

Incremental Placement for Structured ASICs using the Transportation Problem

Andrew C. Ling

Department Electrical and
Computer Engineering
University of Toronto
Toronto, Canada

e-mail: aling@eecg.toronto.edu

Deshanand P. Singh

Altera Corporation
Toronto Technology Centre
Toronto, Canada
e-mail: dsingh@altera.com

Stephen D. Brown

Altera Corporation
Toronto Technology Centre
Toronto, Canada
e-mail: sbrown@altera.com

Abstract—While physically driven synthesis techniques have proven to be an effective method to meet tight timing constraints required by a design, the incremental placement step during physically driven synthesis has emerged as the primary bottleneck. As a solution, this paper introduces a scalable incremental placement algorithm based upon the well known *transportation problem*. This method has an average speedup of 2x and a 30% reduction in memory usage when compared against a commercial incremental placer without any impact on area or speed of the final placed circuit. Furthermore, this method is scalable for structured ASICs.

I. INTRODUCTION

As the complexity of circuits increases exponentially, traditional synthesis methods often fall short of finding a timing-optimal solution. The primary reason for this is that delays of an unplaced circuit are hard to estimate, thus the synthesis tool lacks the knowledge required to optimize for timing. A solution to this is to resynthesize the circuit incrementally after placement. The benefit of doing this is that delays can be modeled accurately after placement since cell positions are known. This incremental process is known as physically-driven synthesis which we will refer to as physical synthesis [12]. The

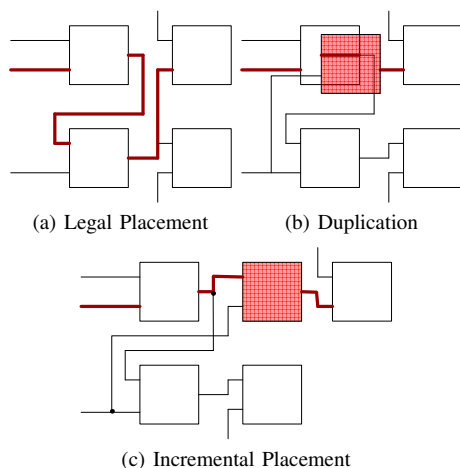


Fig. 1. Physical synthesis illegality example.

optimizations that occur during physical synthesis will usually lead to a placement that is illegal. For example, consider Fig. 1. Fig. 1a is an initial placement of three logic cells where the highlighted connections show the critical path. The length of a wire is often proportional to the delay of that wire, thus most timing optimizations attempt to shorten the critical path. One method to do this is through duplication [3]. Nodes feeding a critical connection are duplicated

and each duplicated node is moved to its critical fanout connection as shown in Fig. 1b. A problem with this optimization is that congestion often occurs after the duplication, which needs to be legalized. Physical synthesis deals with these illegalities with an incremental placement (ICP) step. During ICP, all illegalities are resolved while maintaining the quality of the optimizations done by physical synthesis. Quality is maintained by minimizing the disruption of the original placement. Going back to the previous duplication example, ICP is shown in Fig. 1c. As we will describe shortly, the main problem with current incremental placement algorithms is their localized approach to remove illegalities in the placement. This has proven to be very effective for FPGAs where the number of illegalities incurred after physical synthesis is small [11]. However, as physical synthesis algorithms are migrated to larger devices, such as structured ASICs, current ICP techniques will not scale. As a solution, we present a novel ICP algorithm for structured ASICs using the *transportation problem* [7]. The benefit of this method is that it solves illegalities on a global scale. Thus, more illegalities can be resolved at once in a very fast and efficient manner which outperforms a commercial incremental placer both in terms of runtime and memory use. The novelty of our ICP algorithm is identified by three major heuristics which have proven pivotal in creating a fast incremental placer without any degradation to circuit performance. This includes the guided movement of logic cells to remove illegalities, the use of the transportation problem to create guide paths for logic cell movement, and an ordering scheme that ensures congested regions get placed before uncongested regions.

II. BACKGROUND AND MOTIVATION

A. Structured ASICs

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. The goal of placement is to arrange cells such that no overlaps occur, while optimizing the circuit timing and area. From a conceptual point of view, valid placement regions form a grid where each point of the grid is given a planar coordinate (x,y) . A single cell can occupy multiple adjacent grid coordinates due to their varying widths. Thus, a legal cell placement will have at most one cell covering each grid coordinate. Fig. 2 shows an example legal placement.

B. Previous Work

In general, placement algorithms for ASICs are tackled from a global perspective. This is necessary to ensure the placer runs quickly. In [8], the authors present an incremental placer based on floorplan

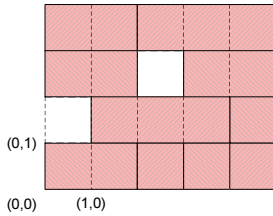


Fig. 2. Legal placement of cells on a grid.

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 RAM blocks; thus this ICP technique is not applicable to heterogeneously structured ASICs.

In [13], 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 [4], the authors present a placement tool called Domino. Like FastPlace, this work uses a quadratic placer [6] 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 [13], the detailed placement step disrupts all cells and thus cannot be applied to ICP. An improvement to this is presented in [5]. 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 [2], 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 while minimizing the disruption of the original placement.

1) *Localized ICP*: Finally, we review the incremental placement algorithm used in Altera's QuartusII. An early description of this tool was published in [11]. This algorithm has proven very successful for FPGAs and has recently been adapted to Altera's structured ASIC family. The primary goal of ICP is to resolve the architectural violations created when the physical synthesis modifications are integrated into the existing placement. In modern FPGAs [1] [14], architectural constraints are found in the clustered logic blocks (also known as LABs) including: a limit on the number of logic elements within the cluster; a limit on the number of distinct inputs to the cluster; and a limit on the number of distinct control signals (e.g. clock, reset) that can be used within the cluster. Structured ASICs have much simpler constraints where the primary constraint is overlaps. This is analogous to a limit on the number of logic elements in a cluster, where in structured ASICs only one cell can occupy a given location. Thus, the ICP algorithm described here primarily needs to remove overlaps between neighbouring cells and ignores other constraints specific to FPGAs.

The ICP algorithm uses an iterative improvement strategy where

cells are moved according to a cost function. This cost function consists of three components:

- *Legality Cost*. Each cell is penalized if it contains any architectural violations such as overlaps. The cost is proportional to the total number of constraints violated.
- *Timing Cost*. The timing cost is used to ensure that critical cells are not moved into locations that would significantly increase the critical path delay.
- *Wire-length Cost*. Wire-length estimation is used to ensure that the circuit is easily routable after the cell moves.

The total cost is a weighted sum of these components:

$$C = K_L \cdot \text{Legality} + K_T \cdot \text{Timing} + K_W \cdot \text{Wire-length} \quad (1)$$

The weighting coefficients (K_i) are used to normalize the contribution of each component so that the components contribute equally when considering a move.

There are several types of moves that can be generated for each candidate cell, x , to be moved:

- *Move to fanin*. Attempt to move x near a location that contains a fanin of x .
- *Move to fanout*. Attempt to move x near a location that contains a fanout of x .
- *Move to sibling*. A sibling of x is a cell driven by a fanin of x . We attempt to move x near a location containing one of its siblings.
- *Move to neighbor*. Attempt to move to an adjacent location.
- *Move to space*. Attempt to move to a randomly selected empty cell location in the device.
- *Move in direction of critical vector*. Compute a *critical vector* for x and move to a location in this direction.

The moves to fanins, fanouts, and siblings are essential to ensure that wire-length is not degraded. The critical vector for a cell x is shown in Fig. 3. The direction of the critical vector is computed by summing the directions of all the critical connections attached to x . A move in the direction of the critical vector helps correct any mistakes when unexpected paths become critical as a result of moves performed in preceding iterations. Note that the critical vector move is similar to the move types attempted by iterative force-directed placement algorithms.

Although the selection of move type is random, the proposed moves are biased in the direction of free space. For example, if the target device had a large amount of free space close to the top edge of the chip, then moves that move cell closer to the top edge are selected more frequently.

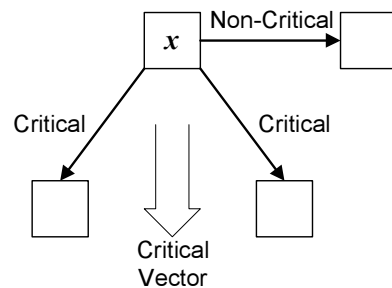


Fig. 3. A critical vector for a cell x .

The ICP algorithm makes cost-lowering moves until no further illegalities exist in the placement. This greedy strategy can easily become stuck in a local minima where none of the proposed moves seem to

improve the cost. For example, consider the situation illustrated in Fig. 4. Assume that the current solution is x and that it still contains illegalities that need to be resolved. All moves in the neighborhood of x increase the cost of the solution and ICP is unlikely to find the global minima, y , through a process of randomized local moves. However, through *basin filling* we can change the shape of the cost function so that ICP is forced to move out of the local minima. Basin filling is achieved by slowly increasing the weight for those illegalities that are consistently present in the local solution space, thereby forcing ICP to select moves that resolve illegality at the cost of increases in timing and/or wire-length. Our basin filling approach is similar to the notion of negotiated congestion in the PathFinder [9] algorithm used for FPGA routing.

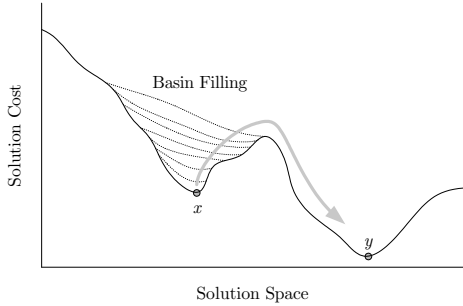


Fig. 4. Using basin filling to escape local minima.

Localized ICP has proven very successful to remove illegalities without harming the placement quality. However, the algorithm described previously can consume more than half the runtime of physical synthesis due to its localized nature. Furthermore, a large amount of data is required during localized ICP to maintain a history of the basin filling and cost function. Thus, this method is not scalable to large chips that contain millions of cells. In our approach, we resolve this problem in ICP by adapting techniques from both the localized ICP method and analytical placers described previously.

C. Transportation Problem

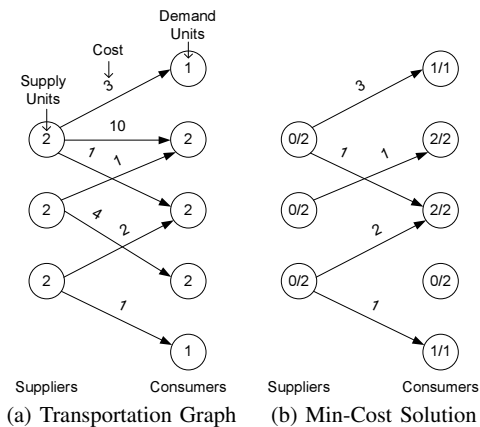


Fig. 5. Illustration of the transportation problem.

The incremental placer we have developed is based on a guided shifting technique where guide paths are calculated using the transportation problem: given a set of suppliers and consumers at fixed locations where there exists a set of directed paths from suppliers to consumers with an associated cost, find a subset of paths that fulfills the demands of all the consumers or exhausts the supply of all the

suppliers such that the total cost of the used paths is minimized. The directed paths are also known as edges. Edges leave the suppliers and enter the consumers, thus the suppliers are the edge *tails* and the consumers are the edge *heads*. Fig. 5 illustrates the transportation problem. Fig. 5a illustrates a possible set of consumers and suppliers connected by a set of paths, and Fig. 5b illustrates a minimum cost path solution such that the supply of all suppliers is exhausted. There has been much work that has been done to solve the transportation problem quickly [7], [10] and is not discussed here.

III. INCREMENTAL PLACEMENT

A. High-Level Algorithm

```

1  EMPTYQUEUE()
2  foreach  $l \leftarrow IllegalLocation$ 
3      PUSHQUEUE( $l$ )
4  end foreach
5   $s \leftarrow SIZEOFQUEUE()$ 
6   $c \leftarrow 0$ 
7  while NOTEMPTYQUEUE() AND ( $c < CUTOFFPOINT(S)$ )
8       $l \leftarrow POPQUEUE()$ 
9       $c \leftarrow c + 1$ 
10      $R \leftarrow CREATESUBREGION(l)$ 
11     REMOVECELLS( $R$ )
12      $L \leftarrow PLACECELLS(R)$ 
13     PUSHQUEUE( $L$ )
14 end while
15 if NOTEMPTYQUEUE()
16     CALLLOCALICP()
17 end if

```

Fig. 6. High-level overview of transportation-based incremental placer.

A high-level overview of our incremental placer is shown in Fig. 6. 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 7). 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 localized ICP algorithm, which in contrast moves cells one by one. It is possible for new illegalities to be created during the re-placement process (line 12, set L). If so, these are added to the back of the illegality queue (line 13). This ensures that the original illegalities are processed before newly formed illegalities.

The original size of the illegality queue is used to form a cutoff point (line 7). If the number of illegal locations processed reaches this cutoff point before the placement is legalized, the incremental placer falls back on a localized algorithm similar to that described in Section II-B.1. This finishes up any fine illegalities that could not be resolved with the transportation-based incremental placer (line 15-17).

B. 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.

C. 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. 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. 7. The heuristic used for determining the costs of paths between a cell and location will be described in the next section. Another consideration

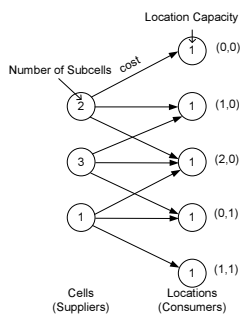


Fig. 7. Representing incremental placement as a transportation problem.

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 [4]. 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, 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. 8a. Fig. 8b 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 graph, 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.

1) *Transportation Problem Path Costs*: Since the transportation problem is a cost minimization problem, the edge costs in the transportation graph must ensure that the transportation solver will find cell locations that reduce the overall delay between cells. One estimate of delay is wire-length where delay is proportional to wire-length. Thus, using a path cost that represents wire-length will ensure that the overall wire-length, and hence delay, will be minimized. The metric we use to estimate the wire-length of a net is the half-length perimeter [4]. The half-length perimeter is the width plus height of the bounding box containing all the pins of a net as illustrated in Fig. 9. We assume that the cell pins are at the center of the cell.

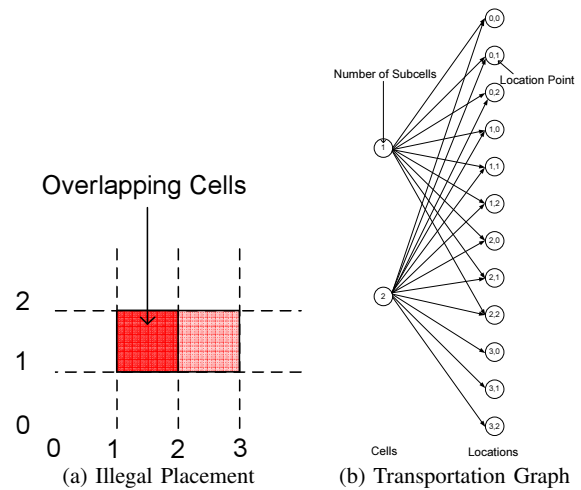


Fig. 8. Incremental placement example.

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.

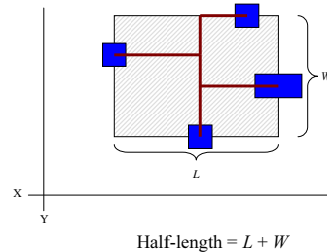
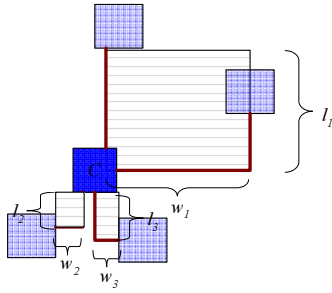


Fig. 9. Half-length perimeter of a net.

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. 10. 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. 10, 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 to physical synthesis optimizations since they use delay values of the current placement to guide their optimizations, even though the final placement after physical synthesis will be different due to ICP. Thus, even though some cells will move during ICP, the initial delay values used in physical synthesis optimizations are sufficient approximations.

D. 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. 11a. Fig. 11b illustrates the shifting process if the original cell location is occupied.



$$C = (l_1 + w_1) + (l_2 + w_2) + (l_3 + w_3)$$

Fig. 10. Path cost calculation.

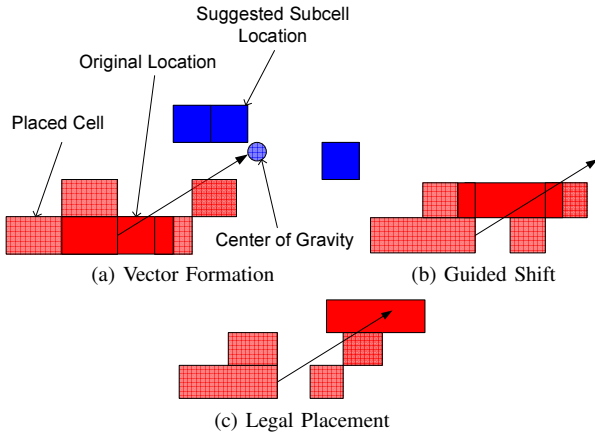


Fig. 11. Example of replacement using suggested location from transportation problem.

This will continue until a final valid location is found as shown in Fig. 11c. 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.

E. Order of Re-Placement

After the subregion is emptied and all cells have been assigned a guided vector, re-placement of cells can occur. Cells are placed one at a time. The order of re-placement is extremely important in determining the quality of the final solution. Crowded regions should be placed before uncrowded regions since crowded regions have much less freedom for cell placement. Furthermore, cells on critical paths should be placed before other cells to ensure they will not be shifted far from their original location. Placing cells in crowded regions first is achieved by starting re-placement at illegal locations and working outwards. If there are more than one illegal location in the area, the illegal locations that contain cells on critical paths are placed first. This process is illustrated in Fig. 12. Fig. 12a illustrates the initial subregion and identifies the locations where illegalities occur. Note that one of the illegal locations is given higher priority since it contains critical cells. Next, placement regions are created (Fig. 12b). Placement will occur starting at each illegal location, then continuing in each vertical direction of the placement region (Fig. 12c). Once placement in the vertical regions completes (Fig. 12d), the placement regions are expanded in the horizontal directions (Fig. 12e). Placement continues in the vertical direction of

the newly expanded placement regions (Fig. 12f,g) and the process continues until the entire region is placed (Fig. 12h). Once the entire subregion has been placed, the next illegal location in the illegality queue can be processed.

IV. RESULTS

We implemented our version of ICP within a commercial placer called QuartusII and compared it against the original ICP algorithm described in section II-B.1. Once implemented, we ran over 60 industrial benchmark circuits through physical synthesis where several designs contained more than 100K gates (designs are proprietary and cannot be disclosed). Table. IV shows our summarized results for our tests.

Percent Reduction		
Runtime	Memory	Clock Frequency
52%	35%	0.2%

TABLE I

GEOMETRIC MEAN REDUCTION IN RUNTIME AND MEMORY USE OF OUR ALGORITHM WHEN COMPARED AGAINST QUARTUSII (RAN THROUGH 60 INDUSTRIAL DESIGNS).

Table IV shows the percent reduction in runtime when comparing the transportation-based ICP versus the localized ICP, with a geometric mean reduction of 52% or 2x speedup. Since it is known that ICP dominates the runtime during physical synthesis, this is a significant reduction in the overall physical synthesis runtime. 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 II-B.1. Note that the improvements shown in Tab. IV had almost no impact on the maximum clock frequency of the circuits tested where we saw an average reduction of only 0.2%.

One concern that could be raised is related to the convergence of our algorithm. Since our algorithm may create new illegalities during our re-placement process, convergence will be a problem if a large number of illegalities exist at the start of our algorithm. Also, if all the chip illegalities are clustered in a small region, we will need several iterations to shift cells in surrounding regions to make space to remove the overlaps in the densely packed area. However, from our experience, we have noticed that less than 4% of chip locations will contain illegalities after physical synthesis. Furthermore, the placement overlaps tend to be distributed throughout the chip. Thus, under these conditions convergence is not a problem, and our global ICP algorithm should be able to remove most overlaps found after physical synthesis optimizations. Our 2x speedup supports this claim and implies that our algorithm converges quite quickly.

V. CONCLUSIONS

We have developed a novel incremental placer based upon guided shifts. This method works well for large designs where there only exists one constraint for placement. This method has proven to be much faster than the existing ICP found in an existing commercial placement tool. More importantly, it is scalable to structured ASICs both in terms of runtime and memory usage.

REFERENCES

- [1] Altera. Component selector guide ver 14.0, 2004.
- [2] U. Brenner and M. Struzyna. Faster and better global placement by a new transportation algorithm. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 591–596, 2005.
- [3] G. Chen and J. Cong. Simultaneous timing-driven placement and duplication. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 51–59, New York, NY, USA, 2005. ACM Press.
- [4] K. Doll, F. Johannes, , and G. Sigl. Domino: Deterministic placement improvement with hill-climbing capabilities. In *VLSI*, pages 3b.1.1–3b.1.10, 1991.
- [5] S. Dutt, H. Ren, F. Yuan, and V. Suthar. A network-flow approach to timing-driven incremental placement for asics. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 375–382, 2006.
- [6] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 356–365, 1991.
- [7] D. Klingman and R. Russel. Solving constrained transportation problems. In *Operations Research*, pages 91–106, 1975.
- [8] C. Li, C.-K. Koh, and P. H. Madden. Floorplan management: Incremental placement for gate sizing and buffer insertion. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 349–354, 2005.
- [9] L. McMurchie and C. Ebeling. Pathfinder: a negotiation-based performance-driven router for FPGAs. In *FPGA '95: Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, pages 111–117, New York, NY, USA, 1995. ACM Press.
- [10] K. G. Ramakrishnan. Solving two-commodity transportation problems with coupling constraints. *J. ACM*, 27(4):736–757, 1980.
- [11] D. P. Singh and S. D. Brown. Incremental placement for layout driven optimizations on FPGAs. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 752–759, New York, NY, USA, 2002. ACM Press.
- [12] D. P. Singh, V. Manoharajah, and S. D. Brown. Two-stage physical synthesis for FPGAs. In *CICC '05: Proceedings of the 2005 IEEE Custom Integrated Circuit Conference*, 2005.
- [13] N. Viswanathan and C. C.-N. Chu. Fastplace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *ISPD '04: Proceedings of the 2004 international symposium on Physical design*, pages 26–33, New York, NY, USA, 2004. ACM Press.
- [14] Xilinx. Virtex-ii complete data sheet ver 3.3, 2004.

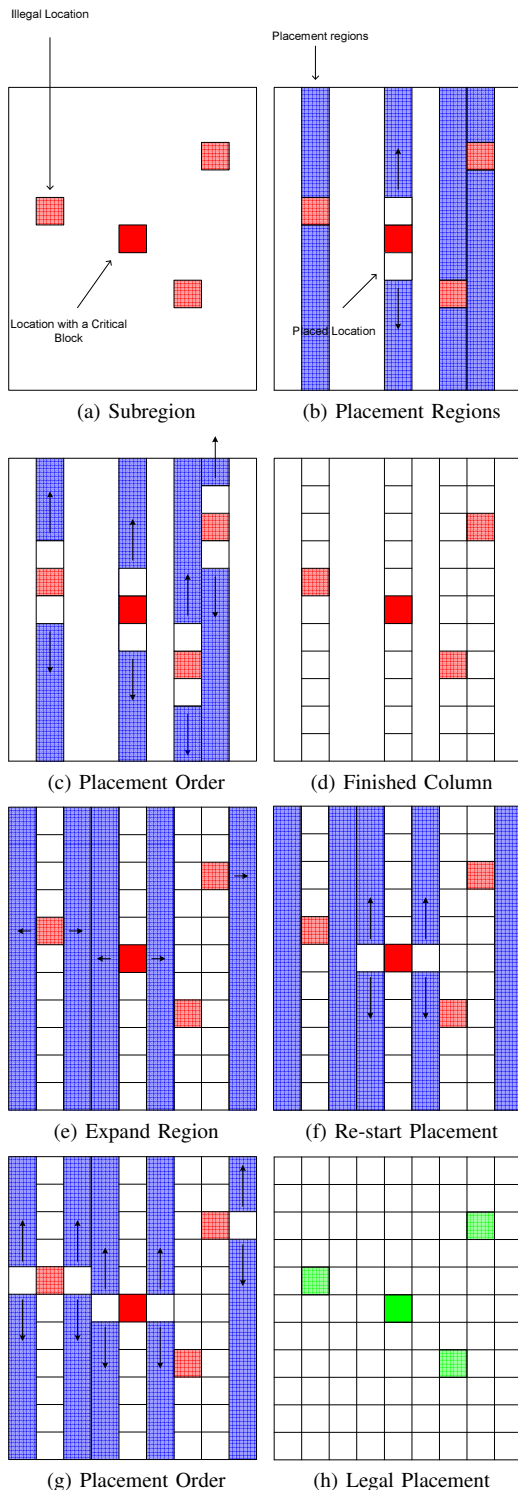


Fig. 12. Order of re-placement in illegal subregion.