ECO-system: Embracing the Change in Placement

Jarrod A. Roy and Igor L. Markov
The University of Michigan, Department of EECS
2260 Hayward Ave., Ann Arbor, MI 48109-2121
{royj, imarkov}@umich.edu

Abstract— In a realistic design flow, circuit and system optimizations must interact with physical aspects of the design. For example, improvements in timing and power may require replacing large modules with variants that have different power/delay trade-off, shape and connectivity. New logic may be added late in the design flow, subject to interconnect optimization. To support such flexibility in design flows we develop a robust system for performing Engineering Change Orders (ECOs). In contrast with existing stand-alone tools that offer poor interfaces to the design flow and cannot handle a full range of modern VLSI layouts, our ECO-system reliably handles fixed objects and movable macros in instances with widely varying amounts of whitespace. It detects geometric regions and sections of the netlist that require modification and applies an adequate amount of change in each case. Given a reasonable initial placement, it applies minimal changes, but is capable of re-placing large regions to handle pathological cases. ECO-system can be used in the range from high-level synthesis, to physical synthesis and detail placement.

I. Introduction

In his keynote speech at ISPD 2006, Cadence CTO Ted Vucurevich expressed the need for "re-entrant, heterogeneous, incremental, and hierarchical" tools for EDA to handle the challenges of next-generation designs [17]. However, the importance of this problem has been realized much earlier, as Cong and Sarrafzadeh surveyed the state-of-the-art in incremental physical design techniques in 2000 and found these techniques to be largely "unfocused and incomplete" [11]. Kahng and Mantik also found disconnects between the relative strengths of incremental optimizers and perturbation techniques [21]. They conclude that CAD tools of the time "may not be correctly designed for ECO-dominated design processes" [21]. Considerable progress has been made since 2000, e.g., in incremental placement [2,4,6,13,18,19,23–27,31], but there is no common agreement on the main tasks solved by incremental tools and how these tasks should be solved. While incremental physical design is not new, it remains a difficult, high-value goal.

We focus on incremental placement legalization and improvement in large-scale layout. The need for such legalization typically arises in two contexts. The first is the separation of placement into global and detail, where rough placements are produced first and incrementally improved to avoid overlaps and fit into cell sites. This is common for analytical placers (APlace [22], mPL [8]) that approximate site constraints, while partitioning-driven tools (Capo [32], PolarBear [12]) and annealing-based tools (mPG [9], Parquet [3]) adopt correct-by-construction frameworks and require little post-processing.

However, the second context for legalization appears entirely unavoidable. During physical synthesis, timing-critical gates may be powered up and other gates may be powered down. These changes affect gate size and typically create overlaps [24]. Buffer insertion often leads to similar area violations, which must be resolved by legalization. The success of such legalization depends on how much the areas have changed, in what patterns, and the strength of a given legalizer. In particular, the legalization of mixed-size and block-based designs with obstacles remains very challenging [29].

Our work is focused on the design of a powerful and robust ECO tool that applies adequate amounts of replacement, in the right locations, to accommodate necessary design changes. To be useful in high-level and physical synthesis, such a tool must be able to entirely replace sections of the netlist, e.g., logic added to the design.

While practical considerations call for an interaction between global placers and legalizers, traditional work on ECO and detail placement focuses on stand-alone tools incapable of global placement. An attractive, but yet unexplored solution would be to extend an existing global placer to an incremental mode where it would automatically identify layout regions and sections of the netlist that need repair, but preserve satisfactory regions. In this work, we propose such an extension, identify and develop new components that allow a global placer to act like a powerful ECO tool, and develop a competitive implementation based on the open-source Capo tool.

As this tool can always resort to calling global placement on the entire design, it robustly handles a full range of modern designs, including those with obstacles and movable macros. Time-consuming global placement is not used when the initial placement is good.

We formulate the basic requirements for ECO placement and offer relevant algorithms. Our tool, ECO-system, is many times faster than a global placer and increases wirelength only slightly. ECO-system outperforms APlace's native legalizer on APlace global placements by over 1% in HPWL while running 3x faster. ECO-system supports extensive cell resizing producing legal results that mirror the original with virtually the same HPWL. Unlike WSA [23, 24], we handle obstacles and displace cells an order of magnitude less.

The rest of the paper is structured as follows. In Section II we review previous work. Key requirements and a likely interface are discussed in Section III. We present ECO-system in Section IV. Support for high-level and physical synthesis is discussed in Section V. In Section VI we show empirical results and conclude in Section VII.

II. Previous Work

Below we describe existing work on incremental techniques and discuss relevant aspects of global placement.

Incremental techniques. Previous work on legalization, incremental placement and detail placement can be broken into three fairly distinct stages: i) cell spreading, ii) legalization through simple end-case techniques, and iii) refinement of the legalized placement. For the first stage, several algorithmic paradigms have been applied in the literature such as network flows [6, 13, 14, 25], linear programming [13], top-down whitespace injection [23, 24] and diffusion gradients [31]. For end-case legalization, generally placers use greedy movement of cells such as in Capo [32], the Tetris legalizer [18] in FengShui [5], and greedy packing in DOMINO [14]. Lastly, placement refinement is done in sliding windows of one or more rows using optimal end-case placers based on branch-and-bound [7] or dynamic programming [19], as well as cell swapping such as in FastPlace [30].

One major theme in much of the literature is minimizing the total movement of cells in the design during legalization [6]. While our legalizer achieves remarkably small total/average movement, we point out that in general this does not always lead to minimal increase in interconnect parameters as shown in [1]. A legalization with minimal total cell displacement may cause a few cells to move a great distance. Better timing may be achieved by legalization with greater average movement, and even if the average movement is the same, there can be many alternative replacements.

1-4244-0630-7/07/$20.00 ©2007 IEEE.

Cell spreading. DOMINO [14] legalizes by splitting cells into pieces of identical sizes, solving a flow formulation to minimize movement, and finally reassembling the cell pieces. This limits the effectiveness of DOMINO to cells of similar sizes. Existing implementations of DOMINO do not account for obstacles and shift all cells to the left, limiting their applicability to modern placement instances, such as those from the ISPD05 contest [28]. Flow-based legalization methods such as those used in [6, 25] divide the core area into regions and redistribute cells between neighboring regions until no region has more cell area than available site area. These techniques can handle movable macros by fixing them early in the legalization process.

In [23, 24] cells are incrementally placed by injecting whitespace in a top-down fashion. The placement region is divided into a grid with bisection steps (based only on the size and shape of the region, not taking into account the cells, macros or fixed obstacles therein), and whitespace is injected based on some particular objective (routing congestion in [23], gate sizing and buffer insertion in [24]). Whitespace injection is done by shifting the geometric cut-lines to change the whitespace balance in regions. When cut-lines are shifted, the positions of the cells in the affected regions are scaled. Whitespace injection can cause significant overlap due to scaling, especially in the presence of fixed obstacles or movable macros as in the ISPD 2005 Contest benchmarks [28]. To remove these overlaps, a standard legalization step must be applied followed by window-based detail placement to recover HPWL. It is unclear how well this technique may work on difficult block-packing instances [29]. The technique may also fail in cases of extreme overlap, such as global placement by analytical placers, as large areas of the placement will be essentially random. The authors of [24] report an average displacement of 2.1% of the core area per cell, whereas the displacements observed with our technique are an order of magnitude smaller.

The diffusion technique of [31] legalizes by dividing the core area into a regular grid. Cells move from areas of high congestion to lower congestion (moving around fixed obstacles) and their directions and speeds are determined by solving equations similar to those in the process of chemical diffusion [31]. New placements are generated at each time step of the diffusion and the first solution which satisfies area constraints is taken to minimize runtime and cell movement [31]. End-case legalizers work within the grid regions to produce a final legal placement, but this may be impaired by difficult block-packing instances [29]. The work in [26] improves that in [31], but does not measure its impact on wirelength, congestion or timing.

The XDP technique [13] uses a combination of constraint graphs, network flows, linear programming and greedy cell movement for legalization of mixed-size designs. Overlaps between macros are legalized first by building constraint graphs until all macros can legally fit into the core. After the constraint graph is finalized, a linear programming instance is built and solved to remove macro overlap and move macros minimally. Standard cells are legalized with a greedy heuristic similar to that of FengShui [5], with the addition of flow-based methods [6, 25] as necessary. After legalization, window-based detail placement techniques are used to improve HPWL.

Macro legalization. It was shown that a fixed-outline floorplanner based on Simulated Annealing with sequence pairs could be used to remove overlap [2]. Techniques in [36] improve on [2] and show how to legalize with minimal perturbation. Removal of overlap between macros can be especially difficult given hard instances of block-packing [29]. To handle such instances, the authors of [29] modify B*-trees to account for obstacles. Recently, FLOORIST [27] has been proposed which uses constraint satisfaction to remove macro overlap.

Greedy legalization. FengShui [5] uses a simple packing algorithm by Hill [18] that is reminiscent of the Tetris game. Such legalization fares poorly in designs with large amounts of whitespace, as shown by the results of the ISPD 2005 Placement Contest. Capo uses two greedy legalizers for its global placements: one for macros and another for standard cells [32]. The macro overlap legalizer tries to move macros as little as possible so as not to affect neighboring standard cells. If space is available, standard cells are legalized via shifting. Otherwise cells are swapped between rows greedily until no row is overfull. Fixed obstacles are handled implicitly as they fracture rows [32].

Min-cut placement. ECO-system uses the top-down min-cut placement framework [5, 12, 29, 32–34]. Recent techniques for min-cut placement [10, 35] have produced some of the best placements in the ISPD 2006 contest [20] and the most routable placements on IBMv2 netlists [33]. In traditional min-cut algorithms, a placement is viewed as a series of placement bins, the first of which encompasses the core area and contains all movable cells. Based on number of cells in a placement bin, the placer either bisects the bin or places the bin's cells with an end-case placer.

When bisecting a bin, a min-cut placer proceeds by selecting a temporary cut-line for the bin based on the size and shape of the bin. Based on the amount of cell and site area in the bin, the placer determines partitioning tolerances. Given the tolerance, the placer uses a balanced min-cut partitioner to determine how to divide the cells between its child bins. Using the partitioning solution, the placer determines a final cut-line based on whitespace allocation techniques and divides the bin into child bins for further processing.

III. Requirements of Incremental Placement

Design optimizations that require incremental placement can alter a design in many ways [15] such as (see also Section V):

- Changing cell dimensions or net weights/criticalities
- Adding/Removing various constraints, such as density (to promote routability), regions (to address timing), etc.
- Inserting cells (with or w/o initial locations), nets or macros
- Adding obstacles (memories, IP blocks, RTL macros, etc.)

Generally these transformations create illegality in localized regions of a design and/or create opportunities for improving an existing placement. All of these transformations can be dealt with by performing placement from scratch, but this is undesirable: i) replacement can be slow, ii) the transformations may assume that they are applied to the current layout, and placement from scratch may invalidate them, and iii) the current layout may include *intangibles* such as designer intent, or be optimized for novel objectives not accounted for by the placement tool. Cong and Sarrafzadeh point out that incremental placers need to be able to trade off potentially several design objectives when operating on a placement [11].

In addition to preserving the original placement, a legalizer must also be able to completely replace sections of the placement that are deemed too suboptimal after design alterations. For example, if all of the cells are moved on top of one another at the center of the placement area, the legalizer should have the ability to replace all of the cells as the initial placement gives little useful information about a legal placement of the design. While this example is not typical of legalization as a whole, it is quite possibly the case for small sections of an illegal placement. This pathological case is not considered by most legalization techniques (such as those described in Section II).

Take for example the case when new cells are added to a design. If the new cells are added to isolated regions of the design, such as during buffer insertion, traditional techniques that perturb the design only slightly are most likely appropriate. Yet, timing optimization may call for pipelining of a multiplier or changing an adder to a different type. Adding a significant amount of new logic to an already placed and optimized design will require the functionality of a full-blown placer rather than just cell spreading to avoid degrading the design's wirelength and timing characteristics.

IV. Top-Down Legalization

To develop a strong ECO tool, we build upon an existing global placement framework and must choose between analytical and top-down. The main considerations include robustness, the handling of movable macros and fixed obstacles, as well as consistent routability of placements and the handling of density constraints. Based on

```
Variables:  queue of placement bins
Initialize queue with top-level placement bin
1     While(queue not empty)
2       Dequeue a bin
3       If(bin not marked to place from scratch)
4         If(bin overfull)
5           Mark bin to place from scratch, break
6         Quickly choose the cut-line which has
            the smallest net-cut considering
            cell area balance constraints
7         If(cut-line causes overfull child bin)
8           Mark bin to place from scratch, break
9         Induce partitioning of bin's cells from cut-line
10        Improve net-cut of partitioning with
            single pass of Fiduccia-Mattheyses
11        If(% of improvement > threshold)
12          Mark bin to place from scratch, break
13        Create child bins using cut-line and partitioning
14        Enqueue each child bin
15      If(bin marked to place from scratch)
16          If(bin small enough)
17            Process end case
18          Else
19            Bi-partition the bin into child bins
20            Mark child bins to place from scratch
21            Enqueue each child bin
```

Fig. 1. Our ECO algorithm. Lines 3-15 and 20 are different from traditional min-cut.

```
Input:  placement bin, balance constraint
Output:  x-coordinate of best cut-line
1     numCutlines = 1 + ⌊(rightBinEdgeX − leftBinEdgeX)/cellSpacing⌋
2     Create three arrays of size numCutlines:  LEFT, RIGHT, AREA
3     Initialize all elements of LEFT, RIGHT, and AREA to 0
4     Foreach net
5         Calculate x-coordinate of leftmost and rightmost pins
6         leftCutlineIndex = max(0,⌈(leftmostPinX − leftBinEdgeX)/cellSpacing⌉)
7         rightCutlineIndex = max(0,⌈(rightmostPinX − leftBinEdgeX)/cellSpacing⌉)
8         if(leftCutlineIndex < numCutlines) LEFT[leftCutlineIndex] += 1
9         if(rightCutlineIndex < numCutlines) RIGHT[rightCutlineIndex] += 1
10    Foreach cell
11        Calculate x-coordinate of the center of the cell
12        cutlineIndex = max(0,⌈(cellCenterX − leftBinEdgeX)/cellSpacing⌉)
13        if(cutlineIndex < numCutlines) AREA[cutlineIndex] += cellArea
14    Set X = leftBinEdge, CURCUT = 0, BESTCUT = ∞, BESTX = ∞, LEFTPARTAREA = 0
15    For(I = 0; I < numCutlines; I += 1, X += cellSpacing)
16        CURCUT += LEFT[I]
17        CURCUT −= RIGHT[I]
18        LEFTPARTAREA += AREA[I]
19        If(CURCUT < BESTCUT and LEFTPARTAREA satisfies balance constraint)
20          BESTCUT = CURCUT
21          BESTX = X
22    Return BESTX
```

Fig. 3. Algorithm for finding the best vertical cut-line from a placement bin. Finding the best horizontal cut-line is largely the same process. Note that the runtime of the algorithm is linear in the number of nets, cells and cut-lines incident to the bin.

recent empirical evidence [29, 33, 34], the top-down framework appears a somewhat better choice. Indeed the 2 out of 9 contestants in the ISPD 2006 Competition that satisfied density constraints were top-down placers. However, analytical algorithms can also be integrated into our ECO-system when particularly extensive changes are required. We base ECO-system on the open-source min-cut placer Capo [32] and plan to distribute it with Capo as well.

General framework. The goal of ECO-system is to reconstruct the internal state of a min-cut placer that could have produced a given placement **without the expense of global placement**. Given this state, we can choose to accept or reject previous decisions based on our own criteria and build a new placement for the design. If many of the decisions of the placer were good, we can achieve a considerable runtime savings. If many of the decisions are determined to be bad, we can do no worse in terms of solution quality than placement from scratch. An overview of the application of ECO-system to an illegal placement is depicted in Figure 2. See the algorithm in Figure 1.

To rebuild the state of a min-cut placer, we must reconstruct a series of cut-lines and partitioning solutions efficiently. To extract a cut-line and partitioning solution from a given placement bin, we examine all possible cut-lines as well as the partitions they induce. We start at one edge of the placement bin (left edge for a vertical cut and bottom edge for a horizontal cut) and move towards the opposite edge. For each potential cut-line encountered, we maintain the cell area on either side of the cut-line, the partition induced by the cut-line and the net cut.

Fast cut-line selection. For simplicity, assume that we are making a vertical cut and are moving the cut-line from the left to the right edge of the placement bin (the techniques necessary for a horizontal cut are analogous). Pseudo-code for choosing the cut-line is shown in Figure 3. To find the net cut for each possible cut-line efficiently, we

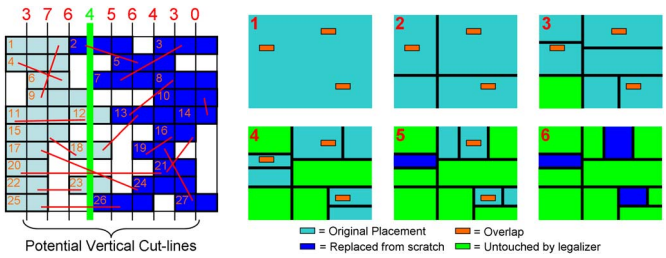

Fig. 2. Fast legalization by ECO-system. The image on the left illustrates choosing a vertical cut-line from an existing placement. Nets are illustrated as red lines. Cells are individually numbered and take 2 or 3 sites each. Cut-lines are evaluated by a left-to-right sweep (net cuts are shown above each line). A cut-line that satisfies partitioning tolerances and minimizes cut is found (thick green line). Cells are assigned to "left" and "right" according to the center locations. On the right, placement bins are subdivided using derived cut-lines until i) a bin contains no overlap and is ignored for the remainder of the legalization process or, ii) the placement in the bin is considered too poor to be kept and is replaced from scratch using min-cut or analytical techniques.

first calculate the bounding box of each net contained in the placement bin from the original placement. We create two lists with the left and right x-coordinates of the bounding boxes of the nets and sort them in increasing x-order. While sliding the cut-line from left to right (in the direction of increasing x-coordinates), we incrementally update the net-cut and amortize the amount of time used to a constant number of operations per net over the entire bin. We do the same with the centers of the cells in the bin to incrementally update the cell areas on either side of the cut-line as well as the induced partitioning. While processing each cut-line, we save the cut-line with smallest cut that is legal given partitioning tolerances. An example of finding the cut-line for a partitioning bin is shown in Figure 2.

Once a partitioning has been chosen, we accept or reject it based on how much it can be improved by **a single pass of a Fiduccia-Mattheyses partitioner with early termination** (which takes only several seconds even on the largest ISPD05 circuit). The intuition is that if the constructed partitioning is not worthy of reuse, a single Fiduccia-Mattheyses pass could improve its cut non-trivially.[1] If the Fiduccia-Mattheyses pass improves the cut beyond a certain threshold, we discard the solution and bisect the entire bin from scratch. If this test passes, we check legality: if a child bin is overfull, we discard the cut-line and bisect from scratch.

Scalability. Pseudo-code for the cut-line location process used by ECO-system is shown in Figure 3. The runtime of the algorithm is linear in the number of pins incident to the bin, cells incident contained in the bin, and possible cut-lines for the bin. Since a single Fiduccia-Mattheyses pass takes also takes linear time [16], the asymptotic complexity of our algorithm is linear. If we let P represent the number of pins incident to the bin, C represent the number of cells in the bin and L represent the number of potential cut-lines in the bin, the cut-line selection process runs in $O(P+C+L)$ time. In the vast majority of cases, $P > C$ and $P > L$, so the runtime estimate simplifies to $O(P)$.

The number of bins may double at each hierarchy layer, until bins are small enough for end-case placement. End-case placement is generally a constant amount of runtime for each bin, so it does not affect asymptotic calculations. Assume that ECO-system is able to reuse all of the original placement. Since ECO-system performs bisection, it will have $O(\log C)$ layers of bisection before end-case placement. At layer i, there will be $O(2^i)$ bins, each taking $O\left(\frac{P}{2^i}\right)$ time. This gives a total time per layer of $O(P)$. Combining all layers gives $O(P \log C)$. Empirically, the runtime of the cut-line selection procedure (which includes a single pass of a Fiduccia-Mattheyses partitioner) is much smaller than partitioning from scratch. On large benchmarks, cut-line selection requires 5% of ECO-system runtime time whereas min-cut partitioning generally requires 50% or more of ECO-system runtime.

Handling macros and obstacles. With the addition of macros, the flow of top-down placement becomes more complex. We adopt the

[1] We do not assume that the initial placement was produced by a min-cut algorithm.

technique of "floorplacement" which proceeds as traditional placement until a bin satisfies criteria for block-packing [29, 32]. If the criteria suggest that the bin should be packed rather than partitioned, a fixed-outline floorplanning instance is induced from the bin where macros are treated as hard blocks and standard cells are clustered into soft blocks. The floorplanning instance is given to a Simulated Annealing-based floorplanner to be solved. If macros are placed legally and without overlap, they are considered fixed. Otherwise, the placement bin is merged with its sibling bin in the top-down hierarchy and the merged bin is floorplanned. Merging and re-floorplanning continues until the solution is legal.

We add a new floorplanning criterion for our legalization technique. If no macros in a placement bin overlap each other, we generate a placement solution for the macros of the bin to be exactly their placements in the initial solution. If some of the macros overlap with each other, we let other criteria for floorplanning decide. If block-packing is invoked, we must discard the placement of all cells and macros in the bin and proceed as described in [32].

During the cut-line selection process, some cut-line locations are considered invalid — namely those that are too close to obstacle boundaries but do not cross the obstacles. This is done to prevent long and narrow slivers of space between cut-lines and obstacle boundaries. Ties for cut-lines are broken based on the number of macros they intersect. This helps to reduce overfullness in child bins allowing deeper partitioning, which reduces runtime.

V. Using ECO-system in High-level and Physical Synthesis

We extend the proposed framework to offer users efficient access to the features of incremental placement described in Sections III and IV as well as provide greater user control and flexibility.

Tunable aggressiveness. ECO-system accepts or rejects derived partitioning solutions based on how much a single pass of a Fiduccia-Mattheyses partitioner can improve them. If the partitioner improves the net cut by more than a threshold percentage, the partitioning solution is rejected. This threshold can be adjusted by the user so as to prevent ECO-system from performing large changes. If a designer wants ECO-system to change the placement as little as possible, the improvement threshold can be given as 100%. Tunable aggressiveness also allows one to adjust the strength of ECO-system legalization to better correlate with the magnitude of design modifications [21].

Changing net weights. Having a legal placement facilitates more precise static timing analysis and finding timing-critical nets. To improve timing, weights are increased for nets with smallest slack, and decreased for non-critical nets. As ECO-system checks if the cut of an induced partitioning solution can be improved significantly, net weights are naturally integrated into this test. With weighted cut, ECO-system recognizes instances when replacement is in order due to the sub-optimality of the initial placement.

User-defined locality. ECO-system operates automatically on the given placement and quickly focuses on sections of overlap. It may be the case that a designer has performed optimization on only a small portion of the design. Having our algorithm run over the entire design to find this small area is potentially wasteful. Thus we allow the user or a physical synthesis tool to specify one or more regions of the placement area to apply legalization.

Satisfying density constraints. A common method for increasing the routability of a design is to inject whitespace into regions that are congested [4, 23]. One can also require a minimum amount of whitespace (equivalent to a maximum cell density) in local regions of the design to achieve a similar effect [34]. As one of ECO-system's legality checks is essentially a density constraint (checking to see if a child bin has more cell area assigned to it than it can physically fit), this legality check is easy to generalize. The new criterion for switching from using the initial placement and partitioning from scratch is based on a child bin having less than a threshold percent of relative whitespace, which is controlled by the user. Combined with user-defined locality, this allows a designer to re-tune whitespace allocation to reduce congestion in localized regions of the design.

Placing new cells and macros. The addition of macros, IP blocks and embedded memories to an already placed netlist can introduce significant overlap. Large modules may need to be fixed due to alignment constraints and will appear as obstacles. Buffer insertion is also a concern as numerous buffers may need to be inserted. There are typically few legal locations for buffer insertion, and, compounding the problem, buffers must be placed precisely to be effective.

Our current technique can accommodate newly added modules for which tentative initial placements are given. All a designer would need to do is place new modules roughly where they should go in the core, and ECO-system will find legal positions for them automatically. If new module locations are not known, they can be found with simple analytical techniques. Specifically, if an unplaced module is connected to several placed modules, an initial location for the module could be the average location of its neighbors. This does not work well, however, when a cluster of new logic is added to a design, especially in the presence of macros and obstacles. For this reason, we develop a technique to place unplaced modules within ECO-system.

To handle new modules separately, one must be able to detect them easily in a design. Some input formats allow the user to specify modules which are new with the keyword UNPLACED. For other input formats without such a keyword, ECO-system checks for modules that are placed outside of the core and marks them as being unplaced. ECO-system also tests to see if several modules are placed at exactly the same location which could indicate a cluster of new logic. Modules placed in exactly the same location, such as a default location like (0,0), are also treated as unplaced.

In each bin, if a cut-line and partitioning are derived, unplaced modules are partitioned with a separate partitioning call to assign them to child bins. If the derived partitioning is not accepted, unplaced modules are combined with the old modules, and placement continues from scratch. In this way, unplaced modules will migrate to good legal locations automatically. As the locations for unplaced modules are chosen based on current locations of all the modules in the design, the final locations of unplaced modules will likely be better than ones that were chosen based on the initial placement.

If new modules are introduced into a design and a user defines a region of the placement to work in, there is some ambiguity in what ECO-system should do with unplaced modules. All unplaced modules could be placed inside the user-specified region, or ECO-system could determine which of the unplaced modules would best be placed in the region. Determining which of the unplaced modules belong in a user-specified rectangular region requires at most four calls to a partitioner (since the region can be carved out with four geometric cut-lines), so this will still be efficient. To avoid uncertainty, the user is allowed to specify which behavior is desired.

VI. Empirical Results

We implemented ECO-system in C++ and ran it on 3.2GHz Pentium Xeon machines. For testing we use two suites of benchmarks. The first suite of benchmarks are the ICCAD 2004 IBM-MSwPins benchmarks: mixed-size netlists with non-trivial macro sizes, aspect ratios and pin offsets [32]. We placed all of the benchmarks with Capo 10 [32] and chose the best of 2 runs. Next we randomly resized the standard cells of the benchmark to simulate cell sizing such that the total area of cells would remain relatively constant. Each standard cell of the design was randomly increased or decreased in size, but no cell was decreased below the minimum cell size or increased beyond the largest cell size. This resizing results in the original Capo placement being illegal. The change in cell area and amount of overlap introduced by the resizing is shown in Table I. The resized benchmarks should have legal placements with HPWL near that of the original benchmarks since total cell area does not change appreciably. Discussions with colleagues in the industry point out that cell resizing is affected by a variety of factors, which are not as random as in our ex-

IBM-MSwPins Benchmarks	Area Ratio	Orig. Time (s)	Orig. HPWL (e6)	Overlap	Capo 10 Legalizer [32]			ECO-system		
					Time (s)	HPWL (e6)	Ratio	Time (s)	HPWL (e6)	Ratio
ibm01	0.9982	248	2.48	7.35%	1.27	2.57	1.0371	37.4	**2.46**	0.9913
ibm02	1.0008	463	5.12	5.56%	2.15	5.28	1.0328	65.6	**5.11**	0.9974
ibm03	1.0011	661	7.58	5.83%	15.9	7.99	1.0543	130	**7.56**	0.9978
ibm04	0.9990	728	8.61	8.13%	11.3	9.03	1.0482	135	**8.65**	1.0046
ibm05	1.0017	593	10.14	13.54%	0.13	10.25	1.0114	110	**10.20**	1.0057
ibm06	1.0018	846	6.78	7.36%	10.5	7.10	1.0469	123	**6.81**	1.0046
ibm07	0.9997	1213	11.63	9.61%	16.4	12.16	1.0455	167	**11.65**	1.0016
ibm08	1.0029	1492	13.42	8.50%	7.36	13.73	1.0232	192	**13.49**	1.0048
ibm09	1.0025	1492	14.96	8.14%	14.8	16.06	1.0732	249	**14.91**	0.9966
ibm10	0.9997	2476	31.79	4.53%	119	32.62	1.0260	384	**31.38**	0.9871
ibm11	0.9993	2067	21.43	8.48%	26.3	22.56	1.0529	317	**21.50**	1.0031
ibm12	0.9996	2903	38.52	5.91%	50.6	39.20	1.0175	345	**37.63**	0.9768
ibm13	1.0014	2667	27.30	7.94%	55.3	28.61	1.0478	494	**27.35**	1.0018
ibm14	1.0002	4954	40.00	13.49%	38.3	41.67	1.0417	594	**40.45**	1.0113
ibm15	1.0016	6241	53.72	10.85%	63.1	56.48	1.0514	1288	**54.48**	1.0142
ibm16	0.9997	7232	61.12	9.19%	36.2	62.74	1.0264	734	**61.08**	0.9993
ibm17	0.9987	7558	70.52	14.09%	36.0	73.09	1.0365	807	**71.09**	1.0081
ibm18	1.0017	6897	46.46	15.91%	13.7	48.11	1.0354	733	**47.05**	1.0128
Average	1.0005						1.0393			1.0010
ISPD05 Benchmarks	Area Ratio	Orig. Time (s)	Orig. HPWL (e6)	Overlap	Capo 10 Legalizer [32]			ECO-system		
					Time (s)	HPWL (e6)	Ratio	Time (s)	HPWL (e6)	Ratio
adaptec1	1.0004	9403	83.87	18.17%	1020	88.81	1.0589	1627	**84.27**	1.0047
adaptec2	1.0012	9978	87.31	16.83%	1246	91.48	1.0477	1731	**88.89**	1.0181
adaptec3	1.0004	26937	231.17	17.37%	3090	240.44	1.0401	4579	**225.12**	0.9738
adaptec4	1.0005	29266	187.65	16.81%	1775	194.89	1.0386	3741	**189.92**	1.0121
bigblue1	1.0005	10752	101.96	15.62%	1.6	104.77	1.0276	1421	**101.72**	0.9976
bigblue2	0.9994	27902	159.08	16.15%	1238	164.21	1.0322	5064	**158.31**	0.9952
bigblue3	0.9999	69498	414.29	15.69%	4169	445.95	1.0764	11083	**391.35**	0.9446
bigblue4	1.0006	118741	884.39	15.58%	953	903.81	1.0220	13501	**876.89**	0.9915
Average	1.0004						1.0428			0.9920

TABLE I

OVERLAP LEGALIZATION ON THE IBM-MSWPINS [32] AND ISPD05 CONTEST BENCHMARKS [28]. "AREA RATIO" REPRESENTS THE CHANGE IN TOTAL CELL AREA AFTER RESIZING. OVERLAP IS MEASURED AS A % OF THE TOTAL MOVABLE CELL AND MACRO AREA. ECO-SYSTEM REQUIRES SIGNIFICANTLY MORE RUNTIME THAN THE CAPO 10 LEGALIZER [32], AND APPROXIMATELY 14% OF THE ORIGINAL PLACEMENT TIME. ECO-SYSTEM INCREASES HPWL BY 0.10% ON AVERAGE WHILE THE CAPO 10 LEGALIZER INCREASES HPWL BY 3.93% ON THE IBM-MSWPINS BENCHMARKS. ECO-SYSTEM *decreases* HPWL BY 0.80% ON AVERAGE WHILE THE CAPO 10 LEGALIZER INCREASES HPWL BY 4.28% ON THE ISPD05 CONTEST BENCHMARKS.

periments. On the other hand, our technique is similar to real resizing in that it creates local areas of high cell overlap and is reasonable. On average, our resizing introduces 9% overlap by cell area (and more when there are fixed obstacles in the design) which is greater than what's typically observed while resizing VLSI circuits.

We compare ECO-system to the legalizer of Capo 10, and the results are summarized in Table I. We use a constant improvement threshold for ECO-system (see Figure 1, line 11) near 100% for all benchmarks to minimize changes to the placement. The Capo legalizer runs quickly and produces legal placements, but it increases HPWL by 3.93% on average. ECO-system takes less than 14% of the original placement time, and only increases HPWL by 0.10% on average. We have also varied the amount of overlap introduced into these benchmarks by reducing the number of cells affected by our sizing. We find that HPWL is mostly unaffected (HPWL generally changes by less than 0.5%) by increasing amounts of overlap for these designs.

The second set of benchmarks are from the ISPD 2005 Placement Contest [28]. They are a standard cell benchmark suite with nontrivial fixed obstacles throughout the placement area [28]. We placed all of the benchmarks with APlace 2.04 [22] (the winning placer of the contest) and randomly resized the standard cells of the benchmark in the same way as the IBM-MSwPins benchmarks. The change in cell area and amount of overlap introduced by the resizing is shown in Table I. A comparison of ECO-system to the legalizer of Capo 10 is summarized in Table I. The Capo legalizer runs 40% faster than ECO-system, but increases HPWL by 4.28% on average. ECO-system takes 14% of the original placement time, and *decreases* HPWL by 0.80%. Figure 4 depicts the benchmark adaptec3 before cell resizing and after legalization with ECO-system. ECO-system's placement is similar to the original APlace 2.04 placement and does not move the majority of cells far from their original locations. The average displacement per cell is 0.3% of the half-perimeter of the design which is an order of magnitude less than WSA's displacements [23,24]. Only 2.7% of the cells have nontrivial displacements.

Lastly, we compare ECO-system to the APlace 2.04 legalizer on APlace 2.04 global placements on the ISPD05 Contest benchmarks. Analytical placement techniques generally produce a significant amount of overlap on the contest benchmarks because of the numerous fixed obstacles in the core region. This can be seen in Table II as the APlace 2.04 global placements have approximately 30% or more overlap. APlace 2.04's legalizer generally increases HPWL by 4.91% while our legalizer produces an increase of only 3.67% on average. In addition, ECO-system is 3x faster than APlace's legalizer.

VII. CONCLUSIONS

Our main contribution is ECO-system — an algorithmic framework designed to interface a wide variety of circuit optimizations with their physical environment. This framework offers, for the first time in the literature, a strong and robust legalizer that can handle a broad range of modern placement instances with movable macros, fixed obstacles, etc. ECO-system automatically focuses on regions of the layout and sections of the netlist that require changes, and performs optimization of adequate strength in each case. ECO-system can be combined with an external global placer invoked when particularly large changes are required. It can also be used in incremental re-synthesis, in high-level and physical synthesis optimizations, and several other contexts.

ECO-system includes all detail placement methods implemented in Capo [29,32–34], and can similarly be grafted onto other top-down placers, such as BonnPlace [37], PolarBear [12] or NTUPlace [20], by performing a one-pass Fiduccia-Mattheyses test. ECO-system can act like the WSA technique [23], and can invoke any black-box global placement algorithm when it decides that a particular bin must be replaced from scratch.

The definitive success of ECO-system in legalizing APlace placements (Table II) allows to answer a long-standing question in placement — whether the slicing structure of min-cut placements costs them HPWL. Given that the placements produced by ECO-system are largely slicing, the answer appears negative.

Benchmark	Orig. Time (s)	Illegal HPWL (e6)	Overlap	APlace 2.04 Legalizer [22]			ECO-system		
				Time (s)	HPWL (e6)	Ratio	Time (s)	HPWL (e6)	Ratio
adaptec1	7569	81.05	34.74%	1346	**83.87**	1.0348	1730	84.84	1.0467
adaptec2	6062	94.22	47.25%	2543	101.64	1.0788	2042	**99.47**	1.0558
adaptec3	15849	211.13	47.12%	11495	231.17	1.0949	4500	**227.32**	1.0767
adaptec4	15404	197.24	36.78%	15271	206.23	1.0456	4132	**203.24**	1.0304
bigblue1	8265	100.51	28.53%	2486	**101.96**	1.0144	1804	105.14	1.0461
bigblue2	13650	154.51	30.15%	14252	159.08	1.0296	5183	**156.63**	1.0137
bigblue3	30624	385.40	41.06%	38873	414.29	1.0750	13708	**388.46**	1.0079
bigblue4	61932	865.03	32.01%	56809	884.39	1.0224	14910	**881.04**	1.0185
Average						1.0491			1.0367

TABLE II

OVERLAP LEGALIZATION OF APLACE 2.04'S [22] GLOBAL PLACEMENTS OF THE ISPD05 CONTEST BENCHMARKS [28]. OVERLAP IS MEASURED AS A % OF THE TOTAL MOVABLE CELL AREA. ECO-SYSTEM PRODUCES LEGAL SOLUTIONS WITH NEARLY THE SAME OR BETTER HPWL THAN APLACE 2.04'S LEGALIZER. APLACE'S LEGALIZER INCREASES HPWL BY 4.91% WHILE ECO-SYSTEM INCREASES HPWL BY ONLY 3.67%. ECO-SYSTEM IS FASTER ON 7 OF THE 8 BENCHMARKS AND 3X FASTER THAN APLACE'S LEGALIZER OVERALL.

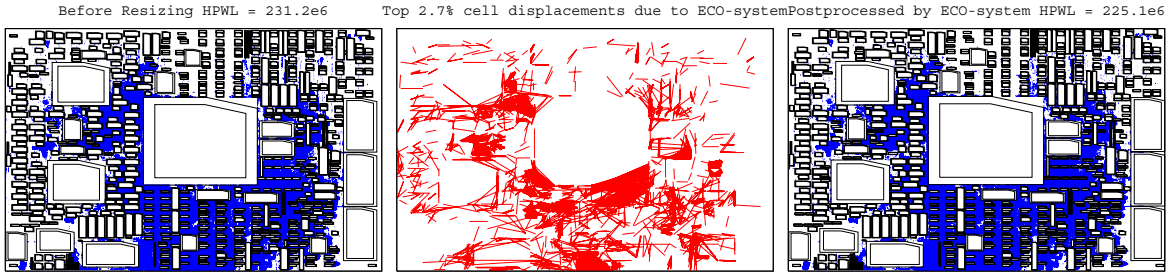

Fig. 4. When applied to resized netlist, ECO-system produces a placement (right) similar to the original placement (left). Fixed objects are outlined in double black lines. The largest cell displacements are shown in red (center). Only displacements larger than 1.5% of the half-perimeter of the design are shown. Average displacement is 0.3% of the half-perimeter. The majority of the large displacements form around the corners of the large, fixed obstacles. Many of these large displacements appear to be clustered, indicating small groups of modules transported to another region of the core or spread to accommodate area increases.

We have analyzed requirements for an ECO placement tool and implemented an interface based on ECO-system applicable to high-level and physical synthesis, allowing the designer to add and remove nets and cells from a design, reallocate whitespace, resize cells and re-weight nets while retaining control of the amount of change performed by ECO-system.

REFERENCES

[1] C. J. Alpert, G.-J. Nam, P. Villarrubia and M. C. Yildiz, "Placement Stability Metrics," *ASPDAC*, pp. 1144-1147, January 2005.

[2] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-blocks Using Floorplanning and Standard-Cell Placement", *ISPD*, pp. 12-17, 2002.

[3] S. N. Adya and I. L. Markov,"Fixed-outline Floorplanning: Enabling Hierarchical Design", *IEEE Trans. on VLSI*, vol. 11, no. 6, pp. 1120-1135, December 2003. (*ICCD 2001*, pp. 328-334).

[4] S. N. Adya, I. L. Markov and P. G. Villarrubia, "On Whitespace and Stability in Mixed-Size Placement," to appear in *Integration: the VLSI Journal*, 2006.

[5] A. Agnihotri et al., "Mixed Block Placement via Fractional Cut Recursive Bisection," *IEEE TCAD*, vol. 24, no. 5, pp 748-761, 2005. (*ICCAD* 2003, pp. 307-310).

[6] U. Brenner and J. Vygen, "Legalizing a Placement With Minimum Total Movement," *IEEE TCAD*, vol. 23, no. 12, pp. 1597-1613, 2004. (*ISPD* 2004, pp. 2-9).

[7] A. E. Caldwell, A. B. Kahng, I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE TCAD*, vol. 19, no. 11, pp. 1304-1314, 2000. (*ISPD* 1999, pp. 90-96).

[8] C.-C. Chang, J. Cong, D. Pan and X. Yuan, "Multilevel Global Placement with Congestion Control," *IEEE TCAD*, vol. 22, no. 4, pp. 395-409, 2003.

[9] C.-C. Chang, J. Cong and X. Yuan, "Multi-Level Placement for Large-Scale Mixed-Size IC Designs," *ASPDAC*, pp. 325-330, 2003.

[10] T. C. Chen, Y. W. Chang and S. C. Lin, "IMF: Interconnect-Driven Multilevel Floorplanning for Large-Scale Building-Module Designs," *ICCAD*, pp. 159-164, November 2005.

[11] J. Cong and M. Sarrafzadeh, "Incremental Physical Design", *ISPD*, pp. 84-92, 2000.

[12] J. Cong, M. Romesis and J. Shinnerl, "Robust Mixed-Size Placement Under Tight White-Space Constraints," *ICCAD*, pp. 165-173, 2005.

[13] J. Cong and M. Xie, "A Robust Detailed Placement for Mixed-Size IC Designs," *ASPDAC*, pp. 188-194, 2006.

[14] K. Doll, F. M. Johannes and K. J. Antreich, "Iterative Placement Improvement By Network Flow Methods," *IEEE TCAD*, vol. 13, no. 10, pp. 1189-1200, Oct. 1994.

[15] W. Donath et al., "Transformational Placement and Synthesis", *DATE*, pp. 194-201, 2000.

[16] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *DAC*, pp. 175-181, June 1982.

[17] R. Goering, "Cadence CTO: CAD 'Foundations' Must Change," EETimes, April 11, 2006, `http://www.eetimes.com/showArticle.jhtml?articleID=185300099`

[18] D. Hill, "Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design," *US Patent 6370673*, April 2002.

[19] S. W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement," *ICCAD*, pp. 165-170, 2000.

[20] Z.-W. Jiang et al., "NTUPlace2: A Hybrid Placer Using Partitioning and Analytical Techniques," *ISPD*, pp. 215-217, 2006.

[21] A. B. Kahng and S. Mantik, "On Mismatches Between Incremental Optimizers and Instance Perturbations in Physical Design Tools," *ICCAD*, pp. 17-22, 2000.

[22] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer," *IEEE TCAD*, vol. 25, no. 5, pp. 734-747, May 2005.

[23] C. Li, M. Xie, C. K. Koh, J. Cong and P. H. Madden, "Routability-driven Placement and White Space Allocation," *ICCAD*, pp. 394-401, 2004.

[24] C. Li, C.-K. Koh and P. H. Madden, "Floorplan Management: Incremental Placement for Gate Sizing and Buffer Insertion," *ASPDAC*, pp. 349-354, January 2005.

[25] L. Luo, Q. Zhou, X. Hong and H. Zhou, "Multi-stage Detailed Placement Algorithm for Large-Scale Mixed-Mode Layout Design," *ICCSA*, pp. 896-905, 2005.

[26] T. Luo, H. Ren, C. J. Alpert and D. Pan, "Computational Geometry Based Placement Migration," *ICCAD*, pp. 41-47, 2005.

[27] M. D. Moffitt, A. N. Ng, I. L. Markov, M. E. Pollack, "Constraint-driven Floorplan Repair," *DAC*, 2006.

[28] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter and M. Yildiz, "The ISPD2005 Placement Contest and Benchmark Suite," *ISPD*, pp. 216-220, 2005.

[29] A. N. Ng, I. Markov, R. Aggarwal and V. Ramachandran, "Solving Hard Instances of Floorplacement," *ISPD*, pp. 170-177, April 2006.

[30] M. Pan, N. Viswanathan and C. Chu, "An Efficient and Effective Detailed Placement Algorithm," *ICCAD*, pp. 48-55, 2005.

[31] H. Ren, D. Z. Pan, C. J. Alpert and P.Villarrubia, "Diffusion-based Placement Migration," *DAC*, pp. 515-520, 2005.

[32] J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov, "Min-cut Floorplacement," *IEEE TCAD*, , vol. 25, no. 7, pp. 1313-1326, 2006.

[33] J. A. Roy, J. F. Lu and I. L. Markov, "Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement," *ISPD*, pp. 78-85, April 2006.

[34] J. A. Roy, D. A. Papa, A. N. Ng, I. L Markov, "Satisfying Whitespace Requirements in Top-down Placement," *ISPD*, pp. 206-208, April 2006.

[35] N. Selvakkumaran and G. Karypis, "Theto - A Fast, Scalable and High-quality Partitioning Driven Placement Tool," Technical report, Univ. of Minnesota, 2004.

[36] N. Viswanathan, M. Pan and C. Chu, "FastPlace 2.0: An Efficient Analytical Placer for Mixed-Mode Designs," *ASPDAC*, pp. 195-200, 2006.

[37] J. Vygen, "Algorithms for Large-Scale Flat Placement," *DAC*, pp. 746-751, 1997.