartitioning

If one of the placement subproblems still cannot be solved after cutline search, the entire process is repeated for up to 10 new hMetis partitionings or until legality is obtained for both subproblems. Experiments to date produce the best quality/run-time tradeoff with 2 runs of hMetis for each of 5 different balance factors: 10, 15, 5, 20, and 25%. Overall, replacing these multiple runs of hMetis by just one run at balance factor 10% increases total wirelength by 9%.

Iterated Block Swapping

When repartitioning and cutline search reach their limits, the first hMetis solution with 10% balance factor is restored for attempted correction by iterative refinement. Suppose that RBP successfully finds a legal placement for subregion $\mathcal{A}$ but not for its sibling subregion $\mathcal{B}$. Usually, a small number

of small adjustments to the given cutsize-driven partitioning suffices to determine legal solutions to both subproblems. A partial solution of $\mathcal{B}$ is generated by running RBP while skipping the placement of the blocks that do not fit in the subregion. The legal solution to $\mathcal{A}$ and the partial solution to $\mathcal{B}$ are used as a starting point. First, the blocks that are not contained in $\mathcal{B}$ by the partial solution are moved across the cutline from $\mathcal{B}$ to $\mathcal{A}$. This step legalizes the placement in $\mathcal{B}$ but typically renders the solution to $\mathcal{A}$ illegal. In order to re-legalize $\mathcal{A}$, the cutline is first moved as far toward $\mathcal{B}$ as possible, so that the width of $\mathcal{B}$ is the same as the width of the widest row of blocks there. Then RBP is rerun on the new subproblem for $\mathcal{A}$. If RBP fails on this new subproblem, then the above steps are repeated with the roles of regions $\mathcal{A}$ and $\mathcal{B}$ reversed. This refinement continues up to a maximum of 10 iterations until either legal placements to both subregions are found, or cycling is detected, i.e., a given set of leftover blocks appears more than once for different iterations of the same subproblem. If a legal target layout is found, but there are multiple blocks of the same dimensions which can be relocated in order to obtain that layout (a common scenario), then the blocks actually moved are selected to reduce wirelength, as estimated by placing all pins at subregion centers.

Experiments demonstrate that iterated block swapping is the most effective of the correction heuristics used in PolarBear. When it is omitted, average total wirelength increases by 14%, while run time increases by only 3%.

Refining an RBP Solution

If iterated block swapping fails to legalize a given placement subproblem, then PolarBear returns to its parent subproblem, for which a legal RBP solution has already been computed and stored. A non-legalized *target* solution to this subproblem is then computed by traditional min-cut placement: recursive cutsize-driven bipartitioning coupled with terminal propagation, cutline specification, and assignment of the block subsets to the subregions defined by the cutline position. Locations of blocks in this target solution are used to guide the refinement of the given RBP solution, as follows. Blocks of identical dimensions in the RBP solution are permuted in order to move them as close to their locations in the target solution as possible. I.e., the original RBP solution is viewed as a template for the ultimate assignment of its blocks to the subregions currently associated with the blocks.

In PolarBear, the permutation is generated by sorting the block locations in the RBP solution by their y-coordinates, if a partition along the x-dimension will follow, or by their x-coordinates, if the partition will be along the y-dimension. The target locations are sorted in the same fashion. Juxtaposing these orderings gives the assignment.

The permuted RBP placement is bipartitioned, and the main algorithm resumes separately on each of its two child subproblems. In order to guarantee the legality of subproblem solutions, however, the permuted RBP placement is partitioned along one of its row or column boundaries, and not by generic, cutsize driven hMetis bipartitioning.[2] A few nearly centered, row or column-separating cutlines for the RBP solution and its symmetric solution (flipped across the cutline) are considered for its bipartitioning. For each of these candidates, wirelength is estimated by placing all blocks in each subregion at the subregion's center and modeling external connections by terminal propagation. The selected cutline produces the least estimated wirelength. An example of our approach is shown in Figure 3. On average, this refinement of the guarantor RBP solution reduces total wirelength by 3%.

4 Experiments and Results

The PolarBear algorithm was implemented with the gcc 3.2.3 compiler on a 2.4 GHz Pentium 4 processor in a RedHat 9.0 Linux environment. It was compared with the two leading mixed-size placement algorithms that are publicly available online: Feng Shui 2.6 [22] and Capo 9.0 [19].

[2] In our experiments, the vast majority of the cases where PolarBear has to resort to this step were observed to be very small in the number of blocks — always less than 30.

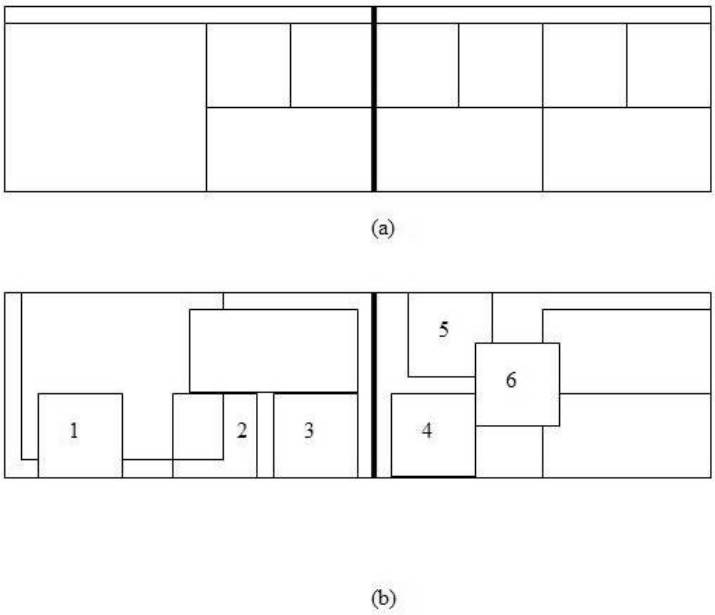

Figure 3: Refining the RBP Layout. The nonlegalized "ideal" min-cut placement (b) is used to guide permutations of identically sized blocks 1 – 6 in the legal RBP layout (a).

Both these tools use recursive bipartitioning, but their methodologies are different (cf. Section 2). Feng Shui is very aggressive during global placement; it shows relatively little consideration for nonoverlapping constraints. After global placement, it uses a simple Tetris-like legalization algorithm [11, 16] to remove overlap. However, this combination consistently fails to produce legal placements on the IBM benchmarks when white space is decreased below 12%. Capo 9.0 uses backtracking and SA-based floorplanning to construct correct layouts without post-hoc legalization. However, as white space decreases to near 5%, it often reports failures also, presumably because its backtracking proceeds to subproblems too large for its floorplanner to handle with acceptable run time.

The results for the IBM/ISPD98 benchmarks [4] for mixed-size placement are reported in Table 1 and Figure 4. Twenty different versions of the IBM benchmarks were generated by setting the white space available in the placement region from 1% up to 20% white space in increments of 1%. PolarBear is clearly much more robust than Feng Shui 2.6 and Capo 9.0. It successfully computed a legal placement for every benchmark tested, with every value of white space, down to 1% white space.[3] On IBM10, PolarBear beats Feng Shui by 25% on average over the different white space values where Feng Shui succeeds. Solutions produced by Feng Shui 2.6 are consistently legal over all the benchmarks only with white space at least 16%. Solutions produced by Capo 9.0 are consistently legal over all the benchmarks only with white space at least 12%. Capo 9.0 typically does find legal solutions when white space is as low as 5%, but not consistently. For some benchmarks, even with 15% white space, multiple runs of Capo 9.0 were needed to find a feasible solution. Table 1 reports for each benchmark both the least value of white space for which a legal solution was found and the greatest value of white space for which a failure was observed.

With 20% white space,[4] PolarBear is on the average 2% better than Feng Shui and 10% than

[3] It seems evident that legal placements can also be computed under much less than 1% white space, at some cost in increased wirelength. This possibility was not investigated.

[4] By default, PolarBear shrinks its internal core region to reduce the white space it actually uses to 10%, when the given white space exceeds 10%. This feature was enabled for the experiments reported in Table 1 but disabled for those described in Figure 4.

Capo 9.0. In runtime, PolarBear is slightly slower than Feng Shui (1.3×) and comparable to Capo. Overall, the results suggest that, compared to existing methods, enforcing legalization during min-cut, mixed-size placement leads both to much improved robustness and better placement quality.

Circuit	PB WL	PB CPU	FS WL	FS CPU	FS BEST	FS WORST	CAPO WL	CAPO CPU	CAPO BEST	CAPO WORST
ibm01	2.35	6	1.07	0.33	12%	11%	1.17	1.00	5%	12%
ibm02	5.09	5	1.02	1.00	13%	12%	1.07	1.00	4%	9%
ibm03	7.72	9	1.01	0.56	16%	15%	1.10	2.22	5%	4%
ibm04	8.2	11	1.05	0.55	7%	6%	1.19	1.45	1%	-
ibm05	10.45	5	0.95	1.60	1%	-	1.04	1.20	1%	-
ibm06	6.69	11	1.08	0.73	14%	13%	1.17	1.36	4%	9%
ibm07	11.55	18	0.99	0.67	10%	9%	1.05	0.78	5%	4%
ibm08	13.27	31	1.02	0.48	15%	14%	1.12	1.16	2%	1%
ibm09	14.02	20	1.00	0.75	13%	12%	1.10	0.90	3%	12%
ibm10	30.49	33	1.30	0.64	16%	15%	1.16	1.00	13%	14%
ibm11	19.51	35	1.00	0.63	16%	15%	1.13	0.74	3%	6%
ibm12	37.33	50	1.03	0.50	13%	12%	1.11	0.72	6%	12%
ibm13	25.19	55	0.98	0.69	14%	13%	1.13	0.91	3%	10%
ibm14	38.76	57	0.98	0.96	4%	3%	1.07	0.89	12%	11%
ibm15	51.74	134	1.01	0.51	11%	10%	1.08	0.61	5%	10%
ibm16	62.22	93	0.97	0.98	8%	7%	1.06	0.87	7%	6%
ibm17	73.33	97	0.98	0.95	7%	6%	1.06	0.71	3%	12%
ibm18	46.83	83	0.94	1.10	6%	5%	1.07	0.73	3%	16%
Avg.	1	1	1.02	0.76			1.10	1.02		

Table 1: Comparison of PolarBear (PB) with Feng Shui 2.6 (FS) and Capo 9.0. Reported wirelength for PolarBear is half-perimeter bounding-box summed over all nets. Reported runtime for PolarBear is in minutes. Values for Feng Shui and Capo are normalized to PolarBear's results. The columns labeled "Worst" report the greatest white space percentages for which Feng Shui and Capo reported at least one failure. The columns labeled "Best" report the least white space percentages for which Feng Shui and Capo reported at least one success.

5 Conclusion

Integrating legalization with global min-cut placement produces solutions of superior quality, speed, and robustness. The approach is scalable, suitable for the incorporation of complex constraints, and readily adaptable to combinations with other techniques. The basic techniques used in the prototype implementation described here are extremely simple and can obviously be enhanced to achieve even better quality. E.g., recent advances in analytical placement [8, 23] can be used to guide partitioning in place of cutsize minimization [5]. As the complexity and heterogeneity of modern designs continue to increase, we expect the flexible "correct by construction" framework to become ever more attractive.

References

[1] S. Adya, I. Markov, and P. Villarrubia. On whitespace and stability in mixed-size placement and physical synthesis. In *Proc. Int'l Conf. on Computer-Aided Design*, 2003.

[2] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. Unification of partitioning, placement, and floorplanning. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 12–17, Nov. 2004.

[3] S. N. Adya and I. L. Markov. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proc. Int'l Symp. on Phys. Design*, pages 12–17, April 2002.

[4] C. Alpert. The ISPD98 circuit benchmark suite. In *Proc. Int'l Symp. on Phys. Design*, pages 80–85, 1998.

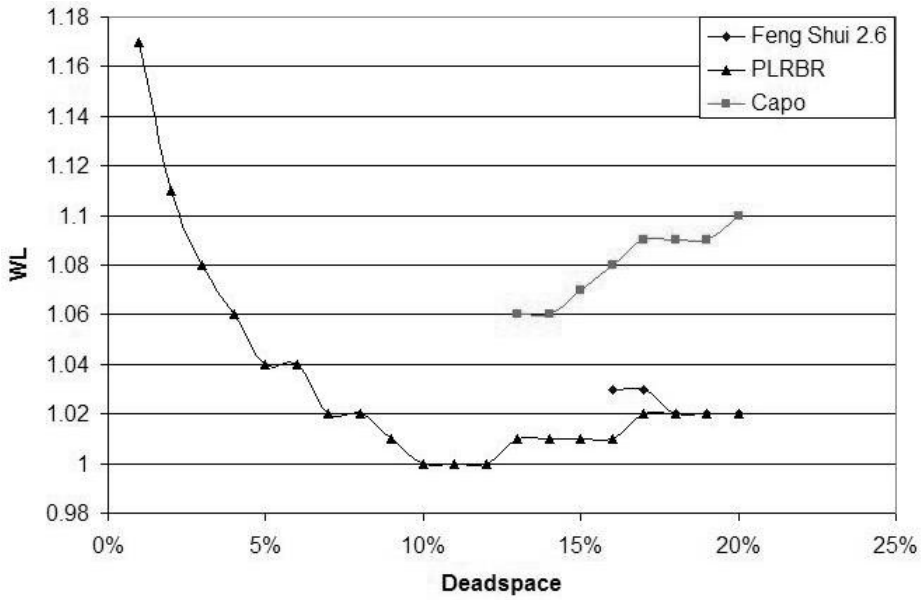

Figure 4: The performance of PolarBear, Feng Shui 2.6 and Capo 9.0 under limited white space. Half-perimeter wirelengths are normalized to PolarBear's results for 10% white space. For each white space value, results for each tool are shown here only when that tool finds a legal placement for every benchmark in the suite. Up to four runs of Capo 9.0 were necessary in order for it to obtain legal solutions in some cases.

[5] U. Brenner and A. Rohe. An effective congestion-driven placement framework. In *Proc. Int'l Symp. on Phys. Design*, Apr 2002.

[6] A. Caldwell, A. Kahng, and I. Markov. Can recursive bisection alone produce routable placements? In *Proc. 37th IEEE/ACM Design Automation Conf.*, pages 477–482, 2000.

[7] C.-C. Chang, J. Cong, and X. Yuan. Multilevel placement for large-scale mixed-size ic designs. In *Proc. Asia South Pacific Design Automation Conference*, pages 325–330, 2003.

[8] C. Chu and N. Viswanathan. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. In *Proc. Int'l Symp. on Phys. Design*, pages 26–33, April 2004.

[9] J. Cong. An interconnect-centric design flow for nanometer technologies. *Proceedings of the IEEE*, 89(4):505–528, 2001.

[10] J. Cong, M. Romesis, and J. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. In *Asia South Pacific Design Automation Conf.*, 2005.

[11] D. Hill. Method and system for high speed detailed placement of cells within an integrated circuit design. In *US Patent 6370673*, Apr 2002.

[12] A. Kahng and S. Reda. Placement feedback: A concept and method for better min-cut placements. In *Proc. Design Automation Conf.*, pages 357–362, June 2004.

[13] A. Kahng and Q. Wang. An analytic placer for mixed-size placement and timing-driven placement. In *Proc. Int'l Conf. on Computer-Aided Design*, 2004.

[14] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. 34th ACM/IEEE Design Automation Conference*, pages 526–529, 1997.

[15] A. Khatkhate, C. Li, A. R. Agnihotri, S. Ono, M. C. Yildiz, C.-K. Koh, and P. H. Madden. Recursive bisection based mixed block placement. In *Proc. Int'l Symp. on Phys. Design*, 2004.

[16] C. Li and C.-K. Koh. On improving recursive bipartitioning-based placement. Technical Report TR-ECE 03-14, Purdue University, 2003.

[17] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-packing-based module placement. In *Proc. International Conference on Computer-Aided Design*, pages 472–479, 1995.

[18] A. Ranjan, K. Bazargan, S. Ogrenci, and M. Sarrafzadeh. Fast floorplanning for effective prediction and construction. In *IEEE Trans. on VLSI Sys.*, pages 341 – 351, 2001.

[19] http://vlsicad.eecs.umich.edu/bk/pdtools/tar.gz/.

[20] `http://er.cs.ucla.edu/Dragon/`

[21] http://www.faraday-tech.com/structuredasic/download.html.

[22] `http://vlsicad.cs.binghamton.edu/software.html`

[23] Z. Xiu, J. Ma, S. Fowler, and R. Rutenbar. Large-scale placement by grid warping. In *Proc. Design Automation Conf.*, pages 351–356, June 2004.