Lossless Compression Algorithm for Hierarchical IC Layout

by

Allan Gu

B.S. University of California, Berkeley 2004

A thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Engineering - Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Avideh Zakhor, Chair
Professor Andrew Neureuther

Spring 2007

The thesis of Allan Gu is approved.

Chair	Date
	Date

University of California, Berkeley

Spring 2007

Lossless Compression Algorithm for Hierarchical IC Layout

by

Allan Gu

Abstract

Lossless Compression Algorithm for Hierarchical IC Layout

by

Allan Gu

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Avideh Zakhor, Chair

An important step in today's Integrated Circuit (IC) manufacturing is optical proximity correction (OPC). While OPC increases the fidelity of pattern transfer to the wafer, it also significantly increases IC layout file size. This has the undesirable side effect of increasing storage, processing, and I.O. times for subsequent steps of mask preparation. To alleviate the growing volume of layout data, a new layout data format, Open Artwork System Interchange Standard (OASIS), was introduced in 2001 by SEMI's Data Path Task Force. Even though OASIS results in a more efficient representation than the previous industry standard format GDSII, there is still room for improvement by applying data compression techniques. In this paper, we propose two such techniques for compressing layout data, including OPC layout, while remaining complaint with existing industry standard formats such as OASIS and GDSII. Such compliance ensures that the resulting compressed files can be viewed, edited, and manipulated by industry standard CAD viewing and editing tools without the need for a decoder. Our approach is to eliminate redundancies in the representation of the geometrical data by finding repeating groups of geometries between multiple cells and within a cell. We refer to the former as "inter-cell sub-cell detection (InterSCD)" and latter as "intra-cell sub-cell detection (IntraSCD)". We show both problems to be NP hard, and propose two sets of heuristics to solve them. For OPC layout data, we

also propose a fast compression method based on IntraSCD which utilizes the hierarchical information in the pre-OPC layout data. We show that the IntraSCD approach can also be effective in reconstructing hierarchy from flattened layout data. We demonstrate the results of our proposed algorithms on actual IC layouts for 90nm, 130nm, and 180nm feature size circuit designs.

Professor Avideh Zakhor
Thesis Committee Chair

Contents

Acknowledgements

This work was conducted under the Research Network for Advanced Lithography, supported jointly by the Semiconductor Research Corporation and the Defense Advanced Research Project Agency.

I would like to thank Robert Gleason for his input on the process of transforming circuit design into masks for lithography. I would also like to thank my advisor, Avideh Zakhor, for all of her guidances. I am very grateful for all of her help with my writing and speaking. Avideh is not only an excellent researcher, but also an excellent mentor.

Finally, I would like to thank my family, friends, and all the members of the VIP lab for all of their help and support.

Chapter 1

Introduction

As the semiconductor industry moves toward denser designs with smaller feature sizes, pattern transfer from reticles to wafers, referred to as lithography, becomes more challenging. To correctly fabricate these circuits using current lithographic machines, resolution enhancement techniques (RET) such as optical proximity correction (OPC), phase shift masking, scattering bars, and tiling are routinely performed on the layout data [1]. Denser circuit designs and increased usage of RET have resulted in significant data volume explosion. Specifically, The International Technology Roadmap for Semiconductors indicates that a single layer of uncompressed fractured layout will exceed 400 Gigabytes in 2007 [2], and GDSII layout file sizes are likely to grow to many gigabytes [3]. In particular, OPC is a major contributor to the expansion of layout data volume. OPC often destroys hierarchical structures in layout designs, and adds vertices to polygons causing over 10X increase in file size.

There exist compression algorithms to reduce the mask data size in the rasterized domain for direct write lithography system [4], [5]. There are also algorithms which can be adapted to compress hierarchical IC layout data. Specifically, Chen *et al.* [6] have investigated algorithms to compress dummy fills in IC layouts which exhibit high degree of spatial regularity. Veltman and Ashida [7] propose a compression technique for E-Beam writers

by finding a set of polygons with identical repetitions that can be referenced as a single geometrical library.

In this thesis, we propose two compression techniques to reduce the layout data size by finding repeating groups of polygons in the layout. Our techniques are designed in such a way that the resulting compressed layouts remain compliant with standard industry formats such as GDSII and OASIS, and can therefore be read by industry standard CAD viewing and editing tools without a decoder. In Section 2, we describe the problem of finding repeating groups of geometries between multiple cells and within a cell. We refer to these problems as inter-cell sub-cell detection (InterSCD) and intra-cell sub-cell detection (IntraSCD) respectively. In Section 3, we present a set of greedy algorithms to solve these two problems. In Section 3.4, we extend the IntraSCD algorithm to exploit the hierarchical information in the pre-OPC layout in order to compress the post-OPC layout; in doing so, we achieve a factor of five speed up with little or no loss in compression efficiency as compared to the IntraSCD method in Section 3.2. Section 4 discusses experimental results on actual IC layout data. Finally, conclusions and future research directions are included in Section 5.

Chapter 2

Sub-cell Detection Problem

IC layouts have a well defined hierarchical structure, and layout interchange formats such as OASIS and GDSII provide syntax to describe the hierarchy efficiently. However, the hierarchical structure is partially destroyed during the OPC process. Despite this, it is possible to reconstruct some hierarchy by finding groups of polygons that undergo the same proximity correction. Empirical observation of post-OPC data reveals repeating groups of polygons both across multiple cells and within a cell. As shown later, we exploit both of these redundancies in reducing the file size of the semi-hierarchical post-OPC layout data.

We begin by defining terminologies used throughout the paper. We define rectangle, trapezoid, polygon, and placement as geometries. A placement is a reference to another cell in the layout. A cell is a collection of geometries in a two dimensional plane, and a sub-cell is a subset of the geometries that are within a cell. A rigid transformation is associated with each placement. Two geometries are the same if they are of the same geometrical shape; in the case of placement, they need to reference the same cell, and have the same type of transformation. The compression ratio (CR) is the ratio of the size of the OASIS layout file to the size of its compressed version.

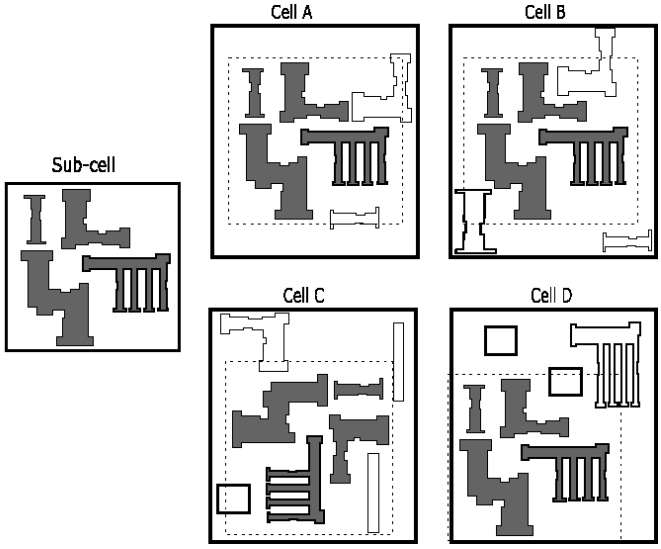


Figure 2.1. *Repeating group of polygons across multiple cells.*

2.1 Inter-cell Sub-cell Detection

In the InterSCD problem, we wish to find a group of geometries that appear in two or more cells. In OASIS, geometries are defined each time they occur in a cell. For instance, if a group of 4 geometries occur in N different cells, then they result in $4N$ definitions when only 4 definitions would suffice. By detecting this group of 4 geometries, it is possible to create one cell from them which can then be referenced by each of the N cells with a placement operator. Figure 2.1 shows an example of four cells and a group of 4 polygons that occur in each of the four cells. Rather than defining the 4 polygons separately in each cell, we create a placement in each of the cells that references a new cell containing the 4 polygons. In this case, it is sufficient to define the 4 polygons once rather than 4 times. In Figure 2.1, the placement in cells A, B, and D are translated version of the sub-cell, and the placement in cell C is a rotated and translated version of the sub-cell. We now formally define the InterSCD problem:

Inter-cell Sub-cell Detection Problem: Given m cells, $\{C_1, C_2, ..., C_m\}$, find the sub-cell which maximizes $|SC_r| * r$ for $m \geq r \geq 2$.

A sub-cell SC is defined to occur in a cell C if there exists a transformation L that maps

every geometry in SC to some geometry in C. $|SC_r|$ denotes the number of geometries in the sub-cell, and r is the number of cells that SC_r occurs in. This problem is NP hard since it is a special case of the largest common point set (LCP) problem [8] with $r = m$, each geometry mapped to a point, and each cell mapped to a point set. In the LCP problem, for a collection of d-dimensional point sets $SS = \{S_1, S_2, ..., S_m\}$, the objective is to find a maximal set U that is congruent to some subset of S_i for $i = \{1, 2, ..., m\}$. A set U is congruent to a set V if there exists a transformation that takes U into V.

2.2 Intra-cell Sub-cell Detection

In the IntraSCD problem, we wish to find groups of geometries that occur at multiple locations within a cell. The OASIS format provides different operators for representing repetitive geometries [3]. In this paper, we assume that all repetitive geometries are represented with the "TYPE 10" repetition operator. With the "TYPE 10" operator, representing N instances of a geometry requires one geometry definition and N two dimensional coordinates. Compression is achieved by finding sub-cells which occur multiple times within the cell. For instance, 4 polygons occurring N times in a cell would require 4 definitions and $4N$ coordinates to represent. Grouping the 4 polygons together into one cell would only require N rather than $4N$ coordinates. Figure 2.2 shows a cell with 30 polygons and a group of 4 polygons that occur four times in the cell. Rather than using 16 coordinates to represent the 16 polygons, only 4 coordinates are used to create 4 placements in the cell that reference the sub-cell. In Figure 2.2, the first, second, and fourth placements are translated versions of the sub-cell, and the third placement is a rotated and translated version of the sub-cell. We now formally define the IntraSCD problem:

Intra-cell Sub-cell Detection Problem: Given a cell, C, find the sub-cell SC_r which maximizes $|SC_r| * r$ for $2 \leq r \leq m$, subject to the constraint that the maximum Euclidean distance between any two geometries in SC_r is less than or equal to $dist$.

The maximum Euclidean distance between two geometries is constrained because most

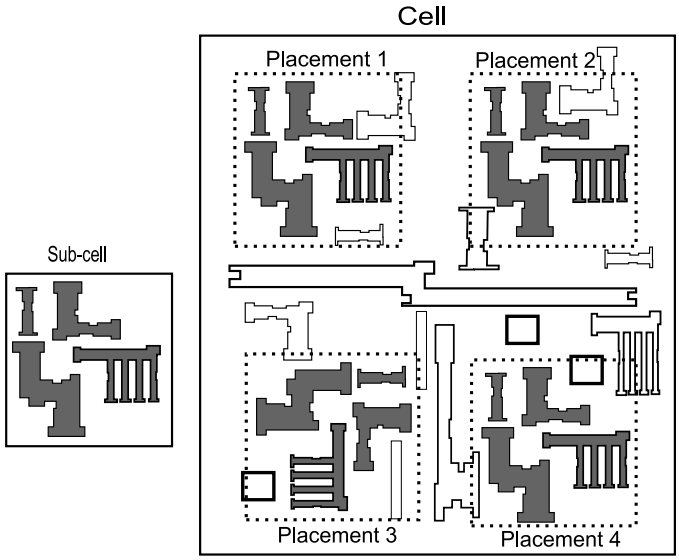


Figure 2.2. *Repeating group of polygons within a cell.*

circuit designs are created by connecting smaller functional circuit units together, and the smaller circuits are limited in size. A sub-cell SC occurring in r locations implies that there exist r transformations, $T_1, T_2, ..., T_r$ such that $T_i(SC)$ maps uniquely to a group of geometries in C. m denotes the frequency of the most repeated geometry in C. As shown in the appendix, IntraSCD is an NP hard problem.

Chapter 3

Sub-cell Detection Algorithms

InterSCD and IntraSCD are both NP hard problems, and cannot be solved optimally within a reasonable amount of time for large layouts. In this Section, we describe two greedy algorithms to solve them. Our proposed approach to the InterSCD problem currently detects groups of geometries that are translation invariant. Future research will address rotation and reflection invariant cases.

3.1 Inter-cell Sub-cell Detection Algorithm

Before detecting a common sub-cell among a large collection of cells, the cells are pre-processed using hierarchical clustering algorithm to group similar cells together. This results in computational efficiency because cells that do not share any geometries with other cells are quickly eliminated from further consideration. Hierarchical clustering begins by assigning each of the N cells in the layout to a separate cluster. Then the two clusters that are most similar according to a distance metric are merged together into a single cluster. The clustering algorithm re-computes the distance between the new cluster and the remaining clusters, and again merges the two most similar clusters. This is repeated until all of the clusters have been merged into a single cluster. The distance between two clusters is defined as:

$$d(Cluster_i, Cluster_j) = \frac{\sum_{m=1}^{N_i} \sum_{n=1}^{N_j} d(C_m^i, C_n^j)}{N_i * N_j} \tag{3.1}$$

where

$$d(C_m^i, C_n^j) = \frac{m - w}{m}, \tag{3.2}$$

and

$$w = |common\_shape(C_m^i, C_n^j)| \tag{3.3}$$

$$m = \min(|C_m^i|, |C_n^j|) \tag{3.4}$$

N_i and N_j denote the number of cells in the ith and jth cluster respectively, and $d(C_m^i, C_n^j)$ is the distance between the mth cell in cluster i and nth cell in cluster j. *common_shape* is a function that determines the number of geometries that cells C_i, C_j have in common irrespective of their locations. The distance between two clusters is the average of the distances from any cell in one cluster to any other cell in the other cluster. Once hierarchical clustering is completed, a collection of clusters are generated by cutting the hierarchical tree at a certain height in such a way that each cluster contains cells that most likely share a common group of geometries. We have empirically determined to cut the tree at the height in which the distance between two clusters exceeds 0.35. Figure 3.1 shows an example of a hierarchical cluster tree created after the clustering process. As seen, cutting the tree at distance 0.35 results in 3 clusters, namely $\{C1, C2, C5\}, \{C3, C6\}$, and $\{C4\}$.

Having obtained a collection of clusters through the above hierarchical clustering and cutting procedure, for each cluster the algorithm looks for a sub-cell which maximizes $|SC_r| * r$, where r is the number of cells the sub-cell occurs in for that cluster, and $|SC_r|$ is the number of geometries that the sub-cell contains. Figure 3.2 shows the flowchart for

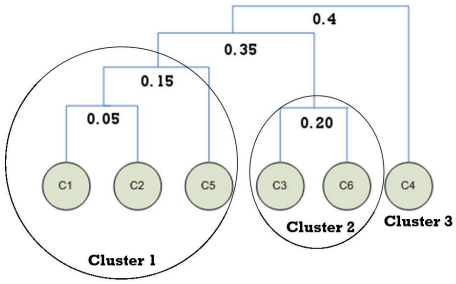


Figure 3.1. *Hierarchical clustering example.*

our proposed InterSCD algorithm. The basic idea behind the algorithm is to recursively update the candidate sub-cell SC^* which maximizes the benefit function $|SC_r| * r$ at each iteration as it goes through all the cells in the cluster one at a time. In the first stage, the algorithm starts by choosing and removing two cells, C_i and C_j, from the cluster that are closest in terms of the distance metric in Equation(3.2). It then exhaustively searches for the largest sub-cell, $SC^{(1)}$, that is common to both cells under translation in a manner to be described shortly. $SC^{(1)}$ is set as the initial sub-cell if its number of geometries exceeds some threshold. Otherwise, another pair of cells whose distance is the next closest is chosen.

Having found the largest sub-cell, $SC^{(1)}$, between C_i and C_j in the first stage, the algorithm sets $SC^* \leftarrow SC^{(1)}$, and $numC \leftarrow 2$ where in general $numC$ denotes the number of cells which contain SC^* as a sub-cell. It then moves on to the next stage as it finds more cells in the cluster that contain overlapping geometries with SC^*. Specifically, at stage 2, the algorithm re-computes the distance between SC^* and the remainder of the cells in the cluster according to Equation(3.2). The cell that is closest to SC^*, i.e. $C^{(2)}$, is chosen from the cluster; then, exhaustive search is applied to find the largest sub-cell, $SC^{(2)}$, between SC^* and $C^{(2)}$. At this point, we need to decide whether to update SC^* with $SC^{(2)}$ as the possible candidate to be considered in future stages. Our approach is to update $SC \leftarrow SC^{(2)}$ if $|SC^{(2)}| * (numC + 1) > |SC^*| * numC$. The reason for having $numC + 1$ in the left side of the inequality is that at this point $SC^{(2)}$ is known to have appeared in 3 cells while SC^* has appeared in only 2 cells. If $SC \leftarrow SC^{(2)}$, then $numC \leftarrow numC + 1$. The algorithm

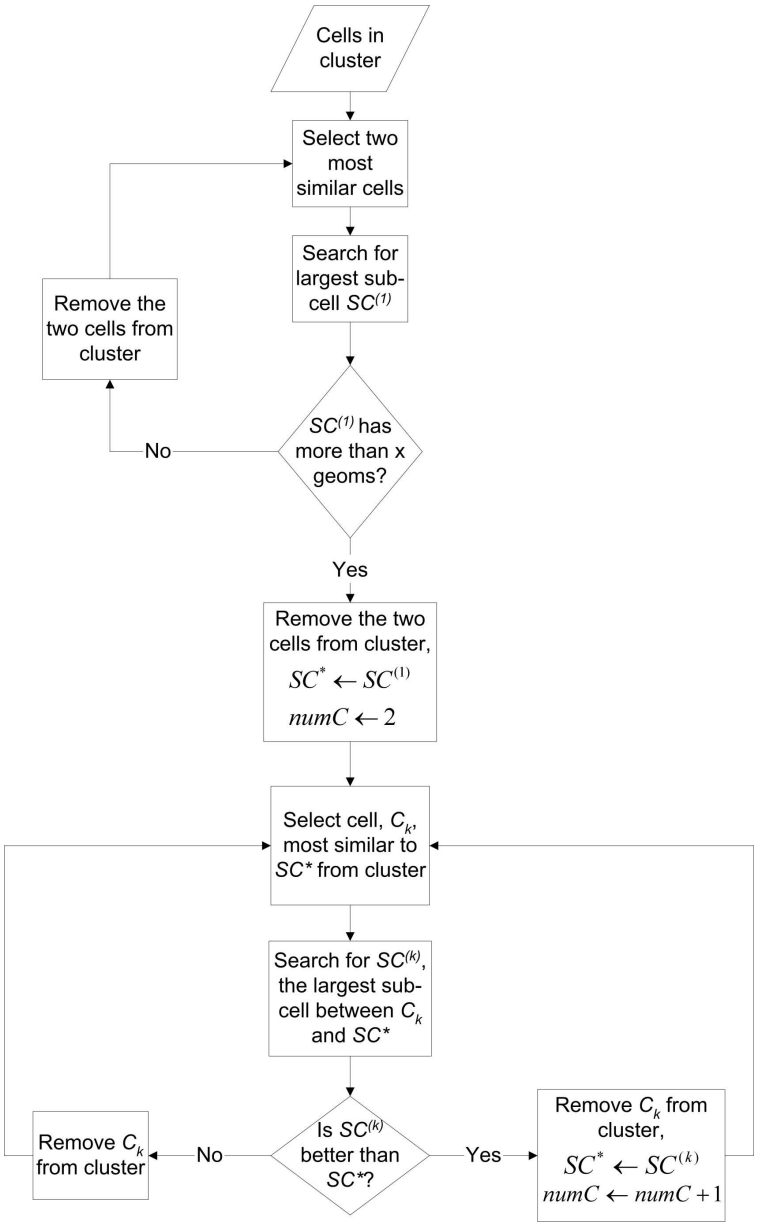


Figure 3.2. *Flowchart of the InterSCD algorithm.*

then proceeds onto stage 3, and follows the same steps taken in stage 2. This process is repeated until all the cells in the cluster have been visited.

From the above description, it is clear that a major component of the described algorithm has to do with finding the largest group of overlapping geometries between 2 given cells, C_i and C_j. Our approach for the above problem is to perform an exhaustive search as follows: for every geometry G that occurs in both C_i and C_j, the algorithm finds a translation mapping, Γ, that takes G in C_i to C_j. This mapping is applied to all of the geometries in C_i, and the number of geometries that $\Gamma(C_i)$ and C_j have in common is determined. The group with the most number of common geometries is selected as the largest group of overlapping geometries.

Figure 3.3 shows an example of how the above approach works. After the hierarchical clustering step, cells A, B, C, and D are assumed to be grouped together in a cluster. Cells A and B are the closest with 6 geometries in common. The exhaustive search finds the largest group of geometries, $SC^{(1)}$, that occurs in cells A and B, and sets $SC^* \leftarrow SC^{(1)}$. Cell C and SC^* are the closest, and $SC^{(2)}$ is the largest group of geometries between cell C and SC^*. Because $SC^{(2)}$ has 4 geometries occurring in three cells, while SC has 4 geometries occurring in two cells, the algorithm updates SC^* as $SC^* \leftarrow SC^{(2)}$. Finally $SC^{(3)}$ is the sub-cell found in the third stage. $SC^{(3)}$ has 3 geometries occurring in all four cells as compared to $SC^{(2)}$ which has 4 geometries occurring in 3 cells; since $|SC^{(3)}| * 4 = 12$ is not greater than $|SC^*| * 3 = 12$, we do not update SC^*. Hence, the final solution as computed by the proposed InterSCD algorithm is $SC^* \leftarrow SC^{(2)}$.

3.2 Intra-cell Sub-cell Detection Algorithm

For IntraSCD, we have developed a greedy algorithm that grows the solution sub-cell at each iteration. The basic idea behind our proposed iterative algorithm is to select an initial geometry as an initial sub-cell, and to add more polygons to the sub-cell until there is no additional benefit in adding more polygons. Once this happens, we replace all the

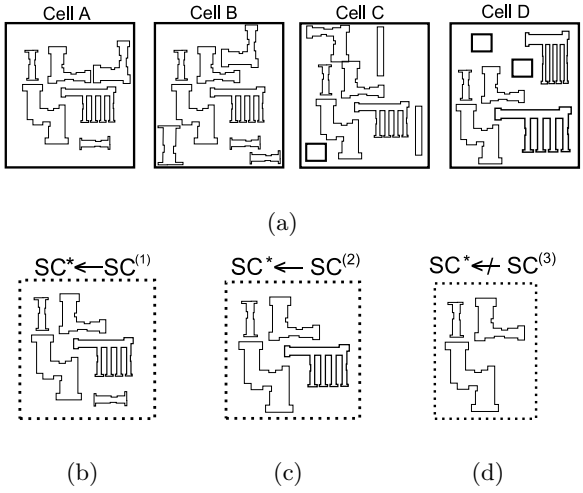


Figure 3.3. *Inter-cell sub-cell detection example. (a) Cell cluster; (b) sub-cell between cells A and B; (c) sub-cell between $SC^{(1)}$ and cell C; (d) sub-cell between $SC^{(2)}$ and cell D.*

geometries in the cell corresponding to the newly found sub-cell with a reference to the sub-cell, and repeat the above process for the remaining geometries in the cell.

Figure 3.4 shows the flow diagram of our proposed IntraSCD algorithm. The algorithm begins by ranking all the geometries according to the number of repetitions of each geometry in the cell. Here, we are primarily concerned with repetitions under translation. In Section 3.3, we will extend this algorithm for rotations and reflections. The geometry, G^{max}, with the most number of repetitions is selected, and set to $SC^{(0)}$ if its number of repetitions is greater than some threshold. Let G_k^{max} denote the kth instance of the geometry in the cell; then for each instance G_k^{max}, all possible combinations of 2 or 3 geometries are created using G_k^{max} and its closest neighbors that are within a certain distance from it. In doing so, the number of neighbors is limited to 200 so as to limit complexity. There are $\binom{200}{2}$= 19,900 combinations of 2 geometries that can be paired with G_k^{max} to form a group of 3 geometries. If there are 2000 instances of G^{max}, then there are over 39 million candidate groups to consider. Hence, even modest number of instances of G^{max} results in large number of candidate groups requiring significant amount of computation to select the best group.

To alleviate this, we have devised a pruning method to eliminate candidates that result in

few instances after adding 1 geometry to $SC^{(0)}$. Specifically, assume the maximum number of instances for a candidate group with 1 added geometry is N; then there is no need to add a second geometry to any of these candidates with M instances if $M < \frac{2N}{3}$. This is because at each iteration, the goal is to choose the candidate sub-cell which maximizes the benefit; therefore, even in the best case scenario whereby the number of occurrences for a candidate group with 1 added geometry remains at M after the addition of a 2^{nd} geometry, the total score for this candidate group is still less than $2N$. In general, assume $SC^{(i)}$ has l geometries, and the maximum number of instances for a candidate group composed of $SC^{(i)}$ and one other geometry is N; then, there is no need to add a second geometry to any candidate groups composed of $SC^{(i)}$ and another geometry having M instances if $M < \frac{l+1}{l+2}N$.

At the end of the first iteration, the best candidate group consisting of 1 or 2 added geometries to $SC^{(0)}$ is selected as follows: $SC^{(1)} \leftarrow \arg\max_{SC_j^{(1)}} |SC_j^{(1)}| * numInst_j^{(1)}$, where $SC_j^{(1)}$ is the jth candidate created during the 1^{st} iteration; the algorithm checks to see whether $|SC^{(1)}| * numInst^{(1)} \geq |SC^{(0)}| * numInst^{(0)}$. If it is, then more geometries that are within a certain distance of the bounding box of $SC^{(1)}$ are added to $SC^{(1)}$ by repeating the above process. If not, the iteration stops, the newly found sub-cell replaces the repeating group of geometries in the cell, and the process repeats by selecting another geometry in the cell as an initial sub-cell.

In general, let $SC^{(i)}$ denote the solution sub-cell at the ith iteration, and $SC_j^{(i)}$ be be the jth candidate sub-cell created during the ith iteration. We set

$$SC^{(i)} \leftarrow \arg\max_{SC_j^{(i)}} |SC_j^{(i)}| * numInst_j^{(i)}.$$

where $|SC_j^{(i)}|$ is the number of geometries in the jth candidate sub-cell generated at iteration i, and $numInst_j^{(i)}$ is the number of instances of $SC_j^{(i)}$ in the cell. After selecting the best candidate generated during iteration i, i.e. $SC^{(i)}$, we continue adding more geometries to the $SC^{(i)}$ if the following condition is satisfied,

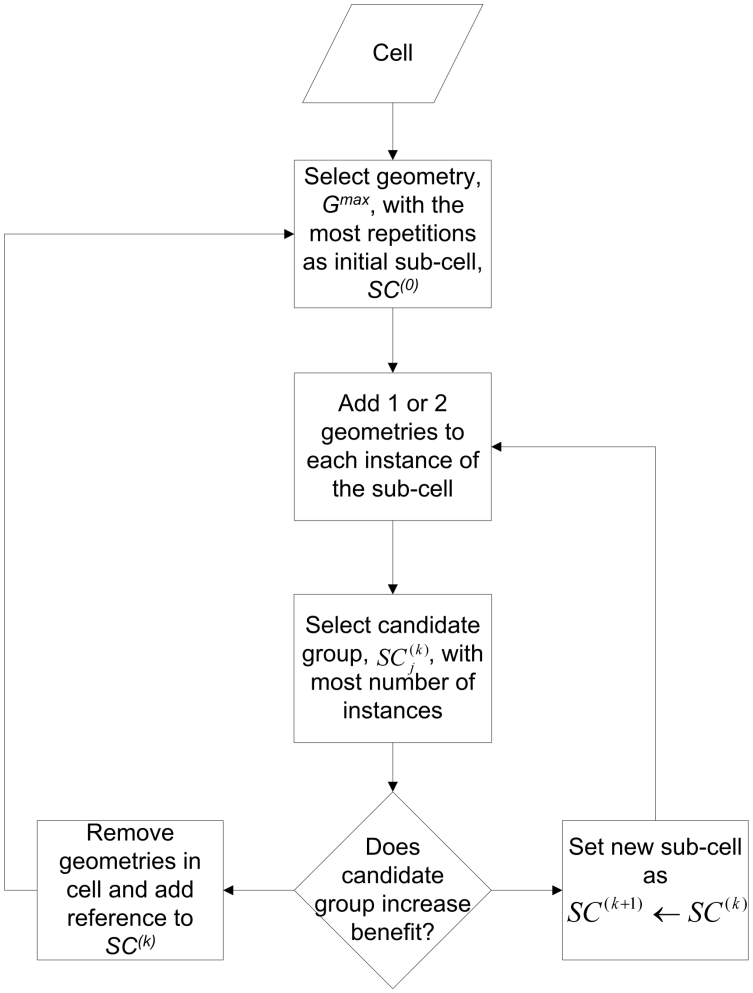


Figure 3.4. *Flowchart of the IntraSCD algorithm.*

$$|SC^{(i)}| * numInst^{(i)} > |SC^{(i-1)}| * numInst^{(i-1)}.$$

. If the condition is not satisfied, then the iterative step of adding more polygons to $SC^{(i-1)}$ ends, and placements referencing $SC^{(i-1)}$ are created at the locations where $SC^{(i-1)}$ occurs in the cell. The above process is repeated until all of the geometries in the cell have been visited to determine whether they can form repeating groups of geometries with their neighbors.

Figure 3.5 shows an example of the IntraSCD process for a cell with 31 different geometries. Initially in Figure 3.5(a), the polygon with 5 instances is selected and set to $SC^{(0)}$. Then all possible combinations of 2 and 3 geometries are formed with $SC^{(0)}$ and its neighbors. Figure 3.5(b) shows the group of three polygons that results in the highest score among all the combinations after the 1^{st} iteration. Since $|SC^{(0)}| * numInst^{(0)} < |SC^{(1)}| * numInst^{(1)}$, the algorithm continues. At the end of the 2^{nd} iteration, another polygon is added to $SC^{(1)}$ resulting in a group of 4 polygons as shown in the top sub-cell in Figure 3.5(c) called $SC^{(2)}$. Figure 3.5(c) also shows two other groups of geometries considered in the second iteration. However, these groups only occur once in the cell and are not selected. $SC^{(2)}$ with 4 geometries appearing on the top of Figure 3.5(c) is selected because it is the one that maximizes our metric, namely $|SC| * numInst$. The algorithm continues since $(|SC^{(2)}| * numInst^{(2)} = 16) > (|SC^{(1)}| * numInst^{(1)} = 12)$. In the third iteration, the algorithm attempts to add more geometries to $SC^{(2)}$. However, $(|SC^{(3)}| * numInst^{(3)} = 7) < (|SC^{(2)}| * numInst^{(2)} = 16)$, and so the iterative step of adding polygons to $SC^{(2)}$ stops. The final solution is $SC^{(2)}$ as shown in Figure 3.5(d); all the geometries corresponding to $SC^{(2)}$ are removed from the cell, and placements that reference $SC^{(2)}$ are added to the original cell as shown in Figure 3.5(d). We continue by selecting the geometry with the most repetition in the cell and setting it as an initial sub-cell. However, at this point either the remaining geometries do not have enough repetitions, or the sub-cell created after their first iteration does not satisfy the condition $|SC^{(1)}| * numInst^{(1)} > |SC^{(0)}| * numInst^{(0)}$.

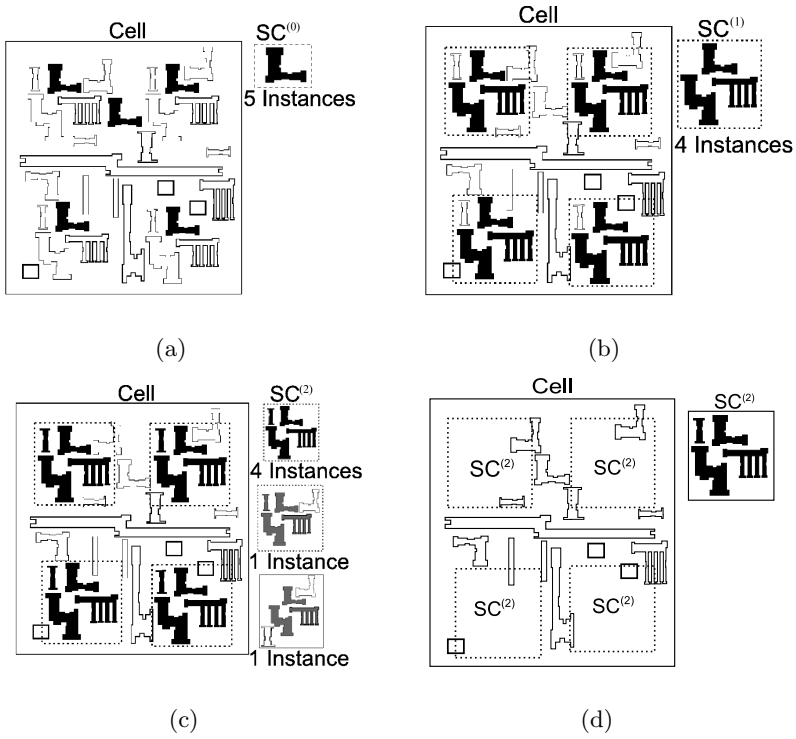


Figure 3.5. *Intra-cell sub-cell detection example. (a)* 0^{th} *iteration; (b)* 1^{st} *iteration; (c)* 2^{nd} *iteration; (d) final result.*

3.3 Extension of IntraSCD to Rotation and Reflection

The IntraSCD algorithm described above only considers geometries that are the same under translation. However, circuit designs contain rotated and/or reflected geometries, and as such, the above algorithm is unable to take advantage of those to further reduce the file size. We now extend the IntraSCD algorithm in Section 3.2 in order to take into account rotations and reflections. We refer to IntraSCD with extensions to rotation and reflection as IntraSCD+Ext.

Recall that in the algorithm of Section 3.2, the geometry with the most number of repetitions under translation is selected as an initial sub-cell. Geometries are added to the sub-cell at each iteration until there is no gain in the score by adding more polygons. To extend the algorithm to rotations and reflections, the geometry, G^{max}, with the most number of repetitions under translation, rotation, and reflection is selected as the initial sub-cell $SC^{(0)}$. Because of the Manhattan nature of layouts, we only focus on multiples of 90 degree rotations.

During the 1^{st} iteration, for each instance, G_i^{max}, we find a transformation such that $T_i(G_i^{max}) = G^{max}$, where G^{max} is a given geometry with an arbitrarily chosen orientation. Let $Group_i^{(max)}$ denote the set of geometries that are within a certain distance of G_i^{max}; then the algorithm applies the transformation, T_i, to $Group_i^{(max)}$, forms all possible candidate groups of 2 or 3 geometries containing $T_i(G_i^{max})$ and its transformed neighbors $T_i(Group_i^{(max)})$, and selects the group, $SC^{(1)}$, with the highest score using the exact same steps described in the IntraSCD algorithm in Section 3.2.

Having found $SC^{(1)}$, the algorithm proceeds in the same way as the IntraSCD algorithm. Specifically, during the kth iteration, the algorithm selects geometries in the cell that are within a certain distance to the bounding box of each instance of $SC^{(k-1)}$ denoted by $SC^{(k-1),l}$. In addition, $SC^{(k-1),l}$ contains an instance of G^{max} namely G_m^{max}, with an associated transformation T_m. The transformation, T_m, is applied to neighboring geometries of $SC^{(k-1),l}$, and candidate groups are created using each instance of $SC^{(k-1)}$ and its

transformed neighboring geometries. The candidate group with the highest score is selected as described in the IntraSCD algorithm. Specifically,

$$SC^{(k)} \leftarrow \arg\max_{SC_j^{(k)}} |SC_j^{(k)}| * numInst_j^{(k)}$$

where $SC_j^{(k)}$ is the jth candidate group generated at iteration k, and $numInst_j^{(k)}$ is the number of instances of $SC_j^{(k)}$ in the cell. We continue the iteration by adding more polygons if $|SC^{(k)}| * numInst^{(k)} > |SC^{(k-1)}| * numInst^{(k-1)}$. If the above condition is not satisfied, then a new cell that contains the geometries of $SC^{(k-1)}$ is created, and placements with the proper transformation referencing the sub-cell are created at the locations where $SC^{(k-1)}$ occurs in the cell. The iteration steps described above are repeated until all of the geometries in the cell have been examined.

Figure 3.6 shows an example of how IntraSCD+Ext algorithm works. In Figure 3.6(a), the polygon with the most number of instances under translation, rotation, and reflection is selected and set as $SC^{(0)}$. Figure 3.6(b) shows the group of three polygons that results in the highest score among all the combinations after the 1^{st} iteration. Since $|SC^{(0)}| * numInst^{(0)} < |SC^{(1)}| * numInst^{(1)}$, the iteration continues in order to add more polygons to $SC^{(1)}$. At the end of the 2^{nd} iteration, we add another polygon to $SC^{(1)}$ resulting in a group of 4 polygons as shown in the top sub-cell in Figure 3.6(c) which we call $SC^{(2)}$. Since $|SC^{(1)}| * numInst^{(1)} = 12 < |SC^{(2)}| * numInst^{(2)} = 16$, we continue the iteration. Finally, in Figure 3.6(d), we see that the group $SC^{(3)}$ occurs only once in the cell, and $|SC^{(3)}| * numInst^{(3)} = 5$ is less than $|SC^{(2)}| * numInst^{(2)} = 16$; therefore, the iteration step stops, and $SC^{(2)}$ is chosen as the solution. The algorithm continues by selecting another geometry with the most repetition in the cell and setting it as an initial sub-cell. However, at this point either the remaining geometries do not have enough repetitions, or the sub-cell created after their first iteration does not satisfy the condition $|SC^{(1)}| * numInst^{(1)} > |SC^{(0)}| * numInst^{(0)}$.

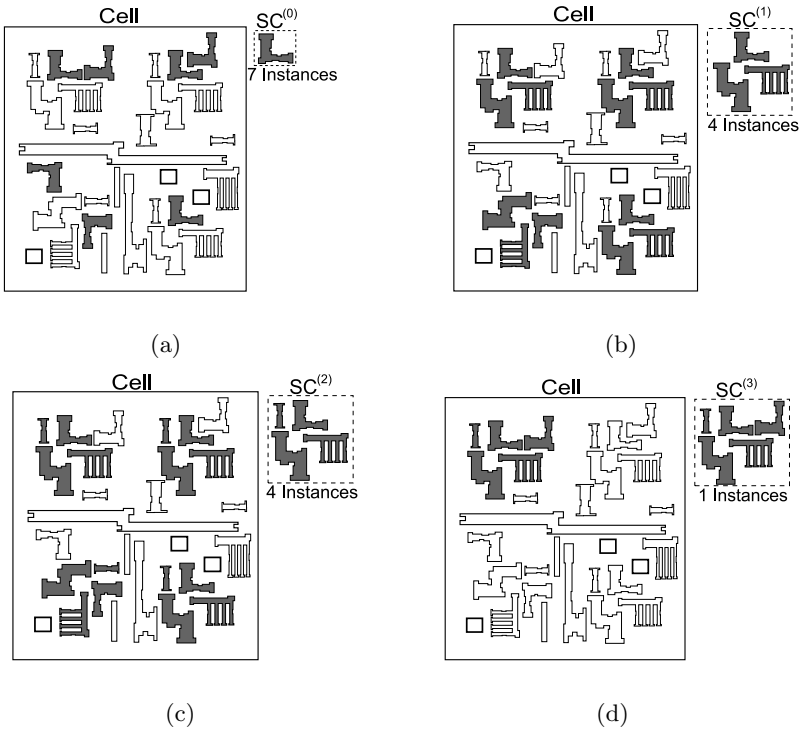
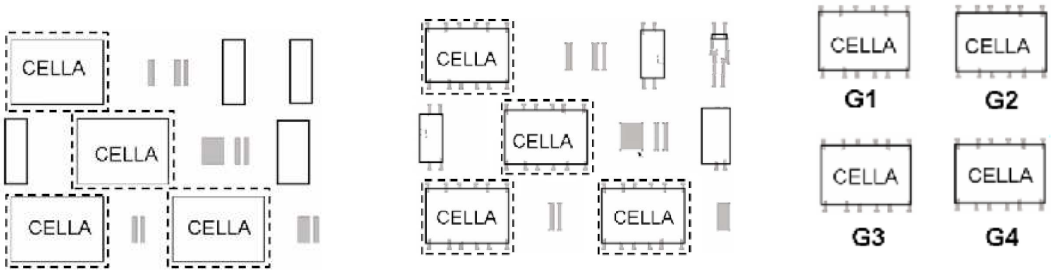


Figure 3.6. *IntraSCD example with rotation and reflection. Repeating geometries are in gray. (a)* 0^{th} *iteration; (b)* 1^{st} *iteration; (c)* 2^{nd} *iteration; (d)* 3^{rd} *iteration.*

3.4 IntraSCD Exploiting Pre-OPC Hierarchy (IntraSCD + EHier)

The greedy IntraSCD algorithm can be computationally expensive on dense layouts. Since part of the data expansion during OPC is due to the destruction of the design hierarchy, it might be possible to exploit the original pre-OPC hierarchy to reconstruct the hierarchy after OPC. As we will show shortly, in doing so, we can also speed up IntraSCD by up to a factor of 6 with little or no loss in compression efficiency.

Close examination of the post-OPC data reveals that much of the original cell hierarchy is destroyed during the OPC process, but some of the geometries from different instances of a cell undergo the same proximity correction. It is possible to find the geometries in the post-OPC layout that correspond to a particular cell instance in the original pre-OPC layout. This way, rather than having to add one or two geometries at a time within the



(a) *Pre-OPC* (b) *Post-OPC* (c) *Groups of geometries*

Figure 3.7. *An example of how to use the pre-OPC hierarchy information to find the geometries belonging to the same cell instance in the post-OPC layout.*

IntraSCD algorithm, we can find all of the geometries that belong to the same group in one step. However, due to proximity effects, not all corrections are identical for each instance of a cell; hence, we need additional processing steps in order to find the repeating group of geometries.

We begin by collecting the N groups of geometries in the post-OPC layout, $G_1, G_2, ..., G_N$, that belong to the N instances of the same cell in the pre-OPC layout. This can be done by intersecting the bounding box of each cell instance in the pre-OPC layout with the geometries in the post-OPC layout. Since OPC only makes local modifications to the polygons, the geometries that intersect with the bounding box correspond to the geometries of each instances of the same cell in the pre-OPC layout. Figure 3.7(a) shows a portion of the pre-OPC layout, and Figure 3.7(b) shows the corresponding post-OPC layout. In the figure, intersecting the bounding box of the cell 'CELLA' in the pre-OPC layout, denoted with the dotted outline, with the post-OPC layout results in 4 groups of geometries, G_1, G_2, G_3, G_4 as shown in Figure 3.7(c).

Designers create complex logic circuits by connecting smaller, simpler functional circuit units such as "AND" gates together. These smaller circuits, when placed on a layout, may be transformed geometrically to satisfy some constraints placed by designers. For instance, suppose C in Figure 3.8(a) represents a small circuit unit, and $C1, C2$ are two placements

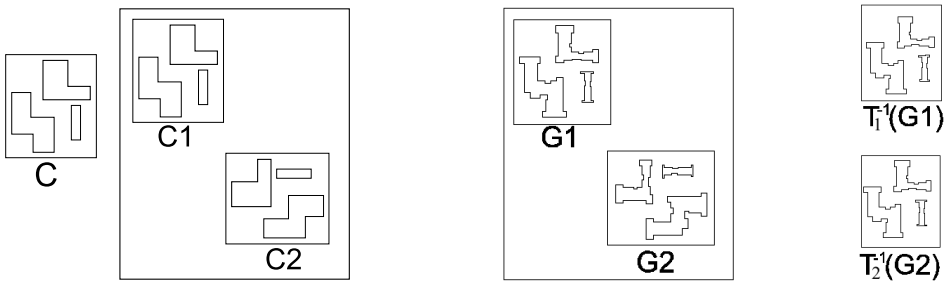
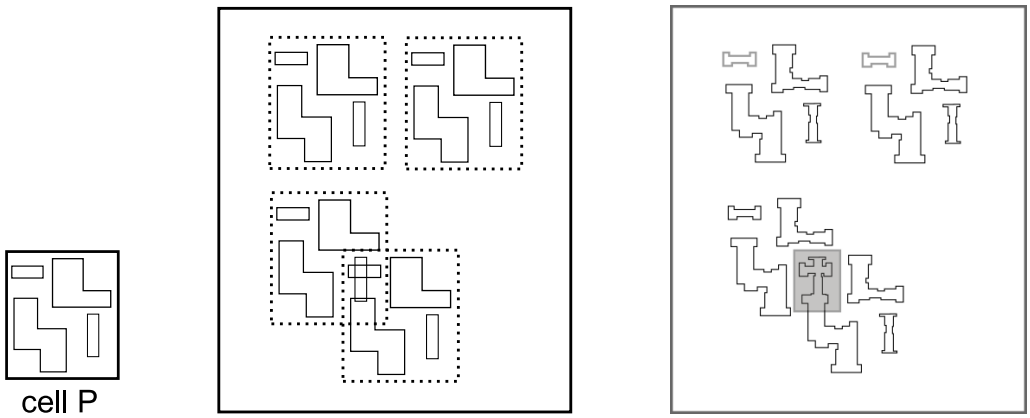


Figure 3.8. *(a) Example of smaller circuit, C, and its instances C_1, C_2 in a pre-OPC layout; (b) the corresponding post-OPC layout with the bounding box of C_1 and C_2 superimposed; (c) by applying the inverse transformations to $G1$ and $G2$, the two groups of geometries are mapped to the same location and orientation.*

that references C in a pre-OPC layout; the corresponding post-OPC layout is shown in Figure 3.8(b). Figure 3.8(b) also shows the two group of geometries $G1$ and $G2$ that can be obtained by intersecting the post-OPC geometries with the bounding box of $C1$ and $C2$ in the pre-OPC layout. As seen, $C1$ is a translated version of C, and $C2$ is a translated and 90° rotated version of C. These transformations are readily available in the pre-OPC layout data, and therefore, corresponding inverse transformations can be applied to $G1$ and $G2$. In doing so, we map all of the geometries in $G1$ and $G2$ to the same location and orientation as shown in Figure 3.8(c).

After obtaining the N groups of geometries in the post-OPC data, $G_1, G_2, ..., G_N$, corresponding to the N instances of the same cell in the pre-OPC data, $C_1, C_2, ..., C_N$, we search for a group of repeating geometries between $G_1, G_2, ..., G_N$. This problem of finding a repeating group of geometries between $G_1, G_2, ..., G_N$ can be solved with the InterSCD algorithm described in Section 3.1. However, the hierarchical clustering step can be omitted since $G_1, G_2, ..., G_N$ are known to share common geometries as they correspond to the same cell instance in the pre-OPC layout. Additionally, since the transformations that were applied to C to create $C_1, C_2, ..., C_N$ in the pre-OPC data are known, corresponding inverse transformations can be applied to $G_1, G_2, ..., G_N$ so that the geometries in $G_1, G_2, ..., G_N$ are mapped to the same location and orientation. In doing so, the exhaustive search per-



(a) (b) (c)

Figure 3.9. *(a) Example of a cell P; (b) 4 placements of P with 2 of them overlapping in pre-OPC layout; (c) the corresponding post-OPC layout; the geometries corresponding to the overlap region in the pre-OPC layout are highlighted in gray.*

formed by the InterSCD algorithm to find the largest group of repeating geometries between two cells can be omitted. Rather, a simple "AND" operation is required to find the largest group of repeating geometries between two cells.

Special attention must be paid to handle overlaps between placements. If two placements overlap in the pre-OPC layout, then their bounding boxes must be restricted to the portion which do not overlap rather than the full bounding box of the cell that they reference. This is because a Boolean "OR" operation is typically performed by the OPC software on the overlapping regions, and therefore, the geometries in these regions are not likely be part of a repeating group of geometries. Figure 3.9(a) shows an example of a cell P, Figure 3.9(b) shows an example of four placements of cell P with two of the them overlapping in the pre-OPC layout, and Figure 3.9(c) shows the corresponding post-OPC layout. As seen in Figure 3.9(c), the geometries in the overlapping region, highlighted in gray, have been "ORed", and are therefore completely different from the other geometries in the post-OPC layout corresponding to other instances of cell P.

A top down, bottom up approach is used to handle multiple levels of hierarchy where children cells may contain other children cells. This is needed in order to find repeating groups of geometries corresponding to children cells within other children cells. Figure 3.10

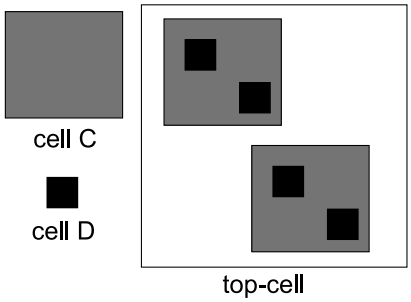


Figure 3.10. *An example of a layout with multiple levels of hierarchy; cell C is a child cell of top-cell and cell D is a child cell of cell C.*

shows an example of a layout with multiple levels of hierarchies where cell C is a child cell of the top-cell, and cell D is a child cell of cell C. By merely intersecting the bounding box of each instance of cell C in the post-OPC layout, the repeating geometries corresponding to cell D would go undetected. To address this, we propose the following: Begin with the largest cell, C, and gather all of the geometries in the post-OPC layout that correspond to each instance of C in the pre-OPC layout. Then the inverse transformation is applied to each group of geometries to obtain N groups of geometries in the same orientation. If C contains children cells, then we gather all of the geometries in the post-OPC corresponding to instances of each child cell in the pre-OPC layout. This is applied recursively until the there are no more children cells. Once the geometries corresponding to the cell at the lowest level of the hierarchy have been reconstructed, the cell on the next higher level of the hierarchy is reconstructed and so on. This process continues until the cell at the top level of the hierarchy is reconstructed.

So far, we have described ways of exploiting the pre-OPC hierarchy information in order to find repeating groups of geometries corresponding to cell instances in the pre-OPC layout. However, there may also exist repeating groups of geometries in the post-OPC layout that do not belong to some cell in the pre-OPC layout. Therefore, to ensure highest compression ratios, we apply the IntraSCD algorithm to find any remaining repeating groups of polygons in the post-OPC layout.

Chapter 4

Results

We have applied the above InterSCD and IntraSCD algorithms to actual industrial post-OPC layouts. The first data set consists of the Poly and Active layers for a 3.5mm × 3.5mm chip with 180nm feature size. For this set, OPC has been carried out by the layout owner with industry standard OPC software. The second data set consists of the Poly, Metal 1, and Metal 2 layers from 8mm × 8mm and 4.3mm × 4.3mm chips with 130nm feature size. The third data set consists of the Poly, and Active layers from 1.4mm × 1.4mm and 1.8mm × 1.8mm chips with 90nm feature size. We run OPC software from a major vendor on the second and third data sets with standard OPC recipes. The original post-OPC data and the compressed post-OPC data are encoded in the OASIS format.

4.1 InterSCD Results

For the first data set, we have found the InterSCD algorithm to work well, and the IntraSCD algorithm not to result in noticeable gain. We believe this is due to the way the data is processed by the OPC software. Furthermore, we notice that many of the post-OPC cells from the first layout data set are much smaller than those from the second and third data sets. Therefore, IntraSCD, which detects similar groups of polygons within a cell, can not result in significant gain on the first layout data set containing small cells. Table 4.1

	Post-OPC Size	InterSCD Size	CR
Poly (L1)	6,391,097	2,793,277	2.2880
Active (L1)	3,496,377	1,777,757	1.9667

Table 4.1. *Compression results with InterSCD algorithm. File sizes are in bytes.*

	Post-OPC Size	IntraSCD Size	IntraSCD+Ext Size	CR IntraSCD	CR IntraSCD+Ext	Ratio of IntraSCD to IntraSCD+Ext
Poly (L2a)	2,413,460	977,294	886,445	2.470	2.723	1.102
Poly (L2b)	1,036,664	576,491	554,949	1.798	1.894	1.053
Poly (L3a)	9,189,288	4,905,897	4,661,388	1.873	1.971	1.052
Poly (L3b)	34,515,762	18,960,928	17,924,204	1.820	1.926	1.058
Metal 1 (L2a)	2,490,423	1,791,495	1,731,540	1.390	1.438	1.035
Metal 1 (L2b)	1,194,192	1,060,746	1,034,056	1.126	1.155	1.026
Metal 2 (L2a)	1,444,367	1,143,360	1,136,757	1.263	1.271	1.006
Metal 2 (L2b)	947,981	775,561	768,770	1.222	1.233	1.009
Active (L3a)	9,666,584	6,899,025	6,514,057	1.401	1.484	1.059
Active (L3b)	35,945,586	23,209,262	22,118,134	1.549	1.625	1.049

Table 4.2. *Compression result with IntraSCD and IntraSCD+Ext algorithm. The file sizes are in bytes.*

shows the inter-cell sub-cell compressed file sizes in bytes encoded in OASIS format for post-OPC data set 1. As shown, the average compression ratio is around 2 for both layers.

4.2 IntraSCD Results

For the second and third data sets, the IntraSCD algorithm works well, while the InterSCD algorithm results in little gain. The fifth column of Table 4.2 shows the results of applying the IntraSCD algorithm on the second and third layout data sets. The compression ratios range from 1.80 to 2.46 for the Poly layer, and 1.40 to 1.55 for the Active layer. However, the compression ratios for the Metal layers are rather low i.e. in the range of 1.12 to 1.39. This can be explained by noting that the Metal layers contain many polygons with only a few instances.

Comparing the fifth and sixth columns of Table 4.2, we see that for all the layouts IntraSCD+Ext achieves higher compression ratio than translation only IntraSCD. From the 7^{th} column of Table 4.2, the compressed Poly and Active layouts using the IntraSCD algorithm are 5 to 10 percent larger than the ones with IntraSCD+Ext. The corresponding

gains for Metal 1 and Metal 2 are on average 3 and 1 percent respectively. The rotation and reflection gains for Metal 2 are smaller than those of Poly and Active because Metal 2 is created using routing software, and as such, has fewer geometries that are invariant under rotation and reflection; on the other hand, Poly and Active layers are typically created by placing reflected, rotated versions of standard cells.

4.3 IntraSCD Applied on Flattened Layout

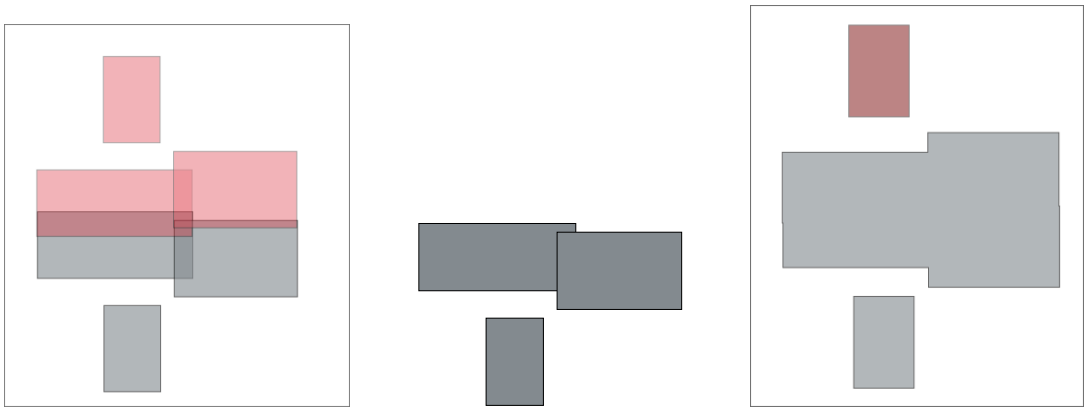
It is also possible to run IntraSCD algorithm to reconstruct the hierarchy for flattened layouts. We carry out this process for pre-OPC data since we can compare the reconstructed hierarchy with the original hierarchy to determine the performance of the IntraSCD algorithm.

There are two ways to flatten a layout; one is to push every geometry from the hierarchical layout to the top level of the hierarchy which we call flattened layout (FL). Another is to push the geometry to the top level and perform a Boolean "OR" operation to remove any overlaps between geometries; we refer to this as flattened layout with "OR" (FLWOR). For the purpose of this discussion, we define FL data as not having gone through a "OR" operation. Industry standard CAD tools offer both alternatives as a way to flatten layout data. In general, we would expect the FLWOR not only to result in smaller file size than FL, but also to have fewer repeating geometries than FL as the "OR" operation removes some of the repetitions in the flattened layout.

Table 4.3 shows file sizes of the pre-OPC hierarchical layout, FLWOR, and the IntraSCD compressed FLWOR in bytes. As shown in the Table, IntraSCD manages to do a reasonable job of reconstructing the hierarchy by significantly reducing the size of the FLWOR even though the "OR" operation destroys repetitions. However, as it is to be expected, IntraSCD cannot result in file sizes that are as small as the original hierarchical ones. Specifically, the compressed file sizes using IntraSCD are 1.31 and 2.11 times larger than the size of the pre-OPC hierarchical layout for the Poly and Active layers of layout 3b respectively, and

	Pre-OPC Hier. Size	FLWOR Size	Ratio of FLWOR to Hier.	IntraSCD Size	Ratio of FLWOR to IntraSCD	Ratio of IntraSCD to Hier.
Poly (L3a)	244,479	1,745,624	7.140	542,736	3.216	2.220
Poly (L3b)	897,578	4,085,736	4.552	1,176,777	3.472	1.311
Active (L3a)	267,104	1,345,587	5.038	859,160	1.565	3.217
Active (L3b)	1,255,861	3,702,918	2.949	2,654,989	1.395	2.114

Table 4.3. *Comparison of pre-OPC hierarchical layouts with compressed FLWOR layouts using IntraSCD algorithm. File sizes are in bytes.*



(a) (b) (c)

Figure 4.1. *(a) 6 geometries with 4 overlapping ones; (b) group of 3 geometries that repeat twice in (a); (c)result of performing Boolean "OR" on (a).*

2.22 and 3.22 times larger for the Poly and Active layers of layout 3a respectively. The IntraSCD algorithm performs worse on the Active than Poly layers of both layouts 3a, and 3b. The main reason is that the Active layer consists of many overlapping geometries, and once a Boolean "OR" operation is performed, many groups of repeating geometries are removed.

To show the effects of the "OR" operation on repeating geometries, consider Figure 4.1. Figure 4.1(a) shows an example of two overlapping groups of geometries without the "OR" operation, and Figure 4.1(b) shows the group of repeating geometries consisting of 3 polygons within Figure 4.1(a). However, performing a Boolean "OR" operation on Figure 4.1(a) as shown in Figure 4.1(c), does not results in a group of 3 geometries that repeats twice; rather it results in one geometry that repeats only twice.

	Pre-OPC Hier. Size	FL Size	Ratio of FL to Hier.	IntraSCD Size	Ratio of FL to IntraSCD	Ratio of IntraSCD to Hier.	Ratio of IntraSCD FLWOR to IntraSCD FL
Poly (L3a)	244,479	2,449,539	10.019	525,530	4.651	2.150	1.033
Poly (L3b)	897,578	4,085,736	4.552	1,176,777	3.472	1.311	1.000
Active (L3a)	267,104	2,401,676	8.992	463,675	5.181	1.736	1.852
Active (L3b)	1,255,861	4,131,499	3.290	1,862,786	2.217	1.483	1.425

Table 4.4. *Comparison of pre-OPC hierarchical layouts with compressed flattened without "OR" layouts using IntraSCD algorithm. File sizes are in bytes.*

Table 4.4 shows the file sizes of pre-OPC hierarchical layouts, FL, and the compressed FL layouts in bytes. Comparing the fifth columns of Tables 4.3 and 4.4, we see that applying IntraSCD to FL results in smaller file sizes than to FLWOR. This implies that compressed IntraSCD file sizes are closer to the original hierarchical file sizes if IntraSCD is applied to FL, rather than to FLWOR. This is also shown in the last column of Table 4.4 which shows that, the ratio of IntraSCD FLWOR to hierarchical is larger than the IntraSCD FL to hierarchical. The gain is more pronounced for the Active than the Poly layers as Active has many more overlapping geometries. The small 3.3% improvement in compression ratio of the Poly layer of layout 3a and no improvement of the Poly layer of layout 3b can be attributed to the fact that layout 3a contains memory cells with overlapping Poly geometries, while the Poly layer of layout 3b has no overlapping geometries. As explained earlier, performing a Boolean "OR" removes some of the repeating groups of geometries that the IntraSCD algorithm exploits to reduce the file size.

4.4 IntraSCD+Ext on Flattened Layout

We apply the IntraSCD+Ext on the pre-OPC FL as shown in Table 4.5. By considering groups of geometries that are the same under translation, rotation, and reflection, the file size can be further reduced as compared to translation only. As seen in column 5 of Table 4.5, for FL data, IntraSCD+Ext decreases the file size by 6 to 37 percent as compared to IntraSCD. In addition, the gain over IntraSCD is higher for layout 3a than layout 3b. This is in part due to the fact that layout 3a contains memory, and memory blocks are assembled by placing translated, rotated and reflected memory bit-cells together. As such, there are

	Ratio of FL to Hier.	Ratio of IntraSCD+Ext to FL	Ratio of IntraSCD+Ext to Hier	Ratio of IntraSCD to IntraSCD+Ext
POLY (L3a)	10.019	0.156	1.565	1.374
POLY (L3b)	4.552	0.259	1.179	1.112
ACTIVE (L3a)	8.992	0.165	1.486	1.168
ACTIVE (L3b)	3.290	0.423	1.392	1.065

Table 4.5. *Comparison of IntraSCD+Ext to IntraSCD on the pre-OPC FL data.*

Layout	IntraSCD CR	EHier + IntraSCD CR	EHier+IntraSCD Run Time in Minutes	Speed Increase
Poly (L3a)	1.8731	1.7860	7.5	1.35
Poly (L3b)	1.8204	1.7787	8.5	6.08
Active (L3a)	1.4012	1.3745	8.5	5.89
Active (L3b)	1.5488	1.4674	12	4.89

Table 4.6. *Compression ratio and run time of IntraSCD+EHier compared to IntraSCD.*

many more groups of geometries that are invariant under rotation, and reflection in layout 3a than in layout 3b. Overall, both IntraSCD and IntraSCD+Ext perform better on layout 3b than 3a. Layout 3a contains memory blocks that are very compact and hierarchical, and our algorithm is unable to rediscover all the repetitions that are present in the original, compact, hierarchical layout data.

4.5 Exploiting pre-OPC Hierarchy Results

Table 4.6 compares the compression ratio and run time of IntraSCD and IntraSCD+EHier described in Section 3.4. As seen in the fifth column of Table 4.6, exploiting pre-OPC hierarchy can reduce the run time by a factor of 1.35 to 6.08 with small or no loss in compression efficiency as compared to the IntraSCD algorithm. The longest runtime for IntraSCD among all the layouts is 58 minutes for the post-OPC Active layer of layout 3b. Exploiting the pre-OPC hierarchy reduces the runtime by a factor of 5. Comparing the second and third columns of Table 4.6, in the worst case we observe less than a 5.5% decrease in compression ratio.

Chapter 5

Conclusion and Future Works

We have presented a class of lossless compression algorithms for post OPC IC layout data. In addition to being lossless, the compressed layout data remains fully format compliant, which means that the compressed data can be read by industry standard CAD viewing/editing tools without the need for a decoder. Our proposed algorithms find redundancies in terms of repeating geometries within a cell and between cells. We have shown that our approach achieves reasonable compression ratio on the Poly and Active layers. Furthermore, we have developed a method to exploit the pre-OPC hierarchy information in order to speed up the process of finding common groups of geometries within a cell. In doing so, we have demonstrated an average of 5 times increase in speed while suffering a small or no loss in compression efficiency.

In the future, we plan to run our algorithms on more extensive sets of data. We also need to gain a better understanding of why the performance on the Metal layers is not as high as the Poly and Active layers, and determine techniques to improve the compression ratio for the Metal layers. Future work also involves extension of the InterSCD algorithm to rotations and reflections.

References

[1] W. Grobman, R. Boone, C. Philbin, and B. Jarvis, "Reticle enhancement technology trends: resource and manufacturability implications for the implementation of physical designs," in *ISPD '01: Proceedings of the 2001 international symposium on Physical design*. New York, NY, USA: ACM Press, 2001, pp. 45–51.

[2] "International technology roadmap for semiconductors 2005 edition, lithography," available at `http://www.itrs.net/Common/2005ITRS/Litho2005.pdf`.

[3] SEMI, *OASIS – Open Artwork System Interchange Standard*, available as http://webstore.ansi.org/ansidocstore/product.asp?sku=SEMI+P39-1105.

[4] V. Dai and A. Zakhor, "Advanced low-complexity compression for maskless lithography data," in *Proceedings of SPIE – Volume 5374, Emerging Lithographic Technologies VIII*, R. S. Mackay, Ed., May 2004, pp. 610–618.

[5] H. Liu, V. Dai, A. Zakhor, and B. Nikolic, "Reduced complexity compression algorithms for direct-write maskless lithography systems," in *Proceedings of SPIE – Volume 6151, Emerging Lithographic Technologies X*, M. J. Lercel, Ed., 2006, pp. 632–645.

[6] Y. Chen, A. B. Kahng, G. Robins, A. Zelikovsky, and Y. Zheng, "Compressible area fill synthesis." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 8, pp. 1169–1187, 2005.

[7] R. Veltman and I. Ashida, "Geometrical library recognition for mask data compression," in *Proceedings of SPIE – Volume 2793, Photomask and X-Ray Mask Technology III*, Y. Tarui, Ed., July 1996, pp. 418–426.

[8] T. Akutsu and M. M. Halldorsson, "On the approximation of largest common subtrees and largest common point sets," in *ISAAC '94: Proceedings of the 5th International Symposium on Algorithms and Computation*. London, UK: Springer-Verlag, 1994, pp. 405–413.

Chapter 6

Appendix

In this appendix, we show that the IntraSCD problem is NP-hard by reducing the known NP hard 1-dimensional largest common point set problem (LCP) to the IntraSCD problem.

Given a collection of point sets $SS = \{S_1, S_2, ..., S_n\}$, with each point set containing points on the real number line, construct a cell , C^{ss}, containing $|S_1| + |S_2| ... + |S_n|$ geometries.

Define:

$$distance(S_i) = max(S_i) - min(S_i)\ S_{dmax} = \arg\max_{S_j} \{distance(S_j)\}.$$
$$d_{min} = \min_{S_j} \{distance(S_j)\}$$

For each point, P_{ji}, in the point set S_i, define a corresponding square in C^{ss}, with bottom left coordinate of the square at $(L(p_{ji}), 0)$, where $L(\cdot)$ is a function that maps P_{ji} to some value in $\Re$, and set $dist$ to d_{min}. To make the notation simpler, each square in C^{ss} is considered as a 1-dimensional point with value $L(p_{ji})$, and C^{ss} as a 1-dimensional point set.

A new point set C^{ss} is created by initially setting it to S_{dmax}. Then each point set from SS is added one at a time to C^{ss}. Let

$$D = distance(C^{ss})\ P = max(C^{ss})\ L(P_{ji}) = P_{ji} + P + D + 1 + distance(S_i).$$

For every point $P_{ji} \in S_i$, a new point, $L(P_{ji})$, is created in C^{ss}. Once all of the points from S_i have been added to C^{ss}, D and P are recomputed, and another point set S_{i+1} is added to C^{ss}. This is repeated until all of the point sets have been added. At the end of the process, there is a new point set $C^{ss} = \left\{P^{ss}_{1,1}, P^{ss}_{2,1}, ..., P^{ss}_{m,2}, P^{ss}_{m+1,2}, ..., P^{ss}_{n,n}\right\}$, where $P^{ss}_{i,j}$ is the i^{th} point in C^{ss} that is mapped from the point set S_j. Clearly this construction process can be done in polynomial time. The construction of C^{ss} guarantees any solution to the intraSCD problem can not come from points that are mapped from two different point sets S_i, S_j, i.e, $T(P^{ss}_{i,m}), T(P^{ss}_{j,n}) \notin C^{ss}_{sub}$, because the distance between two points $P^{ss}_{i,m}, P^{ss}_{j,n}$ is greater than $dist$.

If SS_{sub} is a solution to the LCP problem and SS_{sub} contains K points, then it is also a solution to the intraSCD problem. By construction, a point set that repeats n times in C_{ss} can not contain points mapped from two different point sets S_i and S_j, and therefore can not have point set with more than K points that repeats n times. If there exists such a set, then SS_{sub} would not have been a solution to the LCP problem. Suppose C^{ss}_{sub} is a solution to the intraSCD problem with C^{ss}_{sub} having K points. Since C^{ss}_{sub} occurs n times in C^{ss}, and C^{ss}_{sub} can not contain points mapped from two different point sets in SS_{sub}, there must exist some mapping that takes C^{ss}_{sub} to each of point sets, S_i, in SS.

For example, suppose we have 3 point sets,

$$S_1 = \{2, 6, 9, 12\}\ S_2 = \{3, 5, 7\}\ S_3 = \{12, 13, 16\}.$$

Then $C^{ss} = \{2_1, 6_1, 9_1, 12_1, 30_2, 32_2, 34_2, 75_3, 76_3, 79_3\}$, where the subscript denotes the point set from the point came, and $dist = 4$. Notice any subsets of C^{ss} which are mapped from two different point sets have a distance greater than 4. In this example, $\{2, 6\}$, is a solution to the LCP and the intraSCD problem.