Computational Geometry Based Placement Migration

Tao Luo[†], Haoxing Ren[†‡], Charles J. Alpert[‡], and David Z. Pan[†]
[†] Department of ECE, University of Texas at Austin, Austin TX 78712
[‡] IBM Corporation, 11400 Burnet Road, Austin TX 78758
{*tluo, dpan*}*@ece.utexas.edu,* {*haoxing, alpert*}*@us.ibm.com*

ABSTRACT

Placement migration is a critical step to address a variety of post-placement design closure issues, such as timing, routing congestion, signal integrity, and heat distribution. To fix a design problem, one would like to perturb the design as little as possible while preserving the integrity of the original placement. This work presents a novel computational geometry based placement migration method, and a new stability metric to more accurately measure the "similarity" between two placements. It has two stages, a bin-based spreading at coarse scale and a Delaunay triangulation based spreading at finer grain. It has clear advantage over conventional legalization algorithms such that the neighborhood characteristics of the original placement are preserved. Thus, the placement migration is much more stable, which is important to maintain. Applying this technique to placement legalization demonstrates significant improvements in wire length and stability compared to other popular legalization algorithms.

1. INTRODUCTION

In modern placement and physical synthesis of VLSI circuits, one is increasingly faced with the placement migration problem, which is to take an existing placement, fix some design violations and re-legalize it. For example, during physical synthesis or Engineering Change Order (ECO) optimization, many buffers may be inserted and gates resized, creating a lot of overlapping cells. These cells need to be legalized, but one should avoid disturbing the previous placement too much to achieve design convergence. Also another example, post routing congestion analysis may identify severe hot spots (e.g., congestion, noise, power, thermal), and placement migration is needed to smoothly spread out cells in these hot spots [1]. Due to the complexity of modern nanometer designs, it is unlikely to design one placement algorithm that meets the multi-objective design closure target in a single run. More often, a placement flow involves multiple placement-improvement iterations. So a stable placement migration algorithm is crucial for the multi-objective design closure.

These tasks share a common theme of starting with an initial placement that is "good" and perturbing it so that it is improved in some way while still preserving the essential characteristics (cell ordering, wirelength, etc.) of the original placement. Ideally, the later placement iteration should be able to preserve previous fixes and accumulate additional improvements to achieve the design closure. Therefore, the stability of the placement algorithm is very important. Obviously, we do not want each placement iteration generates entirely different result and destroys all previous optimization efforts.

Among various placement migration applications, legalization is probably the most common one. Therefore, the remainder of the paper will discuss our placement migration algorithm in this context. Existing legalization techniques for legalization include network flow [2] [3], dynamic programming [4][5], heuristic ripple cell movement [6], and single row optimization [7] [8]. The network flow approach [3] uses minimum cost flows to minimize the weighted sum of (squared) cell movements. The dynamic programming based approach [4] solves the optimal assignment of cells to placement sites under the constraint of cell ordering. Mongrel [6] uses a greedy heuristic to move cells from overflowed bins to under capacity bins in a ripple fashion based on total wire length (TWL) gain. The single row optimization techniques [7] [8] use dynamic programming to optimally place cells in a single circuit row.

While there are many existing legalization algorithms, there are very few works directly targeting incremental and stable placement migration.[1] In this paper, we develop a novel technique for stable placement migration based on the *computational geometry*. We also propose a new placement stability *metric* which can be used to measure the placement migration stability. Our algorithm has two key steps: bin-based cell spreading and Delaunay triangulation based overlap reduction. The algorithm takes advantage of the computational geometry property of the existing placement. Thus it captures the relative cell order nicely during placement migration. Our experimental results compared to other widely used legalization algorithms clearly demonstrate the superiority of our algorithm, with over 10% wire length reduction and significantly better stability score.

The rest of the paper is organized as follows. Section 2 presents the bin based spreading algorithm. Section 3 presents the Delaunay based overlap reduction procedure. The complete computational geometry based legalization algorithm is given in section 4. Section 5 proposes a new placement stability metric suitable for placement migration. Very promising experimental results are obtained in section 6, followed by conclusion in section 7.

2. BIN BASED SPREADING

A placement is close to legal if all that is required to legalize the placement is to snap cells to rows or perhaps perform minor cell sliding in order to fit the cells. Assuming the chip layout is divided into equal sized bins, the placement is considered close to

This work is supported in part by SRC under contract 2005-TJ-1321, IBM Faculty Award, Sun, and equipment donations from Intel.

[1]The most recent work on diffusion-based placement [7]simulates the placement spreading using the physical diffusion equation. It shares some common theme with this work, but using very different approaches. See Section 6 for more discussion on these two approaches.

legal if the area density of every bin is less than or equal to D_{max} (e.g., $D_{max} = 1$). For all bins with density greater than D_{max}, cells must be migrated to other bins. Therefore the goal of our migration algorithm is to reduce the density of each bin to no more than D_{max} while avoiding moving these cells far from their original locations thus preserving the original placement characteristics.

Bin based spreading is a geometric approach to evenly reduce cell density on the congested regions. Suppose we divide the entire placement region into K*L square bins, there will be $(K+1)$*$(L+1)$ bin corners. The idea is to move those bin corners such that the resulting bin capacity would satisfy the density constraints, and then move cells accordingly. By stretching the bin corners, we preserve the relative order of neighboring bins; meanwhile by interpolating cells relative to its bin corners, we preserve the relative order of cells inside the bin. We perform the bin stretching and cell interpolation iteratively until all the bins are under the maximum density D_{max}.

2.1 Bin Stretching

At each iteration, we first compute the bin density $D_{k,l}(n)$ (the nth iteration), then compute the amount of stretching needed for each bin. For those overpopulated bins, the idea is to expand that bin such that the density of the new bin is equal to D_{max}. At the same time, to accelerate the spreading process, we allow the adjacent bins to shrink such that their densities equal to D_{max} as well. The amount of stretching for bin (k,l) on both horizontal and vertical directions can be written as:

$$\begin{aligned} \varepsilon^x_{k,l} &= (\sqrt{\frac{D_{k,l}(n)}{D_{max}}} - 1)W \\ \varepsilon^y_{k,l} &= (\sqrt{\frac{D_{k,l}(n)}{D_{max}}} - 1)H \end{aligned} \quad (1)$$

where W and H are the bin width and height, respectively.

Stretching each bin itself would generate overlaps between adjacent bins. Therefore we stretch the bin corners of adjacent bins instead of bins itself. Let $(p^x_{k,l}(n), p^y_{k,l}(n))$ denotes the coordinates of an inner bin corner, which is shared by four neighboring bins, denoted as $(k-1,l-1)$, $(k-1,l)$,$(k,l-1)$, and (k,l). We can use (1) to compute the amount of horizontal and vertical stretching needed for each one of the four bins, which will give us four stretched corner positions, and then compute the combined number of these four as the corner position for next iteration, $(p^x_{k,l}(n+1), p^y_{k,l}(n+1))$,

$$\begin{aligned} p^x_{k,l}(n+1) &= p^x_{k,l}(n) + 0.5(\varepsilon^x_{k-1,l-1} + \varepsilon^x_{k-1,l} - \varepsilon^x_{k,l-1} - \varepsilon^x_{k,l}) \\ p^y_{k,l}(n+1) &= p^y_{k,l}(n) + 0.5(\varepsilon^y_{k-1,l-1} + \varepsilon^y_{k-1,l} - \varepsilon^y_{k-1,l} - \varepsilon^y_{k,l}) \end{aligned} \quad (2)$$

Because the stretching is uniform on both bin corners on the same bin edge, we only take half the stretching value given by (1). If any neighboring bin is on the chip boundary, we take the 0.5 factor off.

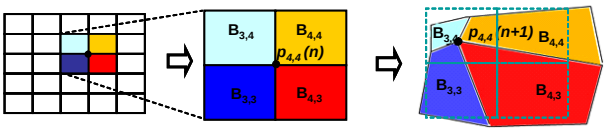

Figure 1: Illustration of bin and corner stretching

Figure 1 is an illustration of the movement of the corner point $p_{4,4}$ under accumulated stretching from all four surrounding bins. Bin $(3,3)$, $(4,3)$, and $(4,4)$ are over the maximum density, therefore we expand them, while bin $(3,4)$ is under the maximum density, thus we compact it. We will have four new corner positions of this corner for each bin. Such process is iterated as needed. After computing coordinates of all points, cells inside the bin will move within the distorted bin as explained in next section.

2.2 Cell Interpolation

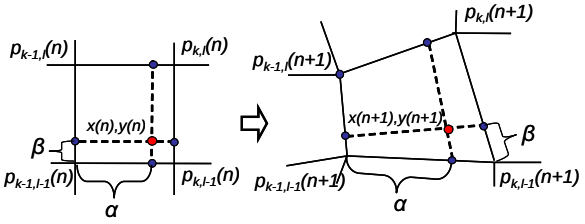

Figure 2: Cell location interpolation on stretched bin

The computation of new cell coordinates is a linear interpolation process, which maps all cells from the original bin into the new bin at the same relative positions. As shown in Figure 2, the four corner coordinates of the bin are $\mathbf{p}_{k-1,l-1}(n)$, $\mathbf{p}_{k,l-1}(n)$, $\mathbf{p}_{k-1,l}(n)$, and $\mathbf{p}_{k,l}(n)$. Their coordinates after bin stretching are: $\mathbf{p}_{k-1,l-1}(n+1)$, $\mathbf{p}_{k,l-1}(n+1)$, $\mathbf{p}_{k-1,l}(n+1)$, and $\mathbf{p}_{k,l}(n+1)$. For a cell $(x(n),y(n))$ within the bin, the new coordinates $x(n+1)$ and $y(n+1)$ can be computed by the following equations.

$$\begin{aligned} x(n+1) &= \gamma_x + \beta(\xi_x - \gamma_x) \\ y(n+1) &= \gamma_y + \alpha(\xi_y - \gamma_y) \end{aligned} \quad (3)$$

where

$$\begin{aligned} \alpha &= \frac{x(n) - p^x_{k-1,l-1}(n)}{p^x_{k,l-1}(n) - p^x_{k-1,l-1}(n)} \\ \beta &= \frac{y(n) - p^y_{k-1,l-1}(n)}{p^y_{k-1,l}(n) - p^y_{k-1,l-1}(n)} \\ \gamma_x &= p^x_{k-1,l-1}(n+1) + \alpha(p^x_{k,l-1}(n+1) - p^x_{k-1,l-1}(n+1)) \\ \xi_x &= p^x_{k-1,l}(n+1) + \alpha(p^x_{k,l}(n+1) - p^x_{k-1,l}(n+1)) \\ \gamma_y &= p^y_{k-1,l-1}(n+1) + \beta(p^y_{k-1,l}(n+1) - p^y_{k-1,l-1}(n+1)) \\ \xi_y &= p^y_{k,l-1}(n+1) + \beta(p^y_{k,l}(n+1) - p^y_{k,l-1}(n+1)) \end{aligned} \quad (4)$$

2.3 Bin Based Spreading Algorithm

At each iteration of the bin based spreading algorithm, it first stretches the bin corner to make congested bin larger, then interpolates cell locations accordingly. It then restores all the bin boundary and starts an new iteration. The new iteration recomputes the bin density and repeats all above procedures. The process stops once that all the bin densities are lower than the maximum density D_{max}.

To avoid over expansion in non-congested region, we only change the bin corners of those bins above D_{max} during bin stretching. It assures that cells are pushed from high density area to low density area steadily and smoothly. It also reduces unnecessary oscillation and computations.

The stability of the migration process is affected by the bin size (area) as well. The ideal initial bin size is depending on the size of the circuits. If the bin size is too large, the internal density distribution inside the bin might still violate the density constraints even if the bin as a whole is under D_{max}. However, if the bin size is too small, oscillation will appear and bin boundary distortion may impact the smoothness of spreading. We may see cells tend

to cluster in some areas. This problem is solved by a hierarchical addition to our original formulation. The idea is straightforward. It uses big bin sizes from at the beginning, then recursively cuts big bins into smaller bins, and adjusts the internal density distribution. The hierarchical technique is necessary to handle fixed macros. At the time the bin size is smaller enough, bin edges be close to macro boundaries. Cells will move along the boundary, they will not move toward the macros. The complete bin based spreading algorithm is given by Algorithm 1.

Algorithm 1 Computational Geometry Bin Based Spreading

Procedure: BIN
Input: cell placement x_i, y_i, bin area $A_B = W \cdot H$, maximum bin density D_{max}
Output: new placement $\hat{x}_i, \hat{y}_i$
begin
  Initialize bin density $D_{k,l}$;
  if A_B is too small **then return;**
  while any $D_{k,l} > D_{max}$
    for each bin with $D_{k,l} > D_{max}$
      Compute bin expansion $\varepsilon^x_{k,l}$, $\varepsilon^y_{k,l}$ with (1);
    end for
    Compute bin corner $\mathbf{p}_{k,l}(n+1)$ with (2);
    Interpolate cell locations $x_i(n+1), y_i(n+1)$ with (3);
    Restore all bin corners, update $D_{k,l}$;
    $n = n+1$;
  end while
  Update $\hat{x}_i = x_i(n), \hat{y}_i = y_i(n)$;
  Reduce bin area $A_B = A_B/2$;
  Recursively call BIN($\hat{x}_i$, $\hat{y}_i$, A_B, D_{max});
end

Note that our approach is different from the grid warping [9] and cell shifting [10]. At each partition step, grid warping slices the region into 2 x 2 or 4 x 4 equal "volume" quadrilateral grids, transforming the grid (and cells) back to equal shape rectangles to form the subproblems. The elastic grids in grid warping are the equivalence to Gordian's min-cut partition [11], both purpose is for partition, while our bins are used for spreading directly. We reshape each bin individually at each step and rely on iterations to flow cells out eventually. The cell shifting [10] technique is an one dimensional greedy shifting, which is used to generate the spreading forces for the global placer. It is the quadratic solver that does the actual spreading; while our approach is a two dimensional approach, and it spreads out cells directly.

3. DELAUNAY TRIANGLE BASED OVERLAPPING REMOVING

Bin based spreading is good for coarse level spreading. However, to further remove overlapping between cells, we need to use more fine-grained migration techniques. In this section, we will present the Delaunay triangulation based algorithm to effectively remove cell overlap while preserving placement stability.

3.1 Delaunay Triangulation

The Delaunay triangulation is the dual of the Voronoi diagram – one of the most fundamental data structures in computational geometry [12]. The Voronoi diagram for a collection of geometric objects is a partition of space such that each of them consists of the points closer to one particular object than to any others. It contains a straight-line edge connecting two sites in the plane if and only if their Voronoi regions share a common edge. The Delaunay triangle edges of an object essentially captures its relative proximity relationship with other objects.

Figure 3 shows an example of the Voronoi diagram and its corresponding Delaunay triangulation. For a given VLSI placement to be migrated smoothly to another solution due to legalization need, congestion or noise mitigation, we can compute the Delaunay triangulation for all cells efficiently. Based on this Delaunay triangulation that captures the "preferred" proximity relationships among all fixed and placeable objects, we can perform stable placement migration, to spread cells smoothly from congested area, as illustrated in Figure 3.

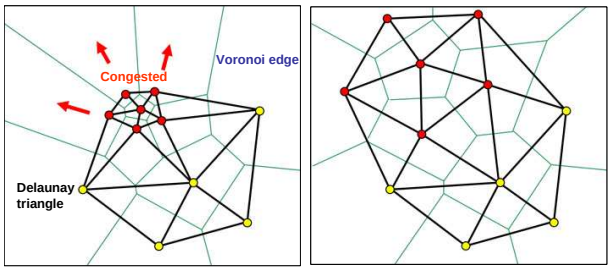

Figure 3: Delaunay triangulation captures the relative order, which can be used to spread cells during placement.

Delaunay triangulation is an important topic in computational geometry and has wide applications in varies field, such as visualization, finite element analysis, and discrete wireless networks. There are quite a few mature Delaunay triangulation algorithms developed, with the computational complexity ranges from O($n \log n$) to O(n^2). The reader is referred to [12] for a comprehensive survey of Delaunay triangulation and Voronoi diagram.

Given a placement, we can construct the Delaunay triangulation of all the cells using its center locations as triangle nodes. Then the placement plane becomes a planar graph $G = (V, E)$, with $V = \{v_1, v_2, ..., v_n\}$ corresponding cells and $E = \{e_1, e_2, ...e_m\}$ triangle edges. The boundary of the graph are fixed pads. We only move non-boundary or non-fixed cells. Figureo 4 shows a Delaunay triangulated placement region.

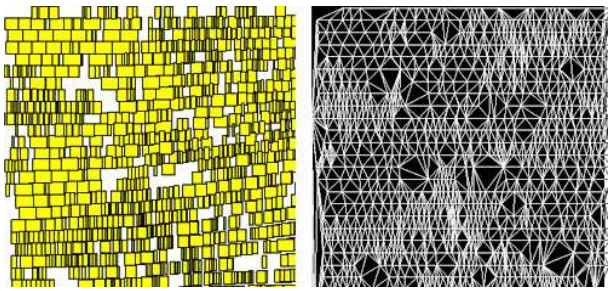

Figure 4: Delaunay triangulation of a placement region

3.2 Fine-grain Overlapping Reduction

Because Delaunay triangulation helps to identify all close neighbors of one cell, such detailed information is valuable for fine-grain adjustments. We use the Delaunay triangulation to do further cell spreading, where the bin based spreading is not applicable. The Delaunay triangulation based cell overlapping reduction works as follows.

To iterate through cells in the placement order, we build a tree structure on the delaunay triangulated placement. One cell in the center of the placement is selected as the tree root, and all cells connecting to the root by Delaunay edges are added into the tree as the second level tree nodes. Then all cells connecting to second level nodes are added as the third level tree nodes. Note that one cell may connect to two second level tree nodes by Delaunay edges. The cell is added to one tree node as the child only. The criteria of where to add the cell is to keep the number of child of each tree node balanced. Similarly, the tree keeps growing until all cells in the placement are added. Figure 5 illustrats the steps to build the tree on a delaunay triangulated region. Cells with the same color are tree node the same level.

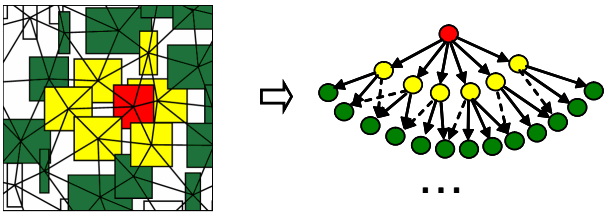

Figure 5: Tree structure for Delaunay edge traversing

Starts from the root, the algorithm traverses the tree in breadth-first manner. For every tree node - cell i, all Delaunay edges connecting cell i with the same or next level nodes are inspected. Let $e_{i,j}$ be the Delaunay edge between cell i and cell j. From the Delaunay triangle properties, we know that i and j are the nearest neighbor to each other. If cell i does not overlap with cell j, we do nothing and move on to the next Delaunay triangle edge. If cell i overlaps with cell j, the overlap distances on x and y directions are measured and cells will be pushed away accordingly. Let $\triangle^x_{i,j}$ and $\triangle^y_{i,j}$ be the x and y direction overlapping between i and j, respectively. If $\triangle^x_{i,j} > \triangle^y_{i,j}$, a repelling force is generated between cell i and cell j on x direction. We try to make minimum movement to remove the overlapping. So the force is inversely proportional to the cell sizes with weight to push the cell away from congestion. Let $f^x_{i->j}$ denote the repelling force from cell i to cell j.

$$f^x_{i->j} = \triangle^x_{i,j} \frac{w_j}{w_i + w_j} \tag{5}$$

where w_i and w_j are the widths of cells i and j. If $\triangle^x_{i,j} < \triangle^y_{i,j}$, the fource will be in the y-direction, i.e.,

$$f^y_{i->j} = \triangle^x_{i,j} \frac{h_j}{h_i + h_j} \tag{6}$$

where h_i and h_j are the heights of cells i and j.

If a movable cell i is connected with multiple neighbors by Delaunay edges, the total force F^x_i on $cell_i$ is the superposition of all overlapped neighboring cells

$$F^x_i = \sum_{j \in Neighbor(i)} f^x_{j->i} \tag{7}$$

where $Neighbor(i)$ denotes the set of cells overlapped with cell i.

Figure 6 is an example to illustrate how forces are added to the overlapping cells. As shown in Figure 6, assume cell A, B, C are within one Delaunay triangle. We can see that B and C are overlapped, and the overlapping in x direction is smaller, i.e. $\triangle^x_{B,C} < \triangle^y_{B,C}$. Then the y-directional force $f^y_{B->C}$ and $f^y_{C->B}$ will be applied on cells C and B, respectively. In the case that a cell overlaps with many surrounding neighbors, the total force tends to cancel each other. This usually happens at the center of congested area, and we can set certain density threshold to avoid redundant computation. The cells close to whitespace will move first and pull cells inside congested areas out smoothly.

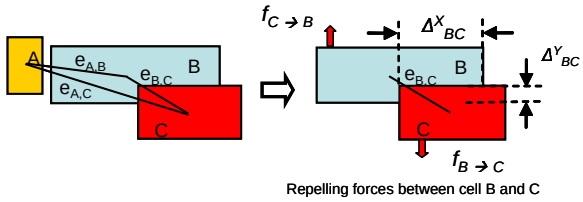

Figure 6: Delaunay force to reduce overlapping

The Delaunay triangulation based overlapping reduction process is outlined in Algorithm 2.

Algorithm 2 Delaunay Based Overlapping Reduction

```
1: Procedure: DELT
2: Input: cell placement x_i, y_i
3: Output: new placement x̂_i, ŷ_i
4: begin
5:   while stopping criteria is not satisfied
6:     if redo Delaunay condition is satisfied then
7:       T = {V, E} ← (x_i, y_i)
8:     end if
9:     BFS (T)
10:      for each edges e_{i,j} connect with cell i
11:        check connected cells i and j
12:        if i does not overlap with j then continue;
13:        if △^x_{i,j} < △^y_{i,j}
14:          compute f^x_{i→j}
15:        else
16:          compute f^y_{i→j}
17:        end if
18:      for each cell i in T
19:       move all cells on force
20:    end BFS
21:    sum up forces and update coordinates of cell
22:    end for
23:  end while
24: end
```

4. COMPUTATIONAL GEOMETRY BASED LEGALIZATION

Our algorithm consists of two major steps: bin based spreading and Delauney Triangle based overlap reduction. As described earlier, bin based spreading reduces the bin density overflow quickly at coarse level, and the Delaunay Triangle based overlap reduction step works at fine-gained level to reduce the overlap between adjacent cells. After bin based spreading and Delaunay Triangle based overlap reduction, the placement should have a max density of D_{max} and is roughly legal. We will run a final legalization step to put cells onto circuit rows without overlap,which takes very small effort since the density constraint is satisfied at fine granu-

larity level. The emphasis is to study the impact of our computational geometry placement migration algorithms comparing with other methods, such as the greedy and the flow, thus we just use a standard legalizer to generate the final legal placement. In fact, it is almost trivial after our migration. The complete computational geometry based legalization (*CGL*) algorithm is given in Algorithm 3). Note that the combined bin based spreading and Delauney Triangle algorithm gives the best result. For comparison purpose, we test the bin based spreading algorithm alone for legalization. It is referred to as CGL_B.

Algorithm 3 Computational Geometry Legalization Algorithm

```
1: Procedure: CGL
2: Input: A cell placement x_i, y_i
3: Output: A new placement x̂_i, ŷ_i
4: Parameters: Initial bin area: A_B, max bin density D_max
5: begin
6:     Call bin based spreading algorithm (Algorithm 1):
7:       (x̂_i, ŷ_i)= BIN( x_i, y_i, A_B, D_max);
8:     Call Delauney Triangle based algorithm (Algorithm 2):
9:       (x̂_i, ŷ_i)=DELT(x̂_i, ŷ_i);
10:    Put cell onto circuit row and remove remaining overlaps;
11:    return x̂_i, ŷ_i
12: end
```

5. GEOMETRIC PLACEMENT STABILITY METRICS

During placement migration one often needs to compare the difference of the original (golden) placement with a new placement generated by placement migration. It can be measured by the placement stability metrics. In the existing literature [13], two placement stability metrics are used: one measures the average cell movement between two placements, and the other measures the change of net clusters. During placement migration, however, it is possible that a large number of cells are shifted, all with a small amount. Thus all nets between cells have very small changes (like in our Delaunay triangulation spreading). For such scenario, the *absolute* cell movement metric is not a good metric [13]. The net cluster metric [13] is good to capture global placement stability where one big cluster can be moved to another part of the chip. But it is not very suitable for placement migration applications where most changes are small. During placement migration, it is desired to keep the relative geometric order and punish the most disruptive changes. Therefore we propose the following geometric stability metric.

Suppose we are given two placements: a golden placement A and a new placement B generated by placement migration. Our idea is to measure the change of cell placement *relative* to its neighboring cells and sum up the most significant changes to capture the difference of these two placements. Both placements have the same number of cells. The coordinates of cell i are (x_i, y_i) and $(\hat{x}_i, \hat{y}_i)$ for placement A and B, respectively. We select a group of cells adjacent to cell i in placement A, and compute the **g**eometric **c**enters (GC) of this group in both placement A and B as (x_i^{GC}, y_i^{GC}) and $(\hat{x}_i^{GC}, \hat{y}_i^{GC})$ as follows.

$$\begin{aligned} x_i^{GC} &= \frac{max(x_j)+min(x_j)}{2} \\ y_i^{GC} &= \frac{max(y_j)+min(y_j)}{2} \\ \hat{x}_i^{GC} &= \frac{max(\hat{x}_j)+min(\hat{x}_j)}{2} \\ \hat{y}_i^{GC} &= \frac{max(\hat{y}_j)+min(\hat{y}_j)}{2} \end{aligned} \tag{8}$$

where j refers to cells within a certain Euclidian distance to cell i in placement A. We can then define the *relative* movement of a cell i from placement A to B as the squared Euclidian distance of the relative positions of cell i to $(x_i^{GC}, y_i^{GC})$and $(\hat{x}_i^{GC}, \hat{y}_i^{GC})$, as shown in following equation:

$$\begin{aligned} R_i &= [(\hat{x}_i - \hat{x}_i^{GC}) - (x_i - x_i^{GC}]^2 \\ &\quad +[(\hat{y}_i - \hat{y}_i^{GC}) - (y_i - y_i^{GC})]^2 \end{aligned} \tag{9}$$

The new placement stability metric R_i for each cell i essentially captures the relative change w.r.t. to its geometric neighborhood. Figure 7 shows two placements of cell i and its adjacent cells in the left placement, assuming the left placement is the original placement and the right one is after placement migration. Suppose originally cell i is placed at $(4,8)$ and the geometric center of its adjacent cells is at $(6,6)$. After placement migration, cell i is moved to $(4,6)$ while all of its original neighbors are shifted to upper right corner with a center at $(18,20)$. The relative positions of cell i to the center of its adjacent cells are shown as vectors in Figure 7. Although the absolute location of cell i does not change much, the relative distance actually changes a lot, $R_i = [(4-18)-(4-6)]^2 + [(6-20)-(8-6)]^2 = 400$ as given by Eqn. (9), which can not be captured by the absolute cell movement between both placements. Note that we use squared distance to emphasize the impact of larger moves, since wiring delay is a quadratic function of wirelength. Larger moves will have higher possibility to degrade the overall timing closure.

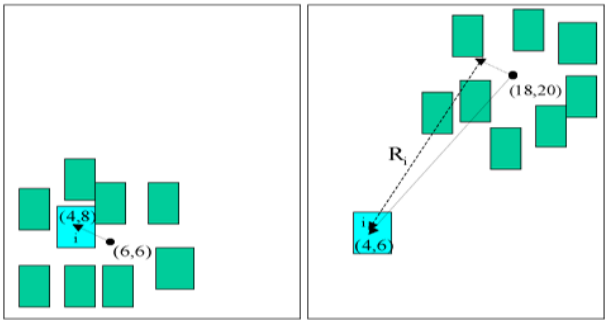

Figure 7: Relative distance of cell *i*.

Naively one may sum up R_i for all cells to measure the total geometric stability. However, it is the most disruptive changes of the relative order that have the biggest impact on the placement migration quality. Therefore, we set some filter and only count the top percent of cells in terms of R_i, e.g., top 1% cells. So, the overall geometry stability metric S_G can be written as

$$S_G = \frac{\sum_{j \in \mathbf{C}_{1\%}} R_j}{N} \tag{10}$$

where $\mathbf{C}_{1\%}$ is the set of top 1% cells with the largest R_j values and N is the number of cells in $\mathbf{C}_{1\%}$. Bigger S_G value means placement B is less similar to the original placement A, and more cells are placed away from their original affinity logics, thus more vulnerable to performance degradation.

Table 1: TWL Comparison (scaled)

testcases	*GREEDY*	*FLOW*	CGL_B	*CGL*
ibm01	1.282	1.314	1.192	1.191
ibm02	1.056	1.064	1.013	1.013
ibm03	1.101	1.096	1.058	1.061
ibm04	1.165	1.195	1.096	1.100
ibm05	1.016	1.018	1.014	1.014
ibm06	1.111	1.123	1.036	1.035
ibm07	1.141	1.139	1.040	1.039
ibm08	1.154	1.153	1.043	1.042
ibm09	1.211	1.221	1.071	1.069
ibm10	1.180	1.181	1.032	1.030
ibm11	1.193	1.187	1.054	1.056
ibm12	1.162	1.167	1.051	1.050
ibm13	1.229	1.233	1.064	1.059
ibm14	1.239	1.235	1.049	1.047
ibm15	1.274	1.273	1.067	1.065
ibm16	1.313	1.318	1.041	1.040
ibm17	1.267	1.285	1.043	1.040
ibm18	1.296	1.318	1.055	1.054
Average	1.188	1.197	1.057	1.056

Table 2: S_G Comparison

testcases	*GREEDY*	*FLOW*	CGL_B	*CGL*
ibm01	52199	37785	12685	11349
ibm02	44735	24126	5225	5172
ibm03	58375	52356	41099	26081
ibm04	119647	58967	52199	39039
ibm05	55040	65344	53387	49538
ibm06	60242	48890	4555	4750
ibm07	131024	119809	5232	5764
ibm08	153395	119758	3610	3750
ibm09	168791	154334	20743	16126
ibm10	302222	238185	5057	4740
ibm11	224925	190321	32519	50898
ibm12	361702	331289	5270	5088
ibm13	370440	349016	177377	117833
ibm14	557898	456627	4581	4364
ibm15	698634	746372	130784	123811
ibm16	937453	746372	4435	4129
ibm17	1169281	1087672	14648	11019
ibm18	1116867	985042	3224	3267
Average	365715	334406	31855	26858

6. EXPERIMENTAL RESULTS

We implemented the computational geometry placement migration algorithms in C++ on a 3.4 GHZ Xeon Linux box, and use the modified *ISPD*02 benchmark [14] to test it. All the testcases of this benchmark place cells with equal density (95%) over the entire chip. To make it similar to the real industry placement after physical synthesis, we first linearly scale down each cell such that the entire chip density for each testcase is reduced to 80%. We then run a detailed placement algorithm to reduce the wirelength. After that, the placement is no longer equally distributed. To test the legalization algorithm, we generate the overlaps by expanding cells in the center of the chip. We linearly expand 15% of cells in the center by 67%. So after expansion, the overall chip density is increased to 90% and this original placement is no longer legal.

We compare the computational geometry based legalizer (*CGL* and CGL_B) to an industry greedy legalizer (*GREEDY*) which uses slide-and-spiral techniques to place cells onto their nearest legal locations and to an industry network flow legalizer (*FLOW*) which uses min-cost flow algorithm to direct cell movements. *FLOW* is an industrial strength legalizer similar to [3]: first, cells are roughly spread out by the min-cost flow algorithm; then, they are moved to their final positions such that all overlaps are removed. *GREEDY* sorts all the cells and place them sequentially. It first tries to place a cell at the original location. If that location is occupied, it performs a spiral search starting from the original location. During a spiral search, it could slide other placed cells a little bit in order to fit in.

Table 1 reports the TWL results of these legalizers. The TWL numbers are scaled to the TWL of the original illegal placement. Both CGL_B and *CGL* get much better TWL than *GREEDY* and *FLOW*. The improvement is over 10% on average.

Table 2 shows the geometric stability scores of these legalizers. Both CGL_B and *CGL* get order of magnitude better scores than *FLOW* and *GREEDY*. And *CGL* can further reduce it by 16% than CGL_B due to the Delaunay triangle based overlap removal.

To further understand the difference of geometric stability result of these approaches, Figure 8 and 9 show the relative distance histogram after legalization on *ibm*01. Figure 9 is a zoom-in view of the top 1% cells in Figure 8 to make it difference clear. *CGL*

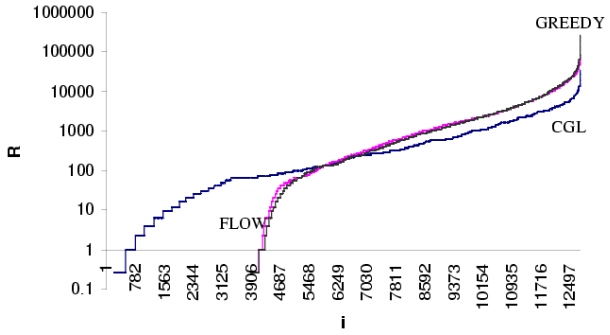

Figure 8: Histogram of R_i from three legalizers on *ibm*01.

has less larger relative distances than *FLOW* and *GREEDY*, which means *CGL* would keep cells closer to their original neighbors, preserving the relative order. On the other hand, *GREEDY* and *FLOW* have less number of smaller relative distances than *CGL*, which means *CGL* tends to move more cells a smaller distance to avoid bigger moves. For placement migration applications, one often wants to limit the bigger moves but not care too much about the smaller moves, therefore *CGL* is well targeted for those applications.

The runtime comparison of is reported in Table 3. We can see *CGL* is much faster than *GREEDY* and *FLOW* (over 10x speedup for bigger circuits). Therefore, it is both effective and fast.

We also implemented the diffusion-based legalization algorithm [15] for the academic benchmarks. Our initial experience is that the diffusion algorithm produces slightly better result while computational geometry based algorithm is faster. However, it is not straightforward to make a fair comparison between them at this stage, because the results depend on the tuning. In general both algorithms share a common smooth spreading nature and generate comparable results.

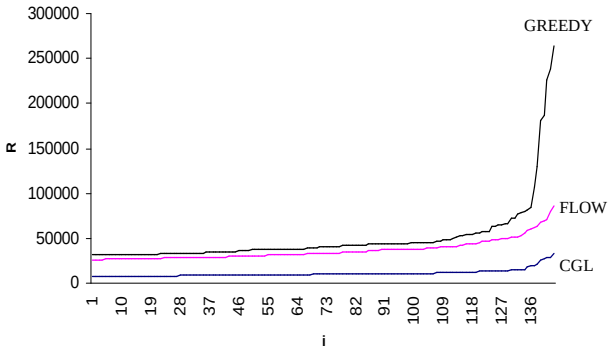

Figure 9: Histogram of top 1% R_i from three legalizers on *ibm*01.

Table 3: CPU Time Comparison (s)

testcases	*GREEDY*	*FLOW*	*CGL*
ibm01	11	10	10
ibm02	23	25	23
ibm03	28	26	25
ibm04	62	29	30
ibm05	33	37	40
ibm06	69	53	35
ibm07	162	150	55
ibm08	307	242	65
ibm09	325	289	75
ibm10	806	575	89
ibm11	617	490	95
ibm12	1299	807	121
ibm13	939	734	134
ibm14	3654	2240	231
ibm15	5210	3643	430
ibm16	9927	6280	511
ibm17	11363	8280	558
ibm18	12724	9160	661

7. CONCLUSIONS

The incremental nature of design optimization demands smooth and stable placement mitigation techniques. They must be capable of spreading cells to satisfy design constrains such as image space, routing congestion, signal integrity and heat distribution, while keeping the original relative order. To address these challenging tasks, we propose a novel computational geometry based placement migration framework. Our experimental results on legalization problem have demonstrated significant improvements on wire length and stability. To the best of our knowledge, this is the first attempt using Delaunay triangulation to perform placement spreading. We believe there is still a lot of room to improve and other effects such as timing and wirelength to be incorporated.

8. REFERENCES

[1] H. Ren, D. Z. Pan, and P. Villarrubia, "True crosstalk aware incremental placement with noise map," in *Proc. Int. Conf. on Computer Aided Design*, pp. 616–619, 2004.

[2] F. M. J. Konrad Doll and K. J. Antreich, "Iterative placement improvement by network flow methods," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13(10), 1994.

[3] U. Brenner, A. Pauli, and J. Vygen, "Almost optimum placement legalization by minimum cost flow and dynamic programming," in *Proc. Int. Symp. on Physical Design*, pp. 2–9, 2004.

[4] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden, "Fractional cut: improved recursive bisection placement," in *Proc. Int. Conf. on Computer Aided Design*, pp. 307–310, 2003.

[5] A. B. Kahng, I. L. Markov, and S. Reda, "On legalization of row-based placements.," in *ACM Great Lakes Symposium on VLSI*, pp. 214–219, 2004.

[6] S. W. Hur and J. Lilis, "Mongrel: hybrid techniques for standard cell placement," in *Proc. Int. Conf. on Computer Aided Design*, pp. 165–170, 2000.

[7] A. B. Kahng, P. Tucker, and A. Zelikovsky, "Optimization of linear placements for wirelength minimization with free sites," in *Proc. Asia and South Pacific Design Automation Conf.*, pp. 18–21, 1999.

[8] U. Brenner and J. Vygen, "Faster optimal single-row placement with fixed ordering," in *Proc. Design, Automation and Test in Eurpoe*, pp. 117–121, 2000.

[9] Z. Xiu, J. D. Ma, S. M. Fowler, and R. A. Rutenbar, "Large-scale placement by grid-warping," in *Proc. Design Automation Conf.*, pp. 351–356, 2004.

[10] N. Viswanathan and C. C. N. Chu, "Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. Int. Symp. on Physical Design*, pp. 26–33, 2004.

[11] J. Kleinhans, G. Sigl, F. M. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-10, pp. 356–365, Mar. 1991.

[12] Fortune, "Voronoi diagrams and delaunay triangulations," in *Computing in Euclidean Geometry, 2nd Edition, World Scientific, Lecture Notes Series on Computing – Vol. 1*, 1992.

[13] C. J. Alpert, G.-J. Nam, P. Villarrubia, and M. C. Yildiz, "Placement stability metrics," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan, 2005.

[14] ISPD2002BenchmarkModified, "http://gibbon.uwaterloo.ca/research.html," 2002.

[15] H. Ren, D. Z. Pan, C. J. Alpert, and P. Villarrubia, "Diffusion-based placement migration," in *Proc. Design Automation Conf.*, June, 2005.