Incremental Placement for Layout-Driven Logic Synthesis for Timing Closure

Andrew C. Ling, Member, IEEE, Deshanand P. Singh, and Stephen D. Brown, Member, IEEE

Abstract—This paper describes a set of incremental placement algorithms necessary after layout-driven optimizations for timing closure. The primary power of our approach is its ability to legalize a disrupted placement, while maintaining circuit performance gains obtained from netlist optimizations. Two incremental placement approaches are shown in detail: a single-stage incremental placement algorithm targeted for FPGAs that legalizes a placement while maintaining the circuit performance throughout the legalization process; and a two-stage approach targeted for structured ASICs that performs a fast placement legalization followed by a performance recovery step. When combined in a full layout-driven logic synthesis flow for FPGAs and structured ASICs, these approaches can improve circuit performance by 10-15% on average.

Index Terms—Incremental Placement, Physically Driven Synthesis, FPGA, Structured ASICs.

I. INTRODUCTION

As interconnect delay is now the dominating factor in determining the overall path delay in ASICs and FPGAs [1], layout-driven logic synthesis late in the design flow is an effective solution to achieve timing closure [2]. Unlike traditional synthesis, layout-driven synthesis applies logic optimizations that are tightly coupled with placement. This is advantageous since the logic netlist can be optimized using layout information derived from the placement. However, these optimizations will disturb the current placement resulting in architectural violations. Thus, layout-driven synthesis must be followed by a legalization step to restore the placement. This important step is known as incremental placement (ICP).

As an illustration of ICP, consider Fig. 1. In Fig. 1, four subcircuits are shown in conjunction with their placement picture. Assume that in the placement grid, each large block can contain at most four logic elements or registers as indicated by the smaller blocks 1. Fig. 1 illustrates a common optimization which attempts to shorten a given path using retiming [3]. Assuming that delay is proportional to interconnect length, shortening critical paths will improve the circuit performance. This is highlighted in Fig. 1(b) where register d is pushed ahead of logic element c. Although this shortens the path between logic element c and register d, the current placement is now illegal since the pushed register is assigned to an illegal grid location. In order to create a legal placement, the cells surrounding register d must be displaced such that the resulting placement is legal as shown in Fig. 1(c) and Fig. 1(d).

The previous example illustrates the general flow of layout-driven logic synthesis illustrated in Fig. 2. As Fig. 2 shows, layout-driven synthesis is an iterative flow where netlist optimizations, which disrupts the current placement, is followed by a legalization step (ICP). This feedback loop continues until some cutoff criterion is reached. The netlist optimizations step will restructure the netlist and return a set of preferred locations for all cells, P. ICP will attempt to map the preferred locations to a new set of locations, P → M, such that the resulting mapped locations are legal. This is highlighted in the previous example where Fig. 1(b) represents the set of preferred locations P and Fig. 1(d) represents the final mapped locations M. Assuming that the function T(L) represents the critical path delay of the circuit with placement L, ICP attempts to minimize the cost function T(M) − T(P). In other words, an effective ICP algorithm legalizes the placement such that the mapping from P to M is minimally disruptive. If not, the resulting placement will not achieve the desired performance intended by the layout-driven optimizations. For

1 We will use the term logic element and cell interchangeably when the meaning is obvious from the context.
example, in Fig. 1(c), although the critical path between logic element c and register d is removed, a new critical path may have been formed in paths crossing through disrupted logic elements. Routing area should also be taken into account since an excessive increase in routing area will prevent the desired timing from being achieved. We represent this with the cost function $A(L)$ which represents the routing cost. Thus, incremental placement should attempt to legalize the placement while minimizing the cost function,

$$T(M) - T(P) + A(M) - A(P) \quad (1)$$

In this work, we present several approaches to ICP both for FPGAs and structured ASICs. We start by describing a single-stage FPGA ICP algorithm found in the commercial CAD tool QuartusII [4]. This version of ICP deals with the large number of architectural constraints found in an FPGA, and evaluates legalization moves individually to minimize the cost function shown in Eq. 1. This proves to be quite powerful and when combined with several layout-driven optimization techniques, we can increase circuit operating frequency by 10-15% on average [2].

Following this, we show how we can modify the FPGA ICP process to structured ASICs such that we can move cells on a larger scale [5]. This is necessary to improve the scalability of the ICP algorithm since structured ASICs contain an order of magnitude more logic elements than when compared to FPGAs. We will show that our ICP adaptations for structured ASICs can reduce the runtime of the ICP algorithm by 50% without hurting performance.

The rest of the paper is organized as follows: Section II describes several FPGA architectural features that makes ICP difficult, along with constraints dealt with structured ASICs. Section III describes both the ICP algorithm for FPGAs and structured ASICs. Finally, section IV shows our results followed by the conclusion in section V.
The term “location” in the FPGA domain is a coordinate that uniquely identifies a particular CLB \(^2\) and a given CLB contains many slots to hold individual logic elements. Thus a location may contain \(n_E\) slots to hold \(n_E\) logic elements. More specific locations could be used, but in most commercial architectures the logic elements within CLBs all have similar delay characteristics.

The architecture described above is fairly simple in structure, but the ICP algorithm must operate on logic blocks with an arbitrary number of complex constraints. This property allows for architectural exploration of many logic block variations. For example, we have been able to study a logic block with increased register flexibility \([9]\) that significantly reduces the area penalties associated with sequential retiming optimizations. The flexible nature of the ICP algorithm allowed us to easily target this block.

### B. Structured ASICs

Unlike FPGAs, the architectural constraints considered for structured ASICs is limited to overlaps. Thus, the goal of ICP in the structured ASIC domain is to remove all overlaps between cells while optimizing for area and timing.

Structured ASICs differ from full-custom chips since logic cells are predefined to form basic building blocks of the chip. These building blocks are used to form more complex cells which are uniform in height, but can vary in width. From a conceptual point of view, valid placement regions form a grid where each location of the grid can occupy at most one cell. A single cell can occupy multiple adjacent grid coordinates due to their varying widths. Fig. 4 shows an example legal placement.

![Fig. 4. Legal placement of cells on a grid.](image)

### C. Previous Work

There has been several works on various techniques tackling ICP for FPGAs. In \([10]\), a ripple approach is taken where locations containing excess logic elements are connected to locations containing free slots through a series of directed edges to form a graph. Logic elements can be moved along the direction of the graph edges to create a “rippling” affect where logic elements are moved between adjacent locations to fix illegalities in the placement. One concern with this technique is that free slots may be difficult to find in some cases or be very far from the locations containing illegalities. Thus, the series of ripple moves might be large which is quite disruptive to the original placement.

More recently, in \([11]\) the author extends a simulated annealing based placer \([8]\) to handle incremental placement where their incremental placement step is 60x faster than the full placer. However, their approach handles moves at the CLB level and is intended for large ECO changes. This is too aggressive for our purposes where our layout-driven optimizations will change the netlist structure by a small amount.

In \([12]\) an ICP algorithm for timing driven optimizations is shown. Here, initial placements of logic elements are found using a quadratic based placement algorithm, resulting in overlaps. Following this, logic elements are moved one by one to remove all overlaps. These fine grain approaches to ICP work well for FPGAs, but are not suitable for ASICs since the number of logic cells in the ASIC domain is in the order of millions. In general, placement algorithms for ASICs are tackled from a global perspective to ensure the placer runs quickly. In \([13]\), the authors present an incremental placer based on floorplan management. They treat ICP as a resource allocation problem where congested areas need much more space resources than non-congested areas. In order to solve this problem, the authors propose a method to move space resources from non-congested areas to congested areas. This method guarantees that the relative position of cells remains the same, thus not affecting the original placement quality. A problem of this approach is that it assumes that the placement area is homogeneous. For heterogeneously structured ASICs, some regions are reserved for specialized logic, such as memory; thus this ICP technique is not applicable to heterogeneously structured ASICs.

In \([14]\), an algorithm called FastPlace is described. In this work, the authors use a cell shifting technique to fix any overlaps created by a quadratic placer. This greatly speeds up the convergence of the quadratic placer to a final legal solution. Since cell shifting must be interleaved with a quadratic placement algorithm, all cells will be disturbed, including those cells that are already legally placed. This is not desirable in ICP since ICP should move only those cells that are illegal to maintain the quality of the original placement.

In \([15]\), the authors present a placement tool called Domino. Like FastPlace, this work uses a quadratic placer \([16]\) to get a loose placement of the cells. This is followed by detailed placement where the placement is legalized using network flow. However, as in \([14]\), the detailed placement step disrupts all cells and thus cannot be applied to ICP. An improvement to this is presented in \([17]\). Here, a network flow approach is also used to legalize a globally optimized placement, however, they use critical path delay information and focus on optimizing critical paths.

In \([18]\), a transportation based algorithm is used to place cells. Their algorithm is based on partitioning to spread and place cells. This is also not suitable for incremental placement since it causes too much disruption to the existing placement picture. In later sections, we will show how we can adapt previous placement techniques to ICP in structured ASICs while minimizing the disruption of the original placement.

\(^2\)the term “location” will be different in the structured ASIC domain since CLBs do not exist
III. ICP ALGORITHM

When designing the ICP algorithms for both the FPGA and structured ASIC domain, two major assumptions are used as follows:

- The number of architectural constraints violated by the preferred locations will be relatively small. This assumption is well founded as experiments indicate that the number of additional logic elements added to the netlist is typically small and the optimizations themselves have some capability to produce sensible preferred locations (i.e., not assigning all of the newly synthesized logic to a single CLB).
- An architecturally feasible set of mapped locations is relatively close to the set of preferred locations in the solution space. This assertion is implied from the previous assumption.

A. FPGA ICP

Considering the assumptions described previously, the FPGA ICP algorithm was designed based on an iterative improvement strategy. The first step assigns mapped locations to be equal to the preferred locations: \( \forall e \in E, (m_x(e), m_y(e)) = (p_x(e), p_y(e)) \). The architectural violations are removed by iterating on this starting solution. Every move is evaluated by a cost function that guides the reduction of architectural violations while ensuring minimal disruption. Recall that iterative improvement algorithms have a general structure that is depicted by the pseudocode in Fig. 5.

```
1 S = INITIAL SOLUTION()
2 do
3     S' = PROPOSEMOVE(S)
4     if(C(S') < C(S))
5         S' = S
6     end if
7 until(exitCriterion = true)
```

Fig. 5. Iterative Improvement Algorithm

When deciding on a move in `PROPOSEMOVE()`, the types of moves are randomly chosen and are defined as follows:

- move to fanin/fanout - moves a logic element to a CLB containing its fanin or fanout. A fanin (fanout) is a logic element whose output (input) connects to the input (output) of the current logic element in question.
- move to sibling - moves a logic element to a CLB containing one of its siblings. A sibling is a logic element that shares a fanin with the current logic element being moved.
- move to neighbour - move to a neighbouring CLB
- move to free space - move to a random CLB with available space
- move in direction of critical path - move to a CLB in the direction of CLBs containing critical logic elements connected to the current logic element being moved.

Since the ICP algorithm described above occurs in one step, the placement quality must be maintained throughout the legalization process. This requires a cost function, \( C \), which controls moves such that critical paths or near critical paths do not increase in delay. Furthermore, we must propose moves such that we can ensure routing will be possible after the placement is legalized. Thus, our cost function used to evaluate moves is a weighted sum of three components: a CLB legality component, a timing component, and a wirelength component. These factors can be combined together in the following cost equation.

\[
C = K_L \ast \text{ClusterCost} + K_T \ast \text{Timing} + K_W \ast \text{Wirelen} \quad (2)
\]

1) CLB Legality Cost: The legality cost associated with each CLB \( CL_i \) can be calculated as shown in Eq. 3.

\[
\text{ClusterCost}(CL_i) = kE_i \ast \text{legality}(CL_i, n_E) + kI_i \ast \text{legality}(CL_i, n_I) + kR_i \ast \text{legality}(CL_i, n_R) + kO_i \ast \text{legality}(CL_i, n_O) + kC_i \ast \text{legality}(CL_i, n_C) \quad (3)
\]

The legality \( (CL_i, \ldots) \) function returns a measure of legality for a particular constraint. A value of 0 indicates legality, while any positive value is proportional to the amount to which the constraint has been violated. For example the function legality \( (CL_i, n_I) \) evaluates if the CLB \( CL_i \) has a feasible number of inputs. A viable return value would be \( \min \{n_I - \text{maxInputs}, 0\} \). The exact nature of the function is not important but it must provide enough information to guide the algorithm to reduce the number of violations. This characteristic is extremely important when several logic elements must be moved to create a legal CLB configuration.

The weighting coefficients \( kE_i, kI_i, kR_i, kO_i, kC_i \) are all initially set to 1 for every CLB \( CL_i \) in the target device. The usefulness of the constants will be described in section III-A.4 which discusses Directed Hill-climbing.

2) Timing Cost: The timing component of Eq. 2 is based upon the cost used by the VPR [8] placer. This cost is shown in Eq. 4.

\[
TC_{VPR} = \sum_{c} \text{crit}(c) \ast \text{delay}(c) \quad (4)
\]

This function sums up the criticality and delay product of all connections \( c \) in the circuit. As a result, this encourages critical connections to reduce delay, while allowing non-critical connections to optimize wirelength and other optimization criteria. This aggressive cost function can cause non-critical connections to become critical. This phenomenon is shown in Fig. 6(a) that plots the critical path delay of a particular benchmark circuit vs. iterations of the incremental placer. There is a significant amount of oscillation because the function in Eq. 4 reduces the delays of critical connections, but as a side-effect the non-critical connections now become critical. This behavior is controlled in VPR through the use of a range-window that limits the motion of logic blocks to a localized neighborhood whose size correlates with the temperature of the anneal. Unfortunately, a static range limitation for logic element moves in ICP significantly impairs
our ability to make critical path moves that reduce architectural violations. Furthermore, the ICP algorithm is not intended to actively improve the critical path delay of the circuit after the netlist optimization, but rather to preserve the delay by moving non-critical logic as little as possible. Hence we reduce the aggressive behaviour of Eq. 4 with the concept of a dynamic range window that is implemented as a damping component of the timing cost function.

\[
TC_{DAMP} = \sum c \text{max}(\text{delay}(c) - \text{maxdelay}(c), 0.0) \\
\text{maxdelay}(c) = \text{delay}(c) + \alpha * \text{slack}(c)
\] (5)

Consider the function shown in Eq. 5. This function penalizes any connection \( c \) whose delay \( \text{delay}(c) \) exceeds a certain maximum value \( \text{maxdelay}(c) \). Any delay value less than the \( \text{maxdelay} \) value is not costed. This step function characteristic allows the freedom to make arbitrary moves along the plateau defined by the maximum delays. These \( \text{maxdelay} \) values are updated every time a timing analysis of the circuit is executed and are controlled by the slack on the connection being considered. The parameter \( \alpha \) determines how much of a connection’s slack will be allocated to the delay growth of the connection. Thus the plateau is defined by the connection slack so that connections with large amounts of slack are free to move large distances in order to resolve architectural violations, while small slack values are relatively confined. Values of \( \alpha \) ranging from 0.35 - 0.55 produce results that effectively control the critical path oscillation problem when the Eq. 6 is used as the new timing cost.

\[
TC = TC_{VPR} + k_{DAMP} * TC_{DAMP}
\] (6)

Fig. 6(b) shows the result of the damping cost on the critical path oscillation problem. The magnitude of the oscillations has been significantly reduced and hence the final critical path delay is also much better as the algorithm never had to recover from a sequence of poor moves that degraded the critical path. It is also evocative to see that the final critical path delay varied only slightly from the first iteration, where \( M = P \), indicating the \( T(M) \approx T(P) \).

Fig. 7 shows a graphical description of the dynamic cost. In this example, two connections are affected by the movement of logic element \( x \). For the connection \( c_1 \), a circle is drawn indicating the range of locations where \( \text{delay}(c_1) \leq \text{maxdelay}(c_1) \). This condition is also graphically depicted for \( c_2 \). The intersection of these two circles is the free zone for logic element \( x \). It may move to any point within these regions without being penalized for exceeding the maximum delays for any of its affected connections. Hence the actual range window varies in size and shape depending on the delay characteristics of the affected connections. The boundaries of the window are also “soft” because logic elements are not confined within their free zone, but rather they are heavily penalized. This freedom is essential for resolving difficult constraints.

3) Wire-length Cost: In order to maintain routability of the circuit, wirelength is taken into account during each ICP move. Fig. 8 shows a high-level description of how the wirelength is monitored. Horizontal and Vertical cut-lines are placed in each horizontal and vertical channel of the target device. These cut-lines are used to measure the number of routing wires that intersect each cut. The cut-lines across the channels of the chip provide a method to measure congestion by finding the regions that contain the largest number of routing wires. This technique helps to ensure that the resulting placement does not contain localized congested areas that can cause circuitous routes.
A net intersects a cut line if the cut line intersect the bounding box formed by the net where a bounding box is defined as the minimum sized rectangle which encapsulates all pins of the net. The total number of routing wires that intersect a particular cut is estimated by finding all the nets which intersect a particular cut-line and summing the average crossing-count for each of these nets. The average crossing-count for a net can be computed using the techniques described in [19] using the following formula:

\[
\text{CrossingCount}(\text{net}) = q(\text{NumCLBlockPins}(\text{net}))
\]  

(7)

The function \( q \), which is presented in [19], gives the number of discrete crossing-counts as a function of net pin count. The value \( q \) is proportional to the number of pins in the net and is used to compensate for the extra routing resources necessary for nets with more than three terminals. The argument to the function \( q \) is the number of CLB pins used to wire the net, as opposed to logic element pins originally used in [19]. We use CLB pins since we are trying to estimate the amount of general purpose interconnect used by the circuit mapping (i.e., inter-CLB routing). For example, a net containing 8 logic elements may be packed into two CLBs so that the net has only 2 pins (\( q(2) = 1 \) as defined by [19]).

4) Hill Climbing: The algorithm described so far is essentially greedy, because only moves that improve the cost function are accepted. The drawback with this approach is that the algorithm could easily get trapped in a configuration where it cannot find moves that decrease the current cost. Consider the situation shown in Fig. 9. Every possible move attempted to resolve the architectural constraints of the center CLB results in another architectural violation. This situation is quite common for architectural violations close to the center of the chip as there are usually few available white spaces. If all architectural violations are costed in the same manner, then the algorithm described previously cannot resolve the constraint violation. Clearly the algorithm must now execute some kind of hill-climbing step to escape this local minima.

Fig. 8. Local Congestion Estimation.

Fig. 9. Trapped in a Local Minima.

Fig. 10. Basin Filling.

Fig. 10 shows the basic strategy for escaping local minima. It shows a two dimensional slice of the multi-dimensional cost function described in section III-A. The current state (shown by the leftmost circle) represents the situation shown in Fig. 9. No single move in the neighborhood of the current state finds a solution with a lower cost, so we are trapped in this state. However, the cost function itself could be modified to allow us to climb the hill. Recall that the CLB legality cost contains per-CLB weighting coefficients for each legality cost. Suppose that the weights of these coefficients were gradually increased for CLBs that have unsatisfied constraints; then the cost function itself would actually be reshaped to allow for hill climbing. This technique gives a higher weight to unsatisfied constraints that have been violated for a long time. Consider the situation again in Fig. 9. Once the weighting coefficients have been increased for the center CLB we are free to make a move to one of the adjacent CLBs allowing us to shift the violations “outward” closer to a free space.

An example of forced moves to an illegal location is shown in Fig. 11. Suppose that a register (shown in the center) is inserted into the CLB \( C_1 \) causing it to violate architectural constraints as only 4 logic elements are allowed in each CLB. In this CLB, the logic element with the least timing critical connections (upper right LE in \( C_1 \)) is moved to CLB \( C_2 \) which contains one of its fanins. \( C_2 \) now violates its architectural constraints, but the adjacent CLB \( C_3 \) contains a free logic
are incremented in proportion to the amount of violation. The filling procedure for directed hill-climbing, Constraint weights Fig. 13, is also called every \( K \) based on the amount of overuse remaining.

The process of violation shifting actively attempts to make these sequences of moves possible by proposing a move that shifts violations into regions with white space such as moving to \( C_2 \) because of the free space in CLB \( C_3 \).

5) Implementation Issues: The basic cost function and move proposal schemes have been discussed previously. Fig. 12 presents the pseudocode for the entire ICP algorithm. The algorithm simply chooses logic elements that participate in illegal CLBs and tries to move them to improve the cost function (line 2-10). Notice also that simple timing analysis (line 12) is performed every \( K \) iterations. This call updates the \( \text{maxdelay} \) and connection criticality values to reflect the current configuration. The value of \( K \) is adaptively updated in the previous section into two stages: a fast legalization step to improve the circuit performance.

The \( \text{UPDATEOVERUSECOEFF}() \) function, shown in detail in Fig. 13, is also called every \( K \) iterations to perform the basin filling procedure for directed hill-climbing. Constraint weights are incremented in proportion to the amount of violation. The parameter \( \gamma \) is called the basin fill rate and it controls the rate at which the local minima are filled. If \( \gamma \) is too large, then basins become peaks of a hill too quickly and almost any move is accepted even if it results in poor timing or wirelength because violation shifting becomes much more important than the timing or wirelength. A small value of \( \gamma \) would result in long runtimes, as basins are slowly filled.

Note that the ICP approach is similar to the Pathfinder [20] algorithm used for FPGA routing. However, in ICP the logic elements “fight” for preferred CLB locations, by negotiating legality, timing and wirelength.

B. Extension to Structured ASICs

As we will show in the results section, the previous ICP algorithm performs quite well and ensures good performance when legalizing a given placement after layout-driven synthesis optimizations. However, when adapting the single-stage ICP algorithm to structured ASICs a few drawbacks to it are apparent. First, the ICP cost function is overly complex for structured ASICs. This was necessary to accommodate the several architectural constraints found in modern day FPGAs, but is unnecessary for structured ASIC where constraints are limited to overlap. Secondly, the ICP process dominates the overall runtime during layout-driven synthesis. Both of these issues must be mitigated if the ICP algorithm can be successfully extended to structured ASICs which contain millions of cells. This is achieved by breaking the ICP algorithm described in the previous section into two stages: a fast legalization process to form a legal placement followed by a performance recovery step to improve the circuit performance.

When legalizing the placement for structured ASICs, the primary concern is the removal of overlaps between cells. Furthermore, in order to improve the scalability of the algorithm, moves must occur on a larger scale. We achieve this by creating direction vectors for several cells in one step using the transportation problem. Following this, all cells are moved in their suggested direction to remove overlaps between cells. Once all illegalities are removed, the ICP algorithm falls back to an annealing based improvement strategy to recover performance.

The high-level overview of the legalization stage is shown in Fig. 14. The algorithm starts off by identifying all illegal positions found in the placement solution and placing the locations in a queue (line 1-4). These locations identify where overlaps occur in the cell grid. After the illegality queue is formed, each illegal location is processed sequentially (line 6). A subregion is created around the current illegal location. Within this subregion, all cells are removed from the region...
creating an empty area. The removed cells are then re-placed
in the subregion one by one in such a way that most, if not
all, illegalities in the subregion are removed. This is the main
area of speedup when compared against the ICP algorithm
described in section III-A, which in contrast moves cells one
by one. It is possible for new illegalities to be created during
the re-placement process (line 10, set L). If so, these are added
to the back of the illegality queue (line 11). This ensures
that the original illegalities are processed before newly formed
illegalities.

1) Re-Placement in Subregion: After cells have been lifted
in the subregion, they must be re-placed in a manner that
removes illegalities. We use a shifting technique to accomplish
this. The shifting works by first placing a cell at its original
location (i.e., where it was lifted from). If that location is
occupied by another cell, it is shifted until a legal position
is found. If no legal position can be found within a fixed
radius from its original location, it is placed back at its original
location creating new illegalities. The shifting is guided by a
direction vector which is calculated using the transportation
problem.

2) Setting Up Transportation Problem: Formulating the
transportation problem involves creating a set of suppliers and
consumers and adding a set of relevant paths with associated
costs between these points to form a transportation graph. In
ICP, the goal is to move cells to valid locations such that all
illegalities are removed. In order to represent this problem as
a transportation problem, we need to introduce subcells. A
subcell can be thought of as any part of the cell that occupies
one grid location on the structured ASIC. Since the width
of a cell spans multiples of $x$ units on the placement grid
and has a unit height, the number of subcells in a cell is
equivalent to its width. This contrasts with FPGAs since each
grid location on the FPGA can hold a single CLB. Breaking
up cells into subcells allows the cells to be thought of as
suppliers who need to ship their subcells to a set of location
consumers, where the demand of each location is at most one
subcell. This is illustrated in Fig. 15. The heuristic used for
determining the costs of paths between a cell and location
will be described in the next section. Another consideration
when forming the transportation graph is determining which
locations will connect to which cells. One approach is to
connect all locations to every cell as in [15]. The problem
with this is that it creates a very dense transportation graph.

Since transportation solver runtimes are proportional to the
number of edges in the graph, a dense transportation graph will
take much longer to solve than a sparse graph. Furthermore,
as stated previously, ICP should minimize disruption of the
original placement. Thus, we connect cells only to locations
that are relatively close to the original location of the cell. For
example, consider the illegal placement shown in Fig. 16a.
Fig. 16b shows the transportation graph created using the
illegal placement where edge costs have been ignored for
simplicity. Notice that paths only exist between cells and
locations that are fairly close to the original location of the
connecting cell. After the transportation solver is run on the
diagram, a set of paths will be selected. The locations found
at the heads of the selected paths are used to determine a
suggested location for the cell found at the tail of the path.

3) Transportation Problem Path Costs: Since the trans-
portation problem is a cost minimization problem, the edge
costs in the transportation graph must ensure that the trans-
portation solver will find cell locations that reduce the overall
delay between cells. One estimate of delay is wirelength
where delay is proportional to wirelength. Thus, using a path
cost that represents wirelength will ensure that the overall
wirelength, and hence delay, will be minimized. The metric
we use to estimate the wirelength of a net is the half-length
perimeter [15]. The half-length perimeter is the width plus
height of the bounding box. When calculating the bounding
box, we assume that the cell pins are at the center of the cell.

---

**Fig. 14.** High-level overview of structured ASIC ICP legalizer.

**Fig. 15.** Representing incremental placement as a transportation problem.

**Fig. 16.** Incremental placement example.
Also, when splitting up cells into subcells, we assume that subcells will share the same path costs as the single cell the subcells were derived from.

The final cost of a path between a cell and a location is the sum of the half-lengths for all nets connected to the cell when placed at the given location as shown in Fig. 17. When costing the edges this way, we are assuming that the net cells connected to the current cell are stationary. For example, referring back to Fig. 17, while costing the edge costs for the center cell, we are assuming all the outer cells will not move. Although this is not true during the ICP process, we can make this assumption since we know that during ICP, cells will only move slightly since the majority of cells are already legally placed. In contrast, if cells were to move dramatically, such as during the initial stages of cell placement, our edge costs will be invalid leading to very poor placement solutions. This is a similar problem during layout-driven optimizations since they use delay values of the current placement to guide their optimizations, even though the final placement after the optimizations will be different due to ICP. Thus, even though some cells will move during ICP, the initial delay values used in the layout-driven optimizations are sufficient approximations.

\[
C = (l_1 \cdot w_1) + (l_2 \cdot w_2) + (l_3 \cdot w_3)
\]

Fig. 17. Path cost calculation.

4) Directed Replacement of Cells: Although the subcells of a single cell will usually be placed close to each other because they share the same edges and costs, there is no guarantee that subcells of a single cell will be placed adjacent to each other by the transportation solver. In order to consolidate the subcells, their center of gravity is used as the suggested location of the cell. This suggested location is used to create the directed vector needed by the guided shifting of cells as illustrated in Fig. 18a. Fig. 18b illustrates the shifting process if the original cell location is occupied. This will continue until a final valid location is found as shown in Fig. 18c. In cases where the region is extremely crowded, some cells may be pushed fairly far. We account for this by creating a radius limit for placement. If a cell cannot be legally placed by the time it reaches the radius limit from its original location, it will be placed back at its original location, creating some overlap illegalities. Any new overlap locations will be placed at the end of the illegality queue for reprocessing later on.

5) Performance Recovery: Since timing and wirelength is degraded due to the fast legalization process, a post processing step is required to recover performance losses incurred during legalization. This performance recovery step consists of a series of fast logic element swaps that is similar to the simulated annealing placement approach in VPR. This algorithm is greedy in nature and considers swaps sequentially. As a result, the performance recovery step is relatively fast when compared to the original ICP algorithm and runs in linear time with respect to circuit size.

IV. RESULTS

A. FPGA ICP

TABLE I

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#LEs</th>
<th>ΔLEs</th>
<th>ΔCP</th>
<th>ΔWl</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigkey-mcnc</td>
<td>1707</td>
<td>215</td>
<td>1.01</td>
<td>1.02</td>
</tr>
<tr>
<td>dsp-mcnc</td>
<td>1370</td>
<td>199</td>
<td>1.01</td>
<td>1.00</td>
</tr>
<tr>
<td>diffq-mcnc</td>
<td>1497</td>
<td>41</td>
<td>1.12</td>
<td>0.93</td>
</tr>
<tr>
<td>elliptie-mcnc</td>
<td>3604</td>
<td>472</td>
<td>0.93</td>
<td>0.88</td>
</tr>
<tr>
<td>frisc-mcnc</td>
<td>3556</td>
<td>400</td>
<td>1.03</td>
<td>0.95</td>
</tr>
<tr>
<td>s38417-mcnc</td>
<td>6406</td>
<td>257</td>
<td>0.94</td>
<td>0.85</td>
</tr>
<tr>
<td>tseng-mcnc</td>
<td>1047</td>
<td>83</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>he11-oc</td>
<td>3877</td>
<td>145</td>
<td>1.02</td>
<td>0.86</td>
</tr>
<tr>
<td>des-fip</td>
<td>15509</td>
<td>1190</td>
<td>1.00</td>
<td>0.88</td>
</tr>
<tr>
<td>sisc8</td>
<td>1434</td>
<td>61</td>
<td>1.05</td>
<td>1.25</td>
</tr>
<tr>
<td><strong>geomean</strong></td>
<td></td>
<td></td>
<td>1.006</td>
<td>0.951</td>
</tr>
</tbody>
</table>

Table I shows the effectiveness of incremental placement on several benchmark circuits. The circuits are initially placed followed by the sequential retiming optimization described in [3]. The number of extra logic elements introduced by the retiming optimization is shown in the column labeled ΔLEs. The ratio of the critical path (CP) after ICP mapping divided by the critical path after the netlist optimization shows our success in minimally disrupting the timing of the preferred locations. Similarly, the wirelength ratio is also shown.

These results indicate that the incremental placement algorithm is quite successful at its goal of producing minimally disruptive architecturally feasible mappings. Overall, there is only a 0.6% speed penalty and a 4.9% wirelength improvement in producing the mapping from preferred locations to mapped locations. As reported in [3], the application of post-placement
retiming in conjunction with ICP results in a 7% increase in operating frequency in comparison to pre-placement retiming. ICP was key to achieving this result. Specifically, if we did a complete re-placement instead of an incremental placement, the performance improvement due to retiming is reduced by more than 50%. This shows that ICP allows for better convergence for circuit optimizations that run after placement. In the case of complete re-placement, there is no guarantee that the optimizations based on the previous placement will correspond to the new re-placement.

![Graph showing ICP Runtime](image)

**Fig. 19. ICP Runtime.**

We found that our ICP algorithm runs almost eight times faster than a full annealing based placement on the largest circuit. However, this comparison is not entirely fair because of many implementation differences. Fig. 19 shows a graph depicting the number of moves ICP uses to resolve illegality vs the numbers of logic elements that have changed in the circuit after a netlist optimization. At first glance, there appears to be a polynomial relationship between the number of moves and the number of LEs changed in the circuit. Indeed, we are able to fit a curve to the experimental data of the form:

\[
\text{Moves} = k_1 \times (\Delta LE)^{1.7} + k_2
\]

This indicates to us that the typical runtime of the ICP algorithm is \(O(\Delta^{1.7})\) and the runtime generally increases in proportion to the amount of logic cells that have changed.

**B. Structured ASICs**

In the ASIC domain, we implemented our version of ICP and compared it against a ICP algorithm in the previous section. Once implemented, we ran over 60 industrial benchmark circuits through layout-driven synthesis where several designs contained more than 100K gates. The circuit set used in these experiments was different from the circuit set in section IV-A since much larger circuits were necessary to test the scalability of the structured ASIC algorithm. Table IV-B shows our summarized results for our tests.

Table IV-B shows the percent reduction in runtime when comparing the transportation-based ICP versus the localized two-stage ICP, with a geometric mean reduction of 52% or 2x speedup. Also shown is the percent reduction in memory usage, with a geometric mean reduction of 32%. The reason for a reduction in memory is because the transportation problem is only run on small subregions of the chip, thus a much smaller portion of data needs to be created at any given time. The localized ICP, however, must contain data for all locations that may contain a cell due to the basin filling and complex cost function described in section III-A. Note that the improvements shown in Tab. IV-B had almost no impact on the operating frequency of the circuits tested where we saw an average speed reduction of only 0.2%.

**V. Conclusion**

We have presented a general framework for solving the incremental placement problem for both FPGAs and structured ASICs. Our results confirm that our methods are practical means to legalize a given placement after the placement is altered by layout-driven optimizations. The main drawback to our approach is that it depends on a relatively small disruption to the netlist where our circuits are altered by only 4% on average. For larger changes, such as ECO alterations, alternative incremental placement algorithms need to be investigated.

One possible extension to this work is extending our analytic approach to incremental placement to the FPGA domain. FPGA architectures will soon contain hundreds of thousands of logic elements, thus a more scalable approach to incremental placement will be necessary. The primary problem would be to incorporate much more complex architectural constraints to current analytic methods, while maintaining circuit performance.

The concepts that we have introduced for incremental placement are required to ensure we have a legal placement while maintaining the performance gains incurred by our layout-driven optimizations. Furthermore, due to the generality of our approach, our algorithms can be easily adapted to a wide range of architectures.

**REFERENCES**


