Constraint-Driven Floorplan Repair

Michael D. Moffitt[†], Jarrod A. Roy[‡], Igor L. Markov[‡], Martha E. Pollack[‡]
[†]IBM Austin Research Lab
[‡]University of Michigan, Ann Arbor

In this work we propose a new and efficient approach to the *floorplan repair* problem, where violated design constraints are satisfied by applying small changes to an existing rough floorplan. Such a floorplan can be produced by a human designer, a scalable placement algorithm, or result from engineering adjustments to an existing floorplan. In such cases, overlapping modules must be separated, and others may need to be repositioned to satisfy additional requirements. Our algorithmic framework uses an expressive graph-based encoding of constraints which can reflect fixed-outline, region, proximity and alignment constraints. By tracking the implications of existing constraints, we resolve violations by imposing gradual modifications to the floorplan, in an attempt to preserve the characteristics of its initial design. Empirically, our approach is effective at removing overlaps and repairing violations that may occur when design constraints are acquired and imposed dynamically.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*placement and routing*; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design; G.4 [**Mathematical Software**]: Algorithm Design and Analysis

General Terms: Design, Algorithms

Additional Key Words and Phrases: Floorplanning, Legalization, Constraints

1. INTRODUCTION

The significance and complexity of floorplanning is continually increasing with the growth of systems-on-chip. With hundreds and thousands of modules in modern floorplans, all-manual design is infeasible. However, existing algorithms fail to handle important design constraints. Even the simplest non-overlapping constraint is often violated by recent floorplanners. A similar challenge in standard-cell placement has been successfully addressed by decoupling global placement from legalization, where the former optimizes interconnect and is allowed to violate many design constraints. While a great deal of work on legalization in placement has been published [Brenner et al. 2004; Cong et al. "Robust ..." 2005; Ren et al. 2005], much of it is inapplicable to classical floorplanning, where modules have different shapes and are not placed within individual rows.

In this work we propose a new efficient approach to floorplan legalization. Our tool – Floorist – performs legalization and repair of existing floorplans that may have been produced by a variety of layout methodologies. In contrast to previous

work, our algorithm takes a *conflict-directed* approach to repair, modifying only those features of the layout that are directly responsible for violated constraints. As a result, the solutions produced by FLOORIST preserve the qualities and characteristics of the original layout. Unlike diffusion-based methods in [Ren et al. 2005], our technique can spread large movable macros, and does not require models of a physical diffusion process. Furthermore, we exploit the expressive power of a graph-based framework to enable the repair of a wide variety of violated constraints.

2. RELATED WORK: LEGALITY AND LEGALIZATION

Techniques for ensuring overlap-free placements can be classified as correct-by-construction (in which legality is guaranteed a priori) and construct-by-correction (in which legalization is postponed until post-processing).

Correct-by-Construction Approaches. mPG [Chang et al. 2003] enforces legalization at every level of a cluster hierarchy in multi-scale optimization, using simulated annealing on sequence-pairs, which is often expensive. Capo [Roy et al. 2006] legalizes subproblems of recursive bi-partitioning using simulated annealing as well. If legalization succeeds, modules are placed after further refinement. If legalization fails, subproblems are merged and legalization is attempted on the larger problem. As a result, Capo can generate infeasible instances if whitespace is tight. In the mixed-size placer from [Cong et al. "Robust ..." 2005], every partition is guaranteed to be legalizable by a fast constructive algorithm.

Construct-by-Correction Approaches. FengShui [Khatkhate et al. 2004] and APlace [Kahng et al. 2005] postpone legalization until a global floorplan has been generated, making use of a simple Tetris-like algorithm by D. Hill (US Patent 6370673), which sorts cells and macros by their x-coordinates and places them one-by-one into rows to greedily minimize displacement. Diffusion-based legalization in [Ren et al. 2005] builds a discrete approximation of a continuous diffusion equation; cells are presumed to fit into rows, and macros must be fixed.

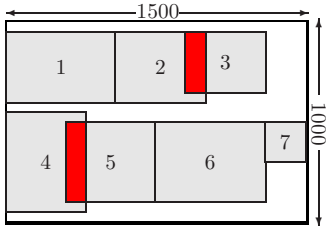
In [Nag and Chaudhary 1999] a given layout is converted into a sequence-pair, and is manipulated by perturbing the ordering of pairs of modules and relocating individual modules. In XDP [Cong and Xie 2006] a constraint-graph is used, in which overlaps are removed in a preprocessing step that increases the dimensions of the layout beyond the fixed outline. However, XDP seeks to reduce HWPL, rather than preserve the characteristics of the original design; as such, it has the potential to degrade other qualities of the solution (such as timing), and is thus less appropriate when considering Engineering Change Orders (ECOs). Furthermore, both of these prior approaches perform adjustments to reduce the size of the resulting floorplan, rather than remove or repair constraint violations. Indeed, neither of the techniques described in [Nag and Chaudhary 1999] and [Cong and Xie 2006] handle the presence of fixed macros and obstacles, making their applicability to complex industrial designs questionable. Our contributions are compared to prior work in Table I.

3. THE FLOORIST ALGORITHM

FLOORIST ("*Floorplan Assistant*") begins with a global floorplan which may violate constraints due to block resizing, or for any other reason. Then it applies a three-

Table I. Comparison of placement legalization tools.

	Legalizes large macros	Handles fixed obstacles	Handles generic constraints	Scalable	Produces small displacement	Supports module spreading
[Nag and Chaudhary 1999]	+	−	−	−	−	−
[Brenner et al. 2004]	−	+	−	+	+	−
[Khatkhate et al. 2004]	+	−	−	+	−	−
[Kahng et al. 2005]	+	+	−	+	−	−
[Ren et al. 2005]	−	+	−	+	−	+
[Cong and Xie 2006]	+	−	−	+	−	−
Our work	+	+	+	+	+	+



i	$w_i \times h_i$	$(x_i\ ,\ y_i)$
1	550 × 350	(0 , 600)
2	450 × 350	(550 , 600)
3	400 × 300	(900 , 650)
4	400 × 500	(0 , 50)
5	450 × 400	(300 , 100)
6	550 × 400	(750 , 100)
7	200 × 200	(1300 , 300)

Fig. 1. A small floorplan in need of repair.

stage procedure outlined below.

3.1 Translation to Constraint Graphs

The lower-left corner of any module M_i (with dimensions $w_i \times h_i$) is assigned a coordinate (x_i, y_i) indicating its position. We refer to this mapping from modules to fixed positions as an *absolute assignment*. However, since flaws in the floorplan typically include overlaps, it is useful to instead consider a *relative assignment* of pairwise relationships between blocks rather than absolute positions. For this, we use a variant of *constraint graphs* [Moffitt and Pollack 2006]. A pair of graphs (G_H and G_V) is constructed, where each graph contains a node i for every module M_i and there exists a directed edge for every pair of modules M_i and M_j. The direction of this edge, and the graph to which it belongs, depends on the pairwise relationship between M_i and M_j. If M_i is to be placed to the left of M_j, equivalently $x_i + w_i \leq x_j$, this would require an edge from node i to node j in G_H with weight w_i. We label this relationship $L(i, j)$ – the other possible labels being $R(i, j)$, $A(i, j)$, $B(i, j)$ (for *right of*, *above*, and *below*). This notation allows us to refer to the set S of all pairwise relationships between modules. The following set contains some of the relationships that could be used to describe the floorplan shown in Figure 1:

$$\{L(1,2), A(1,4), R(3,4), L(5,7)\} \subset S$$

Notice that modules 2 and 3 overlap in this layout, as do modules 4 and 5. The relations introduced thus far are not sufficient for a pair of modules that overlap. As such, we propose the addition of a fifth "empty" relation – $E(i, j)$ – to indicate the lack of any constraint between M_i and M_j. $E(i, j)$ can be regarded as the absence or relaxation of an edge in either constraint graph. Since previous graph-based encodings do not permit the explicit relaxation of such constraints, we will generally refer to these as a *relaxed pair* of constraint graphs.

Create-Graphs$((x_1, y_1, ..., x_N, y_N), (w_1, h_1, ..., w_N, h_N))$

1. $S \leftarrow \emptyset$
2. For $j = 1$ to N
3. For $i = 1$ to j
4. (Of the choices below, keep one w/ smallest slack)
5. If $(x_i + w_i \leq x_j)$ $S \leftarrow S \cup \{L(i,j)\}$
6. If $(x_j + w_j \leq x_i)$ $S \leftarrow S \cup \{R(i,j)\}$
7. If $(y_i + h_i \leq y_j)$ $S \leftarrow S \cup \{A(i,j)\}$
8. If $(y_j + h_j \leq y_i)$ $S \leftarrow S \cup \{B(i,j)\}$
9. (If no choices, $S \leftarrow S \cup \{E(i,j)\}$)
10. return $G_H(S)$, $G_V(S)$

Fig. 2. Translating the initial placement into constraint graphs.

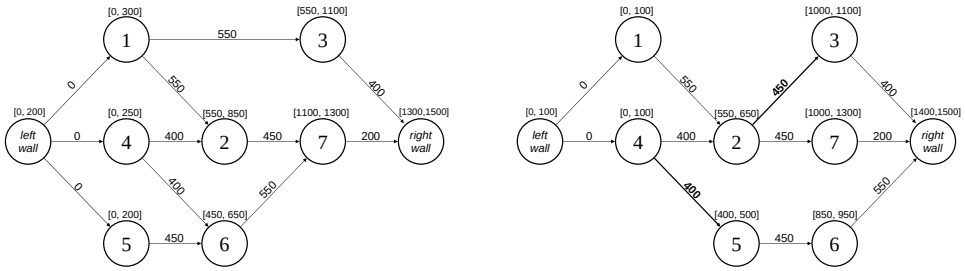


Fig. 3. The horizontal constraint graph G_H corresponding to the floorplan in Figure 1 before repair (left) and after repair (right).

Building relaxed constraint graphs is straightforward (see Figure 2). For every pair of modules that are relatively displaced in a certain direction, the proper relation is added to the set S. When multiple relationships exist, ties are broken based on the horizontal and vertical distances between the modules. Consider modules 1 and 7 in our small example. While edges corresponding to $L(1,7)$ and $A(1,7)$ are both feasible choices, we prefer $L(1,7)$, since the horizontal displacement between these modules is much greater than the vertical displacement. Any pair of modules that overlap is given the $E(i,j)$ relation. Once S has been constructed, constraint graphs can be built in $O(N^2)$ time by adding the appropriate edges and computing single-source longest paths. Feasible regions of each module can be cheaply extracted from the graph-based encoding. Figure 3 (left) shows G_H for Figure 1. Each node is labeled with upper and lower bounds on its possible positions, e.g., the smallest horizontal coordinate for Block 2 is 550.

3.2 Conflict-Directed Iterative Repair

For a given set S of pairwise relationships, our objective is to resolve every empty relation $E(i,j)$ in S, as they correspond to violated constraints. The original solution should be preserved as much as possible. Our greedy, backtrack-free iterative repair algorithm is described in Figure 4. Lines 2-6 attempt to replace $E(i,j)$ relationships whenever one of the other four relationships is freely available (e.g., sufficient slack exists in the graphs to add the appropriate edge). Consider the $E(2,3)$ relation in our example. Module 3 can be shifted toward the right wall to remove its overlap with module 2. This information is obtained directly from the constraint graphs. As a result, we can instead invoke the relation $L(2,3)$ and

```
Iteratively-Repair(S)
1.  While (∃(E(i,j) ∈ S))
2.    // this first loop makes 'trivial' assignments
3.    For each E(i,j) ∈ S
4.      For each possible pairwise relation P(i,j)
5.        If (consistent(S ∪ {P(i,j)} − {E(i,j)}))
6.          S ← S ∪ {P(i,j)} − {E(i,j)} ; break
7.    // this second loop swaps existing assignments
8.    For each E(i,j) ∈ S
9.      For each possible pairwise relation P(i,j)
10.       C ← Critical-Path(M_i) ∪ Critical-Path(M_j)
11.       For each P'(i',j') ∈ C
12.         For each P''(i',j') such that P'' ≠ P'
13.           If (consistent(S ∪ {P''(i',j')} − {P'(i',j')}))
14.             S ← S ∪ {P''(i',j')} − {P'(i',j')}
15.             continue loop @ line 11 with next P'(i',j')
```

Fig. 4. Our iterative repair procedure.

update G_H by adding the constraint $x_2 + w_2 \leq x_3$. The total number of overlaps has now been reduced from two to one. Unfortunately, the relation $E(4, 5)$ cannot be resolved directly — no other pairwise relationship between modules 4 and 5 can be introduced without either violating another non-overlap constraint, or extending the layout beyond the given fixed outline. In other words, the relations in the set $\{L(4, 5), R(4, 5), A(4, 5), B(4, 5)\}$ are in *conflict* with the current constraint graphs and call for more drastic modifications.

Possible alternatives can be seen in Figure 1 — module 6 can be placed below module 7, rather than to its left. This would amount to removing the relation $L(6, 7)$ and replacing it with $B(6, 7)$ and the constraint $y_7+h_7 \leq y_6$ (adding the appropriate edge in G_V). After performing this modification, the last remaining overlap can be resolved by adding the relation $L(4, 5)$ and the constraint $x_4 + w_4 \leq x_5$. Other relations can be swapped in this example, but few help to resolve the violated constraint. For instance, we could reverse the relationship between modules 1 and 2 by replacing $L(1, 2)$ with $R(1, 2)$, but such a modification would have no effect on the overlap between modules 4 and 5. This suggests *conflict-directed* legalization, which is the key concept in our floorplan repair procedure, which computes *critical paths* [Adya and Markov 2003], i.e., tightly packed sequences of blocks that constrain each other in the same direction. The critical path for a particular module can be obtained by traversing predecessor nodes within the appropriate constraint graph. For instance, consider Block 7 in the horizontal constraint graph of Figure 3 (left). One path of blocks that precedes it is the sequence $4 \rightarrow 2 \rightarrow 7$, which would give a lower bound of 1000. However, the sequence $5 \rightarrow 6 \rightarrow 7$ is the true critical path, as it provides a tighter lower bound of 1100. The critical path can be regarded as an explanation for a module's current location.

For each of the four primary relations of a violated non-overlap constraint, we identify a set of *culprits* C by examining edges along the critical paths for M_i and M_j in the appropriate constraint graph (lines 8 – 10 in Figure 4). If any of these edges can be safely replaced with an alternative pairwise relationship (lines 11 – 13), then we perform this replacement (line 14). The order in which these relations are attempted is heuristically chosen based on the modules' original locations. Each swap can be achieved in $O(N^2)$ time using the longest-path algorithm (although

Minimize-Displacement(G_H, G_V)
1. For $i = 1$ to N
2. G_H.AddEdge($left\_wall$, x_i, $\max(orig\_x_i, G_H.lb(x_i))$)
3. G_H.AddEdge(x_i, $right\_wall$, $\min(orig\_x_i, W - G_H.ub(x_i))$)
4. G_V.AddEdge($bottom\_wall$, y_i, $\max(orig\_y_i, G_V.lb(y_i))$)
5. G_V.AddEdge(y_i, top_wall, $\min(orig\_y_i, H - G_V.ub(y_i))$)

Fig. 5. Emulation: The creation of a fixed placement that resembles the original solution.

incremental techniques can be used to improve efficiency). In the event that some previously violated constraints can now be satisfied, these will be caught during the first phase of the next step in iterative repair. Otherwise, additional replacements will be performed repeatedly to further manipulate the layout.

This process continues until all violated constraints have been resolved or some alternative termination criterion has been reached (such as a maximal number of iterations, or timeout limit). We note that since the algorithm is not exhaustive, it is not guaranteed to find a feasible solution within any bounded amount of search (for instance, the blocks given may be entirely unpackable, in which case no legalizer would be able to find a feasible solution). However, our experimental results will demonstrate that even with a conservative amount of whitespace, our algorithm is extremely effective in removing all overlaps, and so additional termination criteria were typically not needed. Since no history of prior configurations is maintained, it is possible for the algorithm to encounter the same configuration multiple times. However, as with traditional local search (e.g., simulated annealing), the frequency of these repetitions will be largely mitigated by randomization.

Example 1: In Figure 3 (right) we show G_H after legalization is complete. This is the result of three consecutive operations applied to the original graph from Figure 3 (left): (a) an edge has been added from Block 2 to 3, (b) an edge has been removed between Blocks 6 and 7, and (c) an edge has been added from Block 4 to 5. New edges have been highlighted in bold. Some edges that are implied by transitivity (e.g., the edge from Block 1 to Block 3) are omitted.

3.3 Translation to Fixed Locations

The final layout can be found from the graph-based encoding in several ways. Aside from the obvious *packing* solutions that gravitate blocks toward a particular side or corner, we develop two more elaborate means to minimize displacement and maximize separation.

Minimizing Displacement from the original locations can be achieved by incrementally adding edges to the graph in order to lock modules into locations near their initial positions. To emulate the initial placement, we propose pseudocode in Figure 5. For each module, we use G_H to extract the lower and upper bounds on its final horizontal position x_i. From this, we can determine whether or not it can be given its original horizontal coordinate. If it can, edges are added to ensure this. Otherwise, the module is slid as close to its original position as possible and edges are added as appropriate. A similar operation is performed to determine the module's vertical position. See illustration in Figure 7.

Example 2: The largest module in our example is Block 6, which has a feasible window of 850 to 950 after legalization (see Figure 6, left). We could select any value in this range and guarantee a consistent layout. Unfortunately, the original

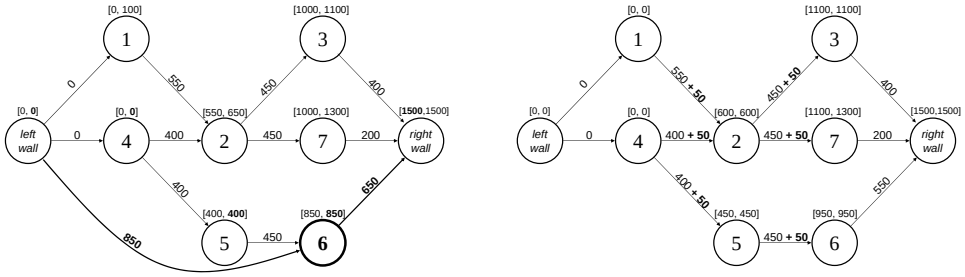


Fig. 6. Finding final positions for all of the blocks by emulating the original placement (left) or maximizing separation between blocks (right). To emulate the initial solution, a pair of edges has been added to lock Block 6 into a horizontal coordinate near its original position (left). To achieve separation, a constant is applied to the weight on each edge between a pair of modules (right).

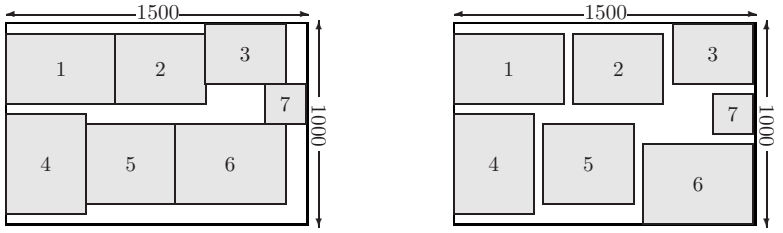


Fig. 7. Our floorplan after repair and either emulation (left) or separation (right).

horizontal position of Block 6, 750, is outside this range. Emulation would thus choose 850 as the closest possible value, adding an edge from the left wall to Block 6 with a weight of 850. To constrain this block from the right, a second edge would be added from Block 6 to the right wall with a weight of 650 (since $1500 - 850 = 650$). As can be seen in Figure 6, these edges serve to lock not only Block 6 into place (horizontally), but also Blocks 4 and 5 as well.

Complexity: Each addition of an edge to the constraint graph requires worst case $O(N^2)$ time. For N blocks, the entire emulation procedure takes $O(N^3)$ time, but is invoked only once, and can be bypassed when preserving initial locations is unimportant. As shown in Section 4, emulation can process thousands of modules in under a second. It is clearly more scalable than sequence-pair based annealing which requires at least N^4 steps in most common implementations.

Maximizing Separation. An alternative objective is the even spreading of modules across the chip, with an eye on subsequent physical synthesis and routing that often require a more balanced allocation of space. Figure 8 outlines our process of *separation maximization*. The effect is similar to that achieved by diffusion-based spreading [Ren et al. 2005] that is based on solving PDEs for chemical diffusion. In contrast, our separation procedure operates upon the constraint-graph representation, is discrete in nature and ensures legality.

Both separation and emulation add edges incrementally until blocks are locked into fixed positions. However, rather than insert new edges between modules and the walls, separation iteratively increases the weight of the arc between each non-

```
Maximize-Separation(G_H, G_V, increment)
1. dist ← 0, changed ← false
2. while (changed)
3.     dist ← dist + increment
4.     For j = 1 to N
5.         For i = 1 to j
6.             If (L(i, j)) changed ← G_H.AddEdge(x_i, x_j, min(w_i + dist, G_H.ub(x_j) − G_H.lb(x_i)))
7.             If (R(i, j)) changed ← G_H.AddEdge(x_j, x_i, min(w_j + dist, G_H.ub(x_i) − G_H.lb(x_j)))
8.             If (A(i, j)) changed ← G_V.AddEdge(y_i, y_j, min(h_i + dist, G_V.ub(y_j) − G_V.lb(y_i)))
9.             If (B(i, j)) changed ← G_V.AddEdge(y_j, y_i, min(h_j + dist, G_V.ub(y_i) − G_V.lb(y_j)))
```

Fig. 8. Separation: The creation of a fixed placement that spreads modules across the chip.

overlapping pair by a small constant. This is repeated until no additional changes in the graph are observed at which point each module will have no additional slack. The additional weight added to each edge is bounded by slack, as to not violate the fixed outline.

Example 3: In Figure 6, we show the horizontal constraint graph after an additional weight of 50 has been applied to each edge. In this case the additional constant term lock all blocks in place horizontally except for Block 7. Figure 7 (right) illustrates layout after the repair and separation procedures.

Complexity: The naïve implementation suggested by Figure 8 would require $O(N^4)$ time for each iteration on the value of the *distance* variable. The total complexity can be reduced to $O(N^3)$ time if all individual edge weights are instead updated in a single collective sweep, with one global pass to propagate these new values along the longest paths contained within the graph.

4. EXPERIMENTAL RESULTS

We compare FLOORIST with legalizers from FengShui 5.1 [Khatkhate et al. 2004] and APlace 2.01 [Kahng et al. 2005], which are based on the Tetris algorithm.[1] Since the tools compared do not support soft blocks, all blocks are considered hard with an aspect ratio of 1.0.

4.1 Repairing the Output of a Global Floorplanner

We ran FengShui 5.1's legalizer and FLOORIST on final solutions produced by APlace 2.01.[2] Table 2 of [Moffitt et al. 2006] (not reproduced due to space limits) shows detailed results of these experiments. We find that FengShui 5.1 is extremely fragile, as it crashes on the majority of instances. Furthermore, when it does generate a solution, it often *increases* the amount of overlap and tends to violate the fixed outline constraint by placing modules out of core. On all instances that we tried, FengShui fails to produce legal solutions. In contrast, FLOORIST achieves legality and 0% overlap on all instances, preserving global interconnect length while remaining competitive in runtime. The layouts produced by using FLOORIST on these initial solutions are each superior in wirelength to the previous best legal

[1] A binary provided to us of the XDP legalizer [Cong and Xie 2006] used by mPL6 crashed on all IBM-HB instances [Cong et al. "Fast ..." 2005] used in our experiments.

[2] We focus on the output of APlace since, as observed in [Moffitt et al. 2006], the latest version of Capo generally does not produce illegal solutions, and FengShui is too unreliable to produce consistent results on these benchmarks.

Table II. Performance of legalizers on placements with reshaped modules.

% reshaped →		10%	20%	30%	40%	50%	60%	70%	80%	90%
avg. % initial ovlp		1.38	2.81	3.98	5.59	6.92	8.23	9.73	11.11	12.31
FengShui	% solved	10	0	0	0	0	0	0	0	0
	avg. % ovlp	81.1	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	avg. HPWL	3.11E6	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	avg. runtime (s)	12.36	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
APlace	% solved	80	35	25	0	0	0	0	0	0
	avg. % ovlp	0.343	0.389	0.431	N/A	N/A	N/A	N/A	N/A	N/A
	avg. HPWL	2.90E6	2.96E6	3.04E6	N/A	N/A	N/A	N/A	N/A	N/A
	avg. runtime (s)	40.5	54.3	75.8	N/A	N/A	N/A	N/A	N/A	N/A
Floorist	% solved	100	100	100	100	100	100	100	100	100
	avg. % ovlp	0	0	0	0	0	0.040	0.221	0.226	0.466
	avg. HPWL	2.86E6	2.88E6	2.90E6	2.91E6	2.93E6	2.94E6	2.94E6	2.96E6	2.97E6
	avg. runtime (s)	2.40	6.44	10.2	15.5	28.6	60.0	180.3	197.4	227.0

solutions. The average reduction in wirelength observed is 7% [Moffitt et al. 2006]. Refer to Figure 9 for sample layouts generated in this experiment.

4.2 Repairing Floorplans with Resized Blocks

In a second set of experiments, we reshape a percentage of blocks in a legal placement (APlace's `ibm01` solution legalized by FLOORIST) and change aspect ratios from 1.0 to either 0.5 or 2.0. The percentage of blocks reshaped is varied between 10% and 90%, and 20 initial solutions are created for each such percentage. The overlaps in the resulting set of benchmarks are much more significant than those in the first experiment, both in number and size. In Table II we report the percentage of instances on which a given legalizer returned a solution, the average amount of overlap in such solutions, average HPWL, and average runtime for solved instances. FengShui's legalizer returns solutions for only two benchmarks, which are far from legal. APlace's legalizer is somewhat more reliable, and produces at least one solution out of twenty for cases where the percentage of blocks resized is 30% or less. However, the solutions are not completely legal. Furthermore, its runtime is over a minute when 30% of the blocks are reshaped. FLOORIST produces entirely legal solutions for all problems where the percentage of blocks reshaped is as high as 50%. It also returns nearly-legal solutions in extreme cases with as many as 90% of blocks being reshaped, and produces HPWL that is superior to that of APlace. On easy problems, its runtime is almost negligible, but increases monotonically with the illegality present in the initial solutions as expected.

4.3 Handling Obstacles

FLOORIST also excels in handling layouts with fixed obstacles; such problems have traditionally been difficult for annealers, in part because popular floorplan representations do not easily permit the encoding of such constraints. The representation used by FLOORIST handles obstacles almost effortlessly - one needs only to add the appropriate edges between the fixed block and the surrounding walls, and preclude these edges from being removed during repair. This simple modification allows FLOORIST to reason about fixed obstacles.

To produce results in Table III, we take the solutions produced by FLOORIST in the previous experiment with 32% whitespace, and drop five fixed blocks on top of these layouts (one in each corner, and a fifth in the middle). We progressively

Table III. Performance of legalizers on placements with increasingly larger obstacles.

% of area taken by obs. →		1%	3%	5%	7%	9%	11%	13%	15%
FengShui	% solved	0	0	0	0	0	0	0	0
	avg. % ovlp	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	avg. movement	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	avg. runtime (s)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
APlace	% solved	100	100	100	100	100	100	5	0
	avg. % ovlp	0.272	0.230	0.281	0.265	0.273	0.279	0.360	N/A
	avg. movement	128k	114k	102k	95k	95k	107k	150k	N/A
	avg. runtime (s)	79.7	80.5	84.5	79.3	74.7	72.6	75.4	N/A
Floorist	% solved	100	100	100	100	100	100	100	100
	avg. % ovlp	0	0	0	0	0	0.015	0.876	3.57
	avg. movement	10k	26k	39k	49k	59k	68k	76k	75k
	avg. runtime (s)	1.50	3.90	7.69	14.32	26.16	78.15	297	400

increase the percentage of area consumed by these obstacles from 1% to 15%. FengShui ignores these obstacles, and makes no effort to repair the floorplan. APlace requires significant runtime to handle obstacles regardless of problem difficulty, introduces dramatic changes to the layouts and fails to completely resolve overlaps on module boundaries. FLOORIST performs significantly better on all counts. See Figure 10 for illustrations.

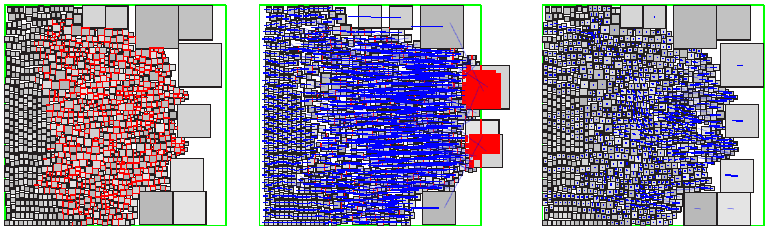
4.4 Repairing Other Constraint Types

As noted in [Young et al. 2002], non-overlap constraints are just one of many types of constraints that traditional constraint graphs can express. For instance, region constraints, proximity constraints, and alignment constraints can all be represented by edges in the graph, and similarly, their violation can be regarded as the *absence* of such edges. Consequently, our conflict-directed approach can repair these constraints just as easily by manipulating the critical paths that render them infeasible.

To illustrate this ability, we present a series of images in Figure 12, showing how FLOORIST repairs a violated region constraint. The initial solution is displayed in the leftmost layout; one block currently violates an artificially imposed constraint that requires its center to be aligned with the horizontal midsection of the floorplan. Second, we show the movement of blocks as displacement vectors, to highlight how the majority of the modules are largely unaffected by this repair. In the third image we show the final layout after FLOORIST has repaired this constraint.

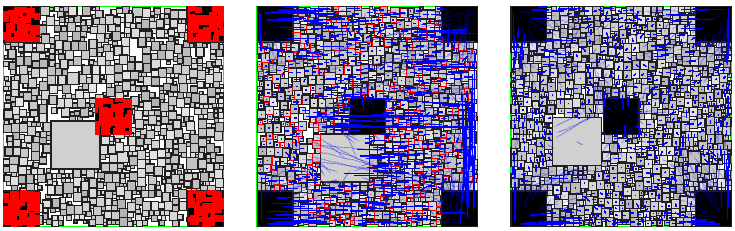
4.5 Comparison of Methods for Constructing Layouts from Constraint Graphs

Section 3.3 offers three methods for computing placements: *packing*, *emulation*, and *separation*. Emulation was used in previous experiments, and we now compare it with alternatives. For each block in a layout, we find the distance from it to the closest block in x or y direction, and average this distance for all blocks to compute the *spread*. We use the instances from Experiment 2 (with 10% blocks resized), and average results over the 20 instances. We find that *packing* is fast and produces reasonable wirelength, but displaces modules significantly. *Emulation* is also efficient but generally preserves the initial placement. Both procedures fail to provide padding on at least one side of most modules. In contrast, the *separation* procedure maintains a healthy distance between pairs of modules, but is an order of magnitude slower. The difference between the latter two alternatives and APlace's



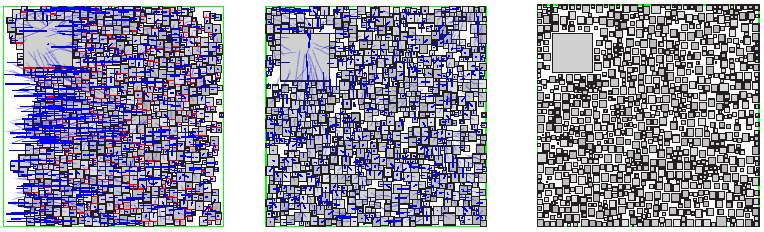
(a) APlace (Initial) (b) FengShui's Legalizer (c) Floorist

Fig. 9. Legalizing the output of APlace (an analytical global placement tool) on `ibm-07`.



(a) Initial (b) APlace's Legalizer (c) Floorist

Fig. 10. Legalizing a placement with obstacles added to the perimeter and center of the floorplan.



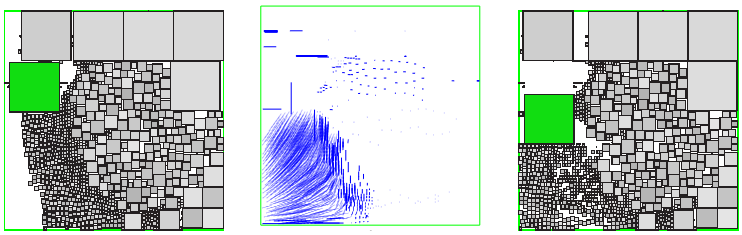
(a) APlace's Packing (b) Floorist's Emulation (c) Floorist's Separation

Fig. 11. Floorist's Emulation and Separation techniques compared against APlace's packing method.

packing is shown in Figure 11 (the rightmost layout was given a greater amount of whitespace to illustrate separation).

5. CONCLUSIONS

We have developed a new efficient approach to post-placement floorplan repair. Our Floorist algorithm legalizes existing floorplans using constraint-driven, conflict-directed modifications. In contrast to previous work, Floorist modifies only those features of the layout that are directly responsible for the violated constraints, and thus preserves the qualities and characteristics of the original layout. The versatility of our approach makes it useful for a number of applications, as it supports the post-processing of outputs of a global floorplanner, removes overlaps introduced by the re-sizing of modules in existing floorplans, and repositions blocks in the presence of fixed macros and obstacles. In each of these categories, Floorist significantly



(a) Initial (b) Movement (c) Final

Fig. 12. Imposing a region constraint on a block in the above HB12 instance forces it (and blocks beneath it) to move down.

outperforms existing tools in solution quality and runtime.

Our formalism does have some limitations; for instance, on instances with millions of blocks and relatively few overlaps, a great deal of runtime could be spent propagating constraints between standard cells whose pairwise relationships will never change. In addition, more intelligent decisions could be made if our legalization flow were cognizant of timing information (e.g., a list of timing-critical gates or paths). These issues, and others, are worthy of continued research.

Acknowledgments. Dr. M. D. Moffitt is supported by the Josef Raviv Memorial Postdoctoral Fellowship. J. A. Roy is supported by a Rackham Predoctoral Fellowship at the University of Michigan. Prof. I. L. Markov is partially supported by the National Science Foundation under grant No. CCF-0448189. Prof. M. E. Pollack is partially supported by the Air Force Research Laboratory under contract No. FA8750-05-1-0282 and the Air Force Office of Scientific Research under Contract No. FA9550-04-1-0043. Opinions, findings and conclusions in this work are those of the authors and do not necessarily reflect the view of funding agencies.

REFERENCES

S. N. Adya and I. L. Markov. Fixed-outline Floorplanning: Enabling Hierarchical Design. In *IEEE Trans. on VLSI Systems* 11(6), pp. 1120–1135, 2003.

U. Brenner, A. Pauli, and J. Vygen. Almost optimum placement legalization by minimum cost flow and dynamic programming. In *Proc. of ISPD*, pp. 2–9, 2004.

C. Chang, J. Cong, and X. Yuan. Multi-level placement for large-scale mixed-size IC designs. In *Proc. of ASP-DAC*, pp. 325–330, 2003.

J. Cong, M. Romesis, and J. R. Shinnerl. Fast floorplanning by look-ahead enabled recursive bipartitioning. In *Proc. of ASP-DAC*, pp. 1119–1122, 2005.

J. Cong, M. Romesis, and J. R. Shinnerl. Robust mixed-size placement under tight white-space constraints. In *Proc. of ICCAD*, pp. 165–172, 2005.

J. Cong and M. Xie. A Robust Detailed Placement for Mixed-Size IC Designs. In *Proc. of ASP-DAC*, pp. 188–194, 2006.

A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In *Proc. of ICCAD*, pp. 891–898, 2005.

A. Khatkhate, et al. Recursive bisection based mixed block placement. In *Proc. of ISPD*, pp. 84–89, 2004.

M. D. Moffitt, A. N. Ng, I. L. Markov, and M. E. Pollack. Constraint-driven floorplan repair. In *Proc. of DAC*, pp. 1103–1108, 2006.

M. D. Moffitt and M. E. Pollack. Optimal rectangle packing: a meta-CSP approach. In *Proc. of ICAPS*, pp. 93–102, 2006.

S. Nag and K. Chaudhary. Post-Placement Residual-Overlap Removal with Minimal Movement. In *Proc. of DATE*, pp. 581–586, 1999.

H. Ren, D. Z. Pan, C. J. Alpert and P. Villarrubia. Diffusion-based placement migration. In *Proc. of DAC*, pp. 515–520, 2005.

J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov. Min-cut floorplacement. In *IEEE Trans. on CAD* 25(7), pp. 1313–1326, 2006.

B. Yao, H. Chen, C.-K. Cheng and R. Graham. Floorplan representations: Complexity and connections. In *ACM TODAES* 8(1), pp. 55–80, 2003.

E. Young, M. L. Ho, and C. Chu. A Unified Method to Handle Different Kinds of Placement Constraints in Floorplan Design. In *Proc. of ASP-DAC*, pp. 661–670, 2002.