Hindawi
Journal of Electrical and Computer Engineering
Volume 2019, Article ID 9430593, 12 pages
https://doi.org/10.1155/2019/9430593

Research Article

CHIP: Clustering Hotspots in Layout Using Integer Programming

Rohit Reddy Takkala and Chris Chu

Dept. of Electrical and Computer Engineering, Iowa State University, Ames 50011, USA

Correspondence should be addressed to Rohit Reddy Takkala; rohitreddytakkala@gmail.com

Received 24 July 2018; Revised 30 October 2018; Accepted 27 November 2018; Published 17 January 2019

Academic Editor: Maurizio Martina

Clustering algorithms have been explored in recent years to solve hotspot clustering problems in integrated circuit design. With various applications in design for manufacturability flow such as hotspot library generation, systematic yield optimization, and design space exploration, generating good quality clusters along with their representative clips is of utmost importance. With several generic clustering algorithms at our disposal, hotspots can be clustered based on the distance metric defined while satisfying some tolerance conditions. However, the clusters generated from generic clustering algorithms need not achieve optimal results. In this paper, we introduce two optimal integer linear programming formulations based on triangle inequality to solve the problem of minimizing cluster count while satisfying given constraints. Apart from minimizing cluster count, we generate representative clips that best represent the clusters formed. We achieve a better cluster count for both formulations in most test cases as compared to the results published in the literature in the ICCAD 2016 contest benchmarks as well as the reference results reported in the ICCAD 2016 contest website.

1. Overview

As the feature size decreases rapidly, the problem of manufacturability in integrated circuits increases due to limitations in lithographic wavelength used during the fabrication stage. These problems identified as hotspots are a set of problematic patterns in the layout that have printing issues. These are detected either using traditional lithographic simulations or machine learning-based detection methods that have been proposed in recent years. When such defects are found, finding patterns of similar kind is of high interest. It becomes useful to cluster these clips of interest into groups and process them together. This is called layout pattern classification [1] or hotspot clustering. Layout pattern classification has been utilized in recent years in design for manufacturability flow for various applications. Few examples of such applications are hotspot library generation [2], hierarchical data storage [3], and systematic yield optimization. With several applications in the DFM stage, finding good quality clusters is important.

Few works such as [4] use pattern classification within their tool flow. They use a modified version of incremental clustering where they update the representative of the clusters formed. Wu et al. [5] have worked on a modified problem statement, where they consider dummy fills during hotspot classification, can therefore accurately identify the process hotspots in the layout with dummification in EUVL. An interesting approach is adopted [6] where they shift the clips to expand the solution space while satisfying given constraints, thereby reducing the cluster count. The minimal cluster count is achieved for the ICCAD 2016 benchmark suite in their approach. Different distance metric instead in of the XOR logic is explored in [7, 8] to encapsulate rotations/mirroring and other topological features. However, using these metrics is a trade-off between computational cost and quality of the clusters generated. There are several other works such as [9–12] which focus on hotspot detection frameworks, whereas hotspot clustering has been rarely explored but plays an important role in various applications in the design for manufacturability flow.

In previous work [2, 13, 14], a few generic clustering algorithms such as k-means clustering [14], hierarchical and incremental clustering [2], and Markov clustering [13, 15] were explored to solve this problem. In k-means clustering, the value of k needs to be provided by the user, but the user may not know the cluster count a priori. Therefore, it does not solve the purpose of finding good quality clusters automatically. In the hierarchical clustering algorithm, starting from each data point as a cluster, the data are hierarchically grouped together based on different types of linkages. Since hierarchical clustering finds groups of data in a hierarchical manner, it is again user dependent to get the clusters. In incremental clustering, in the order of processing data, either new clusters are created or existing clusters are grown incrementally. This algorithm depends on the order of processing data and therefore does not produce good quality solutions. Markov clustering [15] is known to find good quality clusters in a short time, but the clustering depends on fine tuning several parameters in the algorithm. There are several other clustering algorithms in the literature to cluster any kind of data; however, the problem formulations of those algorithms are different from that of the hotspot clustering problem. Therefore, a postprocessing step is required while using those algorithms to regroup clusters in order to satisfy the given constraints.

In this paper, we discuss our tool called CHIP which solves the given hotspot clustering problem optimally. We formulate two integer linear programs to solve for the optimal number of clusters, i.e., the objective of both formulations is to minimize cluster count. With some tolerance given by area constraint or edge constraint, our tool classifies given clips into clusters without assuming that the representative clips must be from the given data set. Since the representative clip is not required to be one of the given clips, we generate the representative clip based on the cluster data and the tolerance provided. This framework can achieve optimal cluster count while satisfying the constraints as per the results from ICCAD 2016 Contest Problem C-Pattern Classification for Integrated Circuit Design Space Analysis [1].

The paper is further organized as follows: In Section 2, we describe the problem, define the terminology, and elaborate the two modes in clustering. In Section 3, we discuss the overview of our tool flow. In Section 4, we define our integer linear programming formulations which exactly represent the problem statement, and in Section 5, the framework to generate representative clips is elaborated. Furthermore, in Section 6, we report the results of the formulation and compare with existing algorithms. We conclude the work in Section 7.

2. Problem Description

2.1. Overview. This problem is taken from the ICCAD 2016 Contest-Problem C. Given a GDS file with markers and clip size and the constraints as inputs, the hotspot classification tool has to cluster the clips formed around the markers and output the corresponding cluster identities and a set of representative clips which represent the clusters. There are two types of constraints given to the tool, i.e., area constraint (a) and edge constraint (e). Based on the type of constraint, the tool has to perform clustering in the respective mode. The tool takes either area constraint or edge constraint but not both as the input.

Figure 1 depicts a sample layout where the polygons (in yellow) are the interconnects of a circuit indicated in a layer. On this layer, there are several markers placed at various locations throughout the layout. Given these input data, clips centered at the markers should be extracted according to the given dimensions. Note that the center of the clip can be anywhere inside the marker.

We define the terminology used in the problem description as follows:

Definition 1. Marker: a marker is a polygon which locates the presence of a hotspot in the layout. These markers are placed on a different layer other than the design layer. For practical purposes, these markers are picked to be small—about the height & width of minimum width allowed in the layout.

Definition 2. Clip: a clip is defined as a set of polygons extracted from the layout, based on the position of the marker. These set of polygons are extracted by a bounding polygon (width w and height h) with its center anywhere inside the marker. A sample clip is shown in Figure 2. For practical purposes, the center of the clip can be assumed to be the center of the marker.

Definition 3. Cluster: a cluster is a set of clips which are grouped together based on the similarity metric defined.

Definition 4. Representative clip: a representative clip is defined for each cluster which is similar to all its clips, where the degree of similarity is constrained by a tolerance parameter given as input. For practical purposes, representative clips can be chosen from existing clips for each cluster. But it need not necessarily exist in the layout.

2.1.1. Additional Specifications. Mirroring of clips is allowed, i.e., 180° rotation along the axes passing through the clip's center. Therefore, there are 4 possible combinations for each clip. This is depicted in Figure 3. Also, since the clip's center need not be at the center of the marker, clip shifting can be performed to generate a set of clips for one marker. A sample set of possible clips are depicted in Figure 4 by shifting a clip's center. In this work, for simplicity, we consider the center of the clip to be at the center of the marker, i.e., we only consider the clip in Figure 4(a).

In general, clustering algorithms require pairwise similarity relation of data points in order to group the data into clusters. Pairwise distances of data points are one of the ways to establish the similarity measure, i.e., the greater the distance, the greater the dissimilarity. There are various types of distances used for different applications such as L1 norm for images, L2 norm for any d-dimensional set of points, and Hamming distance for distance between two strings.

In hotspot classification, each clip (x_i) is represented as a $w \times h$ dimensional data point, i.e., $x_i \in R^d$, where $d = w \times h$. For any two clips x_1 and x_2, $\mathrm{XOR}(x_1, x_2)$ produces a clip

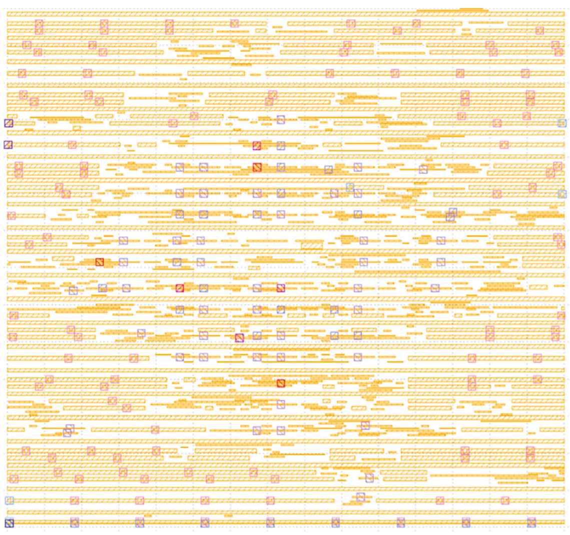
Figure 1: Sample layout.

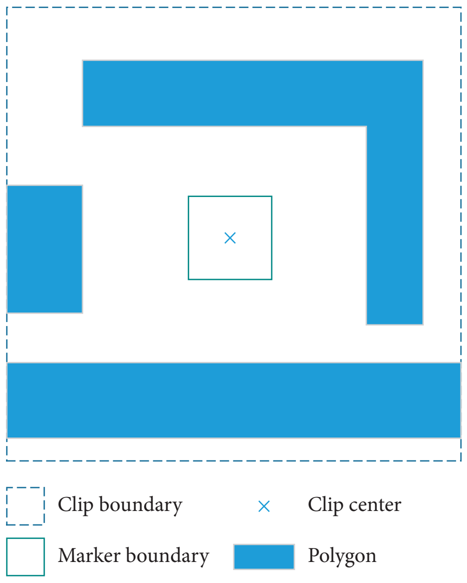

Figure 2: Sample clip.

which depicts the dissimilarity between the given two clips. Furthermore, based on the two constraints—area-constrained clustering and edge-constrained clustering—the distance metric is defined for each mode by imposing the respective constraints on the resultant clip. In the following subsections, the two constraint-based clustering modes are explained in detail.

2.2. Constrained Clustering

2.2.1. Area-Constrained Clustering. In area-constrained clustering (ACC), the distance metric is computed based on the area of the resultant clip from exclusive OR operation applied to two clips x_1 and x_2:

$$D(x_1, x_2) = \text{area}(\text{XOR}(x_1, x_2)). \tag{1}$$

For example, in Figure 5, two clips are overlapped against each other whose dissimilarity is depicted through arrows. In Figure 6, the resultant XOR of the two clips is shown. The distance of the clips is therefore the area of the polygons (rectangles in this case) in Figure 6, i.e., $D(x_1, x_2) = \text{area}(\text{XOR}(x_1, x_2))$.

Given this distance function between a pair of clips, ACC constrains the distance between any clip S in a cluster and its representative clip R as follows:

$$\frac{D(R, S)}{(w \times h)} \leq (1 - a), \tag{2}$$

where $w \times h$ is the area of the clip and $0 \leq a \leq 1$. Here, a is the parameter given to the tool which constraints the distance between the clips.

If $a = 1$, the tool has to perform exact clip matching. For practical purposes, a is close to 1. This constraint need not enforce two clips to be clustered together if they satisfy it. However, if two clips do not satisfy the constraint, then they cannot be clustered together.

2.2.2. Edge-Constrained Clustering. In edge-constrained clustering (ECC), the distance between two clips x_1 and x_2 is given by the maximum shift along an edge either inward or outward in clip x_1 with respect to clip x_2; that is, if e_i is ith shift along one edge out of all possible edge shifts in clip x_1 with respect to the clip x_2, then $D(x_1, x_2) = \max(e_1, e_2, \ldots)$. An example pair of clips with edge shifts is shown in Figure 5.

For any clip S in a cluster and its representative clip R, then according to ECC, the following should be satisfied:

$$D(R, S) \leq e, \tag{3}$$

where e is given as a parameter. Here, e is a nonnegative real number. For practical purposes, e is close to 0. Similar to ACC, ECC does not enforce the clips to be clustered together if they satisfy the constraint. If the clips do not satisfy the constraint, they should not be clustered together.

3. Overview of the Tool

The proposed tool flow is discussed in this chapter. Figure 7(a) shows our proposed tool flow with the steps. In the layout data processing step, we convert all the polygons into rectangles for easier data processing. We then handle the layout data (in rectangles) using a grid structure in order to speed up the process of clip extraction. In distance computation step, we reorient all clips in a canonical way to consider mirroring of the clips. Exact pattern matching is performed to reduce data size, and therefore, redundant computations are avoided in the subsequent steps.

Then, we compute the pairwise distances between these reduced data according to the constraint type. Using this distance matrix (D) and given tolerance (D_c, which is determined by either a or e depending on the constraint type), in the clustering step, an optimizer is called to solve the optimization problem based on the formulations discussed in Section 4 and an optimal solution is arrived along with the cluster indices. Furthermore, since we assume each cluster need not have its representative amongst given data, we use ILP formulation again to search feasible solution space to

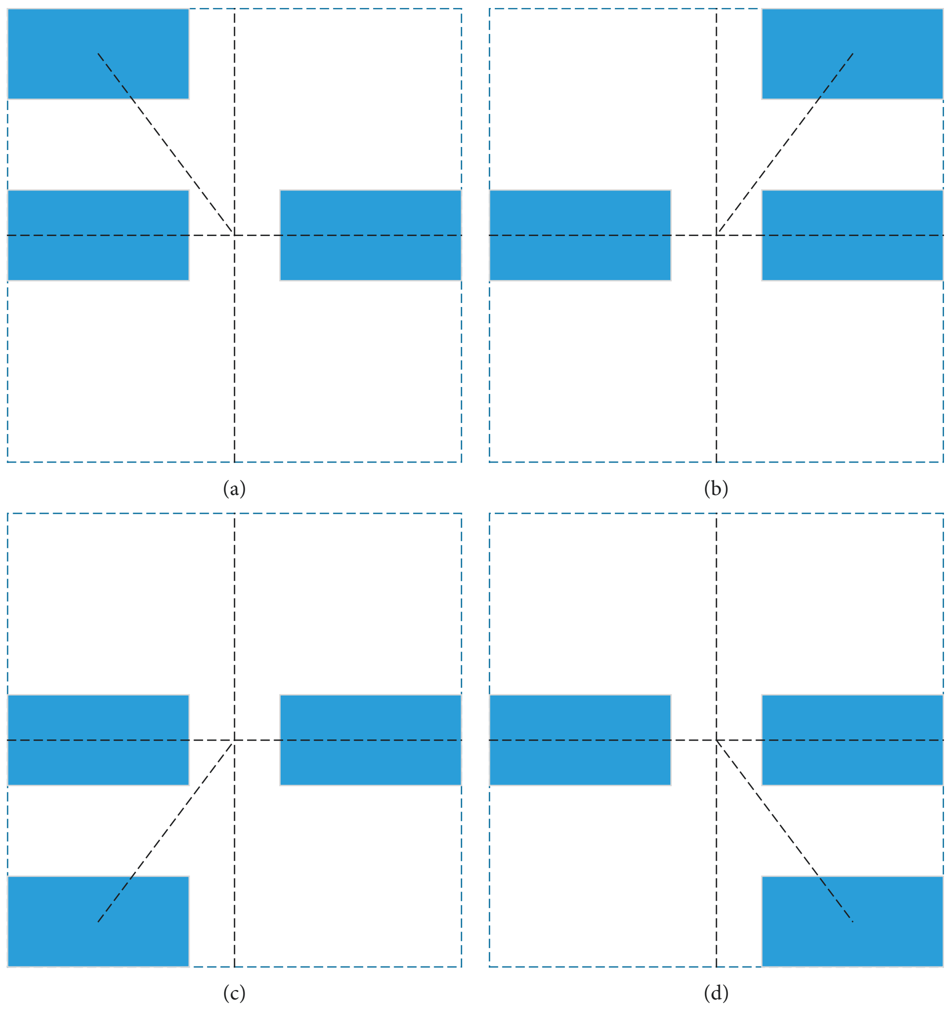

Figure 3: Four possible configurations of a clip.

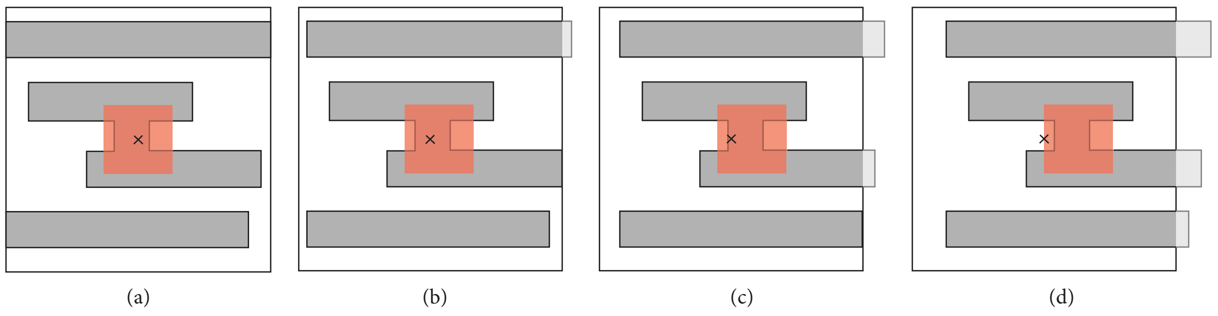

Figure 4: A sample set of configurations of a clip (with shifting) [6].

generate the representative clip. The following subsections and chapters discuss each step in our proposed flow in detail.

3.1. Layout Data Processing. In this step, firstly, the polygons are converted into rectangles using a standard algorithm. Note that this conversion need not be optimal in nature. Then, the entire layout is divided into a grid structure where each unit is of width w and height h as shown in Figure 7(b). With this grid structure, the rectangles overlapping each grid are stored in a data structure. While extracting the clip for a given marker, we use the information stored in the data structure to take relevant rectangles to form the clip. This process avoids scanning all rectangles and finding intersection between them and the clips of interest. This is illustrated in Figure 7(b). At most 4 grid structures and

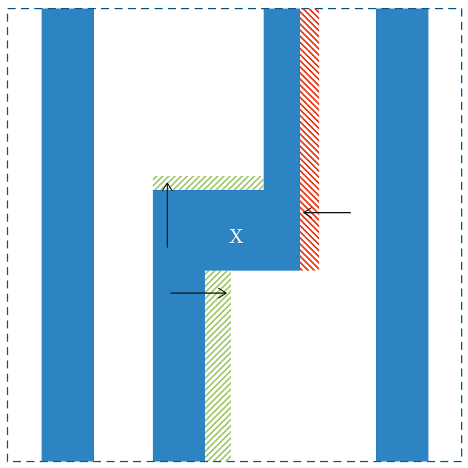

Figure 5: Two clips overlapped with each other [1].

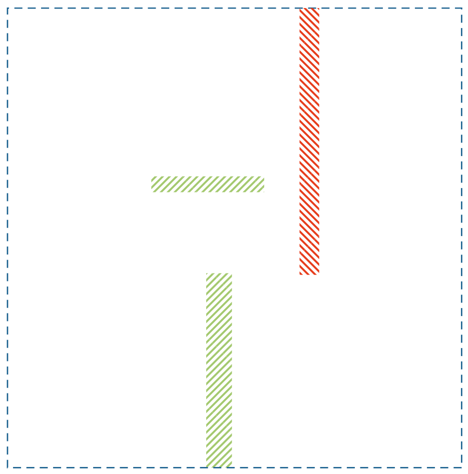

Figure 6: XOR of the two clips [1].

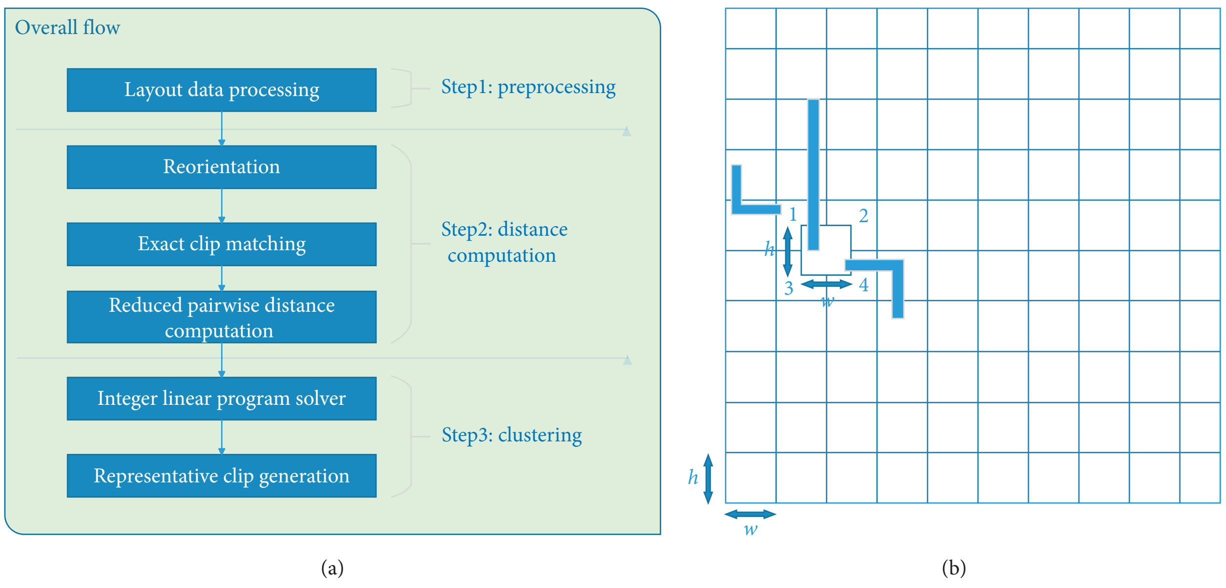
Overall flow
Layout data processing
Step1: preprocessing
Reorientation
Exact clip matching
Reduced pairwise distance computation
Step2: distance computation
Integer linear program solver
Representative clip generation
Step3: clustering
(a)
1
2
3
4
h
w
h
w
(b)

Figure 7: Overview of the tool and preprocessing step. (a) An overview of the tool. (b) Layout data processing.

correspondingly the rectangles present in them are scanned for any clip to be extracted.

3.2. Distance Computation

3.2.1. Reorientation. Since we consider reflections along x-axis or y-axis or both, in this step, before computing distances between the clips based on area or edge constraint, we perform reorientation of the clips in a canonical way. We compute the center of mass (COM) for a given clip and divide the clip into 4 quadrants. Here, center of mass metric is defined as follows.

Let a_i be a clip which is mapped to a R^2 space with $w \times h$ number of data points, with the range $-w/2$ to $w/2$ on the x-axis, $-h/2$ to $h/2$ on the y-axis, and the center of the clip at (0, 0). With this mapping, if there is a pixel at (x, y), then its value is 1, i.e., $a_i(x, y) = 1$ and 0 otherwise. Let (x_c, y_c) represent the center of mass of this notation. Therefore,

$$x_c = \frac{\sum_{x=-w/2}^{w/2}\sum_{y=-h/2}^{h/2} a_i(x, y) * x}{\sum_{x=-w/2}^{w/2}\sum_{y=-h/2}^{h/2} a_i(x, y)},$$
$$y_c = \frac{\sum_{x=-w/2}^{w/2}\sum_{y=-h/2}^{h/2} a_i(x, y) * y}{\sum_{x=-w/2}^{w/2}\sum_{y=-h/2}^{h/2} a_i(x, y)}. \tag{4}$$

Then, we orient all the clips such that every clip's COM is in a fixed quadrant, e.g., lower-left quadrant as shown in Figure 8. This preprocessing step enables us to find exact clip matching patterns. In Figures 8(a)–8(d), all possible reflections along the axes are indicated, and Figure 8(e) is the canonical representation of all orientations. Note that in case the center of mass is closer to the origin, then a higher order metric can be computed to shift the COM away from the origin.

3.2.2. Clip Matching. Once the clips are reoriented in a canonical way, clip matching step is performed in order to merge exact clips in the given data. In an IC with millions of gates, it is most likely to find identical patterns in the layout, and hence, this step would reduce the amount of data to be processed. Exact clip matching can be performed with pattern matching algorithms or by string comparison if each clip is encoded into a string as proposed in [4].

In this work, exact clip matching is performed in two levels. First, the given data are divided into different bins, where a bin contains all the clips of same area. Then, the clips in each of the bins are iterated through, with new clusters formed whenever there is a mismatch with the existing clusters in the bin; that is, incremental clustering is performed, where two or more clips are clustered together if the pairwise distance between them is zero.

To compute the distance between the two clips, each clip is divided into a nonuniform grid where the grid lines are along the boundaries of the polygons on the two clips. Therefore, each grid in the clip is either completely covered by a polygon or completely empty and, hence, can now be represented by a binary value. As a result, the distance of the two clips can be easily computed based on the binary values for each grid and its corresponding area, as shown in Figure 9.

3.2.3. Distance Computation. In this final step, pairwise distances are computed between the reduced set of clips. To compute the distance between the two clips, each clip is divided into nonuniform grids where the grid lines are along the boundaries of the polygons on the two clips as discussed in the Section 3.2.2. Therefore, the distance between a pair of clips can be easily computed based on the binary values for each grid and its corresponding area, as shown in Figure 9.

4. ILP Formulations

One of the objectives of the problem is to minimize the cluster count while satisfying the tolerance in terms of ACC/ECC. In the following formulations, we define the objective of the ILP as minimizing the number of clusters. Therefore, the optimizer solves for the optimal number of clusters for a given constraint. Also, we leverage the idea of triangle inequality, as defined in Proposition 1, in order to generate the minimal cluster count; that is, the representative clip need not be chosen from the given clips, and therefore, we explore the solution space without unnecessary restrictions while satisfying the given constraints. We formulate two integer linear programming approaches describing the given problem in different ways. Both these formulations are described in the following subsections.

4.1. CHIP Node. In this formulation, we describe the clustering problem using nodes as variables, where each node is assigned a cluster identity based on the distance metric and the constraints. We define C_i as the variable representing each data point i, and its value indicates the cluster index of that data point:

$$C_i = C_j \Longleftrightarrow i, j \in \text{same cluster},$$
$$C_i \neq C_j \Longleftrightarrow i, j \notin \text{same cluster}, \quad \forall i, j = 1, 2, \ldots n, \tag{5}$$

where n = number of data points.

Here, the variables C_i are upper bounded by another variable, K, representing the cluster count, i.e., $1 \leq C_i \leq K$, $\forall i = 1, 2, \ldots, n$ and $K \geq 1$. With this setup, the objective to minimize the cluster count is to minimize K in our formulation.

Let $D(i, j)$ be the distance between ith clip and jth clip and the constrained distance be D_c.

Proposition 1. *Triangle inequality for clustering: given a cluster of clips and the distance constraint D_c, if $D(i, j) \leq 2 \times D_c$, $\forall i, j \in$ same cluster, then $\exists\ r$ such that $D(i, r) \leq D_c \cdot \forall i$.*

ILP Formulation:

Objective: *minimize K.*

Constraints: $\forall i \neq j$

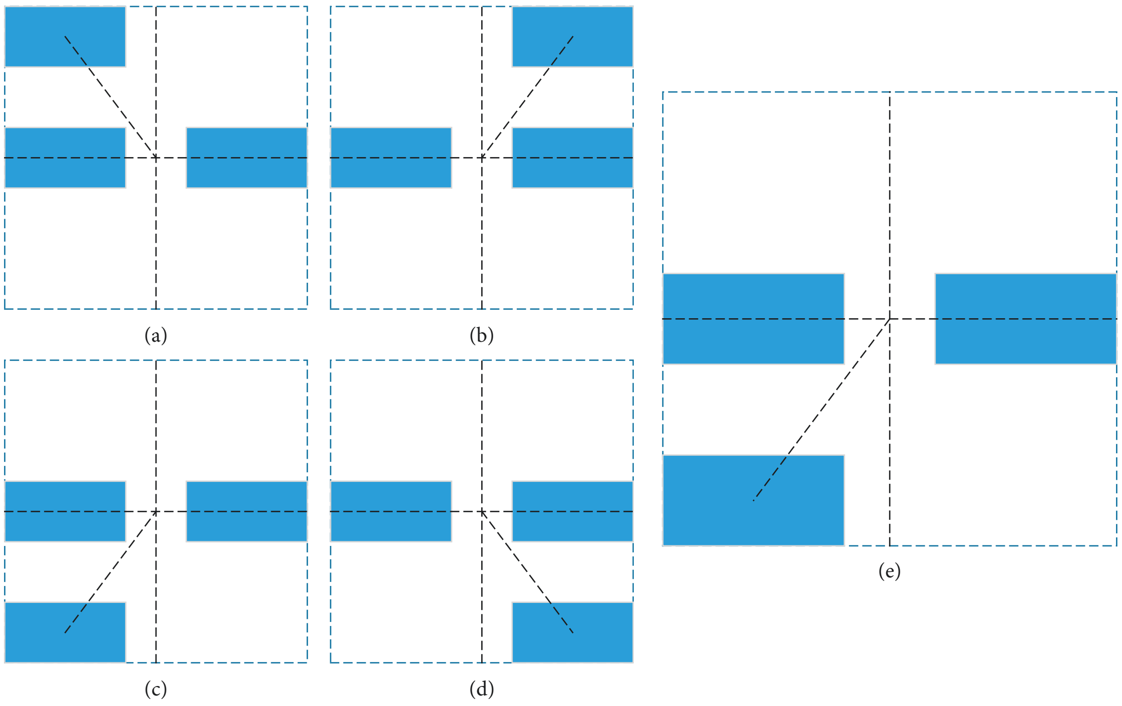

Figure 8: Reorientation of the clip.

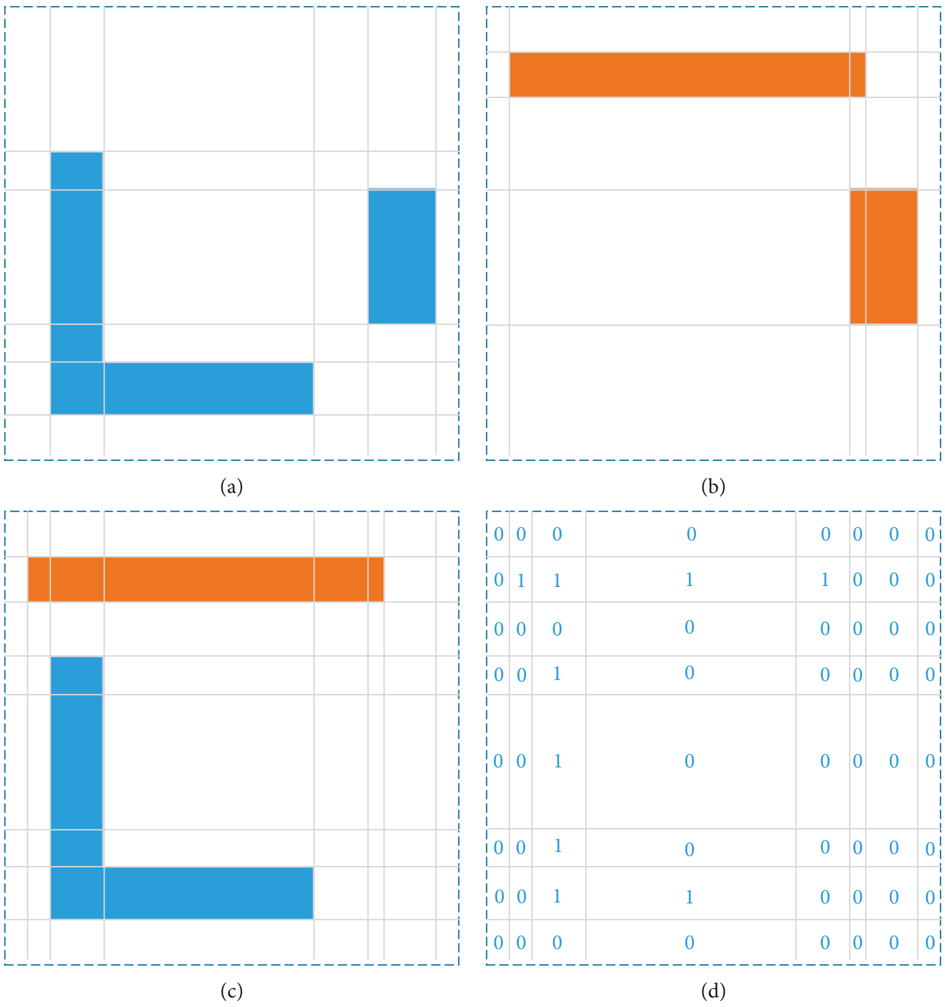

Figure 9: Grid data structure to compute distance.

$$C_i \geq C_j + \left[1 - \left(\frac{2D_c}{D(i,j)}\right)\right] - S_{ij} \times H,$$
$$C_i \leq C_j - \left[1 - \left(\frac{2D_c}{D(i,j)}\right)\right] + \left(1 - S_{ij}\right) \times H. \quad (6)$$

Here, H is a huge constant, C_i is integer $\forall i$ and S_{ij} is 0 or 1 $\forall i, j$.

Bounds: $1 \leq C_i \leq K \forall i$

The above two constraints enforce the condition that if the distance between two clips (i and j) $D(i,j) > 2D_c$, then the two clips (nodes) cannot be clustered together, i.e., $C_i \neq C_j$. However, the constraints can be ignored whenever the distance constraint is satisfied; that is, the clips can be either clustered together or not. This is elaborated in the following two cases:

Case 1. If $D(i,j) > 2D_c$:

Constraints:

$$C_i \geq C_j + \epsilon - S_{ij} \times H,$$
$$C_i \leq C_j - \epsilon + \left(1 - S_{ij}\right) \times H \Longrightarrow C_i \neq C_j. \quad (7)$$

Note: Here, ε is a small value.

Case 2. If $D(i,j) \leq 2D_c$:

Constraints:

$C_i \leq C_j$ or $C_i \geq C_j$ depending on the value of S_{ij}.

Note that a preprocessing step elaborated in Section 3.2.2 is applied to eliminate exactly the matched patterns. Hence, $D(i,j)$ will never be zero in this formulation.

4.1.1. Area-Constrained Clustering. In the case of area-constrained clustering, $D(i,j) = \text{area}(\text{XOR}(x_i, x_j))$ as defined in Section 2.2.1 and $D_c = w \times h \times (1 - a)$, where a is the area constraint ranging between 0 and 1. Notice that, for $a = 1$, $D_c = 0 \Longrightarrow C_i \neq C_j$, $\forall i, j$.

4.1.2. Edge-Constrained Clustering. In the case of edge-constrained clustering, $D(i,j) = \max(e_1, e_2, \ldots)$ as defined in Section 2.2.2 and $D_c = e$, where e is the given edge constraint (in nm).

4.2. CHIP Edge. In the 2nd formulation, we describe the clustering problem using edges, where two nodes connected by an edge are clustered together. We define that the objective of the ILP is to minimize the number of clusters. Similar to the previous formulation, we leverage the idea of triangle inequality in order to generate minimal cluster count; that is, the representative clip need not be chosen from the given clips, and therefore, we explore the solution space without unnecessary restrictions while satisfying the given constraints.

We define a graph where the nodes are clips and the edges between them indicate whether the clips can be clustered together. We define s_{ij} as a variable indicating whether two clips i and j are clustered together, i.e., $s_{ij} = 1$ if i, j are clustered together and 0 otherwise $\forall i, j$.

In other words,

$$s_{ij} = 1 \Longleftrightarrow i, j \in \text{same cluster},$$
$$s_{ij} = 0 \Longleftrightarrow i, j \notin \text{same cluster}, \quad \forall i, j = 1, 2, \ldots, n. \quad (8)$$

These s_{ij} variables are given as input (constant value = 0) if two clips cannot be clustered together. Else, they can take either 0 or 1 (variable in the formulation). This is based on the condition that two clips cannot be clustered together if the distance constraint is not satisfied. However, they can either be clustered or not, if the distance constraint is satisfied.

ILP Formulation:

Objective: minimize $n - (\sum_{i<j} t_{ij})$

Constraints:

$$t_{ij} \leq s_{ij} \forall i < j, \quad (9)$$

$$t_{ij} \leq 2 - s_{ki} - s_{kj} \forall k < i < j, \quad (10)$$

$$s_{ij} + s_{jk} - 2 \times s_{ik} \leq 1, \quad (11)$$

$\forall i, j$ and k where $1 \leq i, j, k \leq n, i \neq j \neq k$.

Here, constraint (11) enforces the condition that if i and j are in the same cluster and j and k are in same the cluster, then i and k has to be in the same cluster.

Apart from s_{ij}, binary variables t_{ij} are introduced.

Constraint (9) implies that t_{ij} must be 0 if s_{ij} is 0. Even if $s_{ij} = 1$, if there exists k such that $k < i < j$ and both s_{ki} and s_{kj} are 1, then t_{ij} must be 0 too. Therefore, t_{ij} can be 1 if $i (< j)$ is the node with the smallest index in the cluster defined by s = 1 containing the edge ij.

As the sum, $\sum_{i,j} t_{ij}$ is maximized, the edges with $t_{ij} = 1$ will define a spanning forest (i.e., collection of trees) which is a subgraph of the graph defined by the edges with $s_{ij} = 1$.

Here, the summation, $\sum_{i,j} t_{ij}$, indicates the summation of [number of cluster members − 1] of all the clusters. Therefore, it can be observed that the objective $n - (\sum_{i<j} t_{ij}) = K$, where K is the number of clusters as per the 1st formulation.

Example for CHIP Edge. Let there be 9 clips (nodes). Given, a pairwise distance relation amongst these 9 clips, the graph formed (with s_{ij} as edges) at an instance during the optimization is shown in Figure 10. According to the constraints, the variables t_{ij} take the values 0 or 1. The resultant graph with t_{ij} as edges is shown in Figure 11. From this figure, the objective value can be computed, which is $9 - (2 + 3 + 1) = 9 - 6 = 3$ (=number of clusters).

4.2.1. Area-Constrained Clustering. In the case of area-constrained clustering, $D(i,j) = \text{area}(\text{XOR}(x_i, x_j))$ as

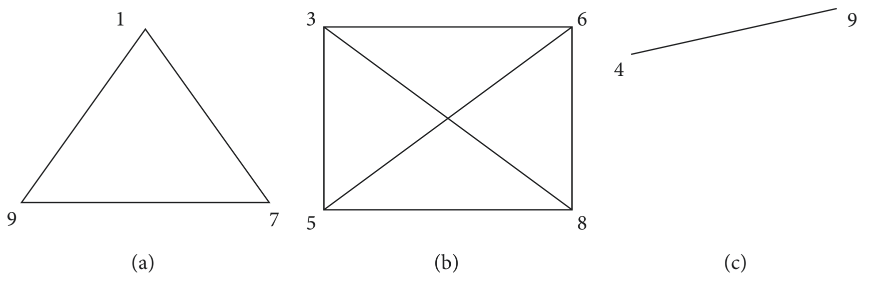

FIGURE 10: Example for the CHIP edge formulation (with s_{ij} as edges).

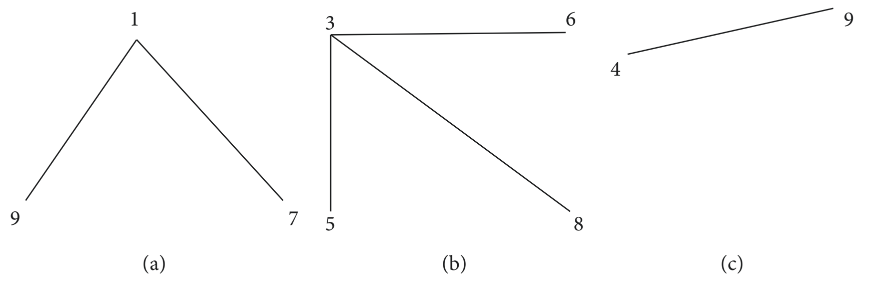

FIGURE 11: Example for the CHIP edge formulation (with t_{ij} as edges).

defined in Section 2.2.1 and $D_c = w \times h \times (1 - a)$, where a is the area-constraint ranging between 0 and 1.

4.2.2. Edge-Constrained Clustering. In the case of edge-constrained clustering, $D(i, j) = \max(e_1, e_2, \ldots)$ as defined in Section 2.2.2 and $D_c = e$, where e is the given edge constraint (in nm).

5. Representative Clip Generation

In this section, we discuss the framework to generate representative clips for the clusters formed in the clustering step. Firstly, each cluster is checked whether there exists any clip among the cluster members that satisfies the constraints to be a representative clip. If a representative clip does not exist amongst the given clips, then we proceed to the following steps: (1) data preprocessing; (2) MILP formulation; and (3) representative clip generation, and these steps are discussed in the following subsections.

5.1. Data Preprocessing. In this step, we build a grid data structure formed along the edges of polygons of all the clips in the cluster. This structure is similar to that used in distance computation in Section 3.2.2, where only two clips are used to form the grid data structure as compared to considering all the clips in the cluster in this step. Using this structure, we can represent each clip in the cluster using a vector where each dimension represents the area covered by a polygon in a particular grid. Each grid data structure is unique with respect to the clusters.

5.2. MILP Formulation. Using the grid data structure, we formulate a mixed integer linear program to find a feasible solution that satisfies the given constraints. This feasible solution is then used to generate the representative clip.

Formulation:

Let $c_1, c_2, c_3, \ldots, c_q$ be a set of clips which belong to a cluster and c_r be its representative clip. Therefore, as per the clustering formulations, $\exists\, c_r$ such that $D(c_r, c_i) \le D_c, \forall\ i = 1, 2, \ldots, n$ where D_c is the given constraint.

Let the number of grids in the grid data structure of a particular cluster be d; that is, the given clips and the representative clip of the cluster can be represented by a d-dimensional vector with corresponding areas (A_j) as upper bound for each dimension. Let the vector be represented by

$$c_i = \left[c_{i_1}, c_{i_2}, c_{i_3}, \ldots, c_{i_d}\right], \tag{12}$$

where each $c_{i_j} \le A_j, \forall j$.

For a cluster, we define another d-dimensional vector called the area vector (A) where $A = [A_1, A_2, A_3, \ldots, A_d]$; that is, each A_i is the area of a grid in the grid data structure.

Based on the grid structure formulation, each given clip in the cluster can be represented by either 0 (empty) or A_j (filled) $\forall j$. Therefore, distance between c_r and c_i can be written as a linear function.

For example, let area vector of a cluster be $A = [\,100\ \ 200\ \ 150\ \ 50\,]$ and one of the clips (c_1) be $[\,100\ \ 200\ \ 0\ \ 50\,] \Longrightarrow D(c_r, c_1) = 100 - c_{r_1} + 200 - c_{r_2} + c_{r_3} + 50 - c_{r_4}$.

Objective: No objective

Constraints: $\forall i$

$$D(c_r, c_i) \le D_c. \tag{13}$$

Bounds: $c_{r_j} \le A_j$

As per the constraints and bounds, each variable (c_{r_j}) takes values from 0 to $A_j \forall j$; that is, it takes continuous values rather than discrete. These values are then used to fill the grids using heuristics discussed in the next subsection.

Finding feasible solution step can be further sped up by removing redundant dimensions (grids) which are either always empty or always filled in all the clips of a cluster.

5.3. Representative Clip Generation. In this subsection, a heuristic is proposed to generate the representative clip as described in Algorithm 1. From the feasible solution of MILP formulation, we obtain a solution vector of continuous variables, where each dimension is in the range $[0, A_j]$. Let the result vector be represented by $c_r = [c_{r_1}, c_{r_2}, c_{r_3}, \ldots, c_{r_d}]$.

In Algorithm 1, c_{r_l}, c_{r_r}, c_{r_t}, and c_{r_b} represent the neighboring grids (left, right, top, and bottom, respectively) of a grid in c_r. In this algorithm, if $c_{r_j} = A_j$, we fill the grid entirely. If $c_{r_j} < A_j$, then the grid has to be filled partially. This can either be done along the x-axis or y-axis, until the condition is satisfied. For uniformity, we design a heuristic function (PREFERENCE in Algorithm 1) to capture the local neighborhood and fill the grid accordingly.

```
(1) function CLIPGENERATION ($c_r, c$)
(2)   for each dimension in grid data structure do
(3)     if $c_{r_j} = A_j$ then
(4)       fill the grid completely
(5)     if $c_{r_j} < A_j$ then
(6)       if PREFERENCE ($l, r, t, b$) = $x$ then
(7)         while fill < $c_{r_j}$ do
(8)           fill the grid with horizontal rows of pixels
(9)       if PREFERENCE ($l, r, t, b$) = $y$ then
(10)        while fill < $c_{r_j}$ do
(11)          fill the grid with vertical rows of pixels
(12) function PREFERENCE ($l, r, t, b$)
(13)   $neighbors = [c_{r_l}, c_{r_r}, c_{r_t}, c_{r_b}]$;
(14)   sort the $neighbors$ vector in descending order;
(15)   if most filled cell is left or right then return $y$
(16)   else return $x$
```

Algorithm 1: Representative clip generation.

Table 1: Benchmarks from ICCAD 2016.

Test case	No. of markers	No. of polygons	Clip size
1	16	77	200×200
2	200	845	200×200
3	5068	9779	200×200
4	264824	147764	250×250

6. Experimental Results

We implemented our approach using C++ programming language with STL and Boost libraries. We use IBM CPLEX Optimizer [17] to solve the integer linear program. We performed the experiments based on the benchmarks provided by ICCAD 2016 Contest as shown in Table 1. A 1.7 GHz dual-core system with a memory of 8 GB is used to perform the evaluation. Since the results reported in contest and the papers are based on experiments conducted in different platforms (8 Core 2.3 GHz KVM Processors and with 64 GB memory), the runtime reported here is used to get a rough estimation but not to be compared with the previous results. In some test cases, we found the optimization to exceed the time limit of 2 hours as per the contest. Therefore, for practical purposes, linear/binary search is performed for the minimal k value, where each iteration of the optimization is limited by time threshold. Without such time threshold, the optimization may consume more than 2 hours.

From Table 2, it can be observed that the ILP formulations, which solve the constrained clustering problem, scale well for the test cases, due to the reduction in data size after exact pattern matching is performed in prior steps. Also, we observe that default case (exact pattern matching) takes majority of the runtime (from Table 3). It can be easily reduced with the parallelization of the exact pattern matching tasks. In future work, the preprocessing steps could be further optimized in order to reduce the bottleneck of our tool and therefore achieve even faster overall runtime for the tool.

Based on Table 4, we achieve better results in most of the test cases in terms of the cluster count as compared to previous work. Since we perform a search for the minimal k value, where each iteration of the optimization is limited by time threshold (for practical run times), optimality is not seen in some of the test cases. Also, due to this search, different cluster counts are seen for CHIP node and CHIP edge, which should otherwise return the same optimal cluster count, i.e., both CHIP node and CHIP edge are optimal in theory. It is also observed that CHIP node is better in practice as compared to CHIP edge in both runtime and cluster count. Even though we do not adopt clip shifting, we achieve results that are comparable to the results in [6], which are best in terms of the cluster count so far but employ clip shifting. Also, clip shifting could be easily added to our formulations to further reduce the cluster count.

7. Conclusion

In this paper, we introduce the problem of layout pattern classification in the integrated circuit design. With several applications in design for manufacturability flow such as hotspot library generation, hierarchical data storage, and systematic yield optimization, clustering the hotspots optimally with good quality representative hotspots is important.

We formally introduce the hotspot clustering problem and briefly discuss the overview of our proposed tool. Then, we introduce the two integer linear program formulations which solve for optimal clusters for the pattern classification problem in IC layout, subject to constraints given by ACC or ECC. Apart from minimizing cluster count, we generate representative clips that best represent the clusters.

Table 2: Constrained clustering—comparison of CHIP v/s others (runtime (s)).

Test case	Constraints	Reference [1] ($T_e + T_c$)	iClaire [13] (T_c)	GRASP [18] ($T_e + T_c$)	Chen et al. [6] (T_c)	CHIP node (T_c)	CHIP edge (T_c)
1	acc = 0.95	1.808	0.004	0.001	0.01	0.01	0.08
	$e = 4$	1.324	0.004	0.001	0.03	0.02	0.06
2	acc = 0.9	1.168	0.006	0.014	0.05	0.09	2.6
	$e = 4$	0.874	0.007	0.017	0.19	0.09	0.68
3	acc = 0.85	1.232	0.020	0.159	0.15	1.09	28.4
	$e = 8$	1.311	0.040	0.161	1.04	6.44	25.1
4	acc = 0.99	4.740	0.1	2.352	21	15.1	35.3
	$e = 2$	4.364	0310	2.428	39	13.4	29.7

Table 3: Exact pattern matching (default constraint).

Test case	iClaire [13]		Our approach	
	#Clusters (CC)	Runtime, T_e (s)	#Clusters (CC)	Runtime, T_e (s)
1	8	0.001	8	0.012
2	26	0.004	26	0.047
3	70	0.060	70	0.760
4	72	4.170	72	125.2

Table 4: Constrained clustering—comparison of CHIP v/s others (cluster count).

Test case	Constraints	Reference [1]	iClaire [13]	GRASP [18]	Chen et al. [6]	CHIP node	CHIP edge
1	acc = 0.95	4	3	3	3	3	3
	$e = 4$	5	5	5	5	5	5
2	acc = 0.9	10	7	11	4	4	5
	$e = 4$	18	18	18	13	18	18
3	acc = 0.85	26	13	13	8	8	10
	$e = 8$	52	37	47	34	39	39
4	acc = 0.99	31	24	24	17	21	21
	$e = 2$	57	46	52	44	48	48

We achieve better results in majority of the test cases as compared to the existing results published in the literature and the reference results reported in ICCAD 2016 contest website. Although the runtime of the ILP is more than the other methods, our main focus in this work is to develop a generic framework to cluster the hotspots in a layout optimally. These formulations describe the given problem exactly unlike other works in the literature which try to adapt the existing clustering algorithms to this problem, with some postprocessing steps. In future work, clip shifting can be adopted to the tool flow to increase the solution space and thereby further reduce the cluster count.

Data Availability

The benchmark data used to support the findings of this study are provided by the ICCAD 2016 Contest organizers and can be found in [1]. The program developed to support the findings of this study are available from the corresponding author upon request.

Disclosure

This paper is a revised version of the corresponding author's thesis [16] submitted to the Graduate College, Iowa State University.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] R. O. Topaloglu, "ICCAD-2016 CAD contest in pattern classification for integrated circuit design space analysis and benchmark suite," in *Proceedings of 35th International Conference on Computer-Aided Design (ICCAD'16)*, pp. 41:1–41:4, ACM, Austin, TX, USA, November 2016.

[2] N. Ma, *Automatic IC hotspot classification and detection using pattern-based clustering*, Ph.D. thesis, University of California, Berkeley, CA, USA, 2009.

[3] P. Morey-Chaisemartin and F. Brault, "Is it time to switch to oasis. mask?," *Solid State Technology*, vol. 58, no. 5, pp. 33–38, 2015.

[4] Y.-T. Yu, G.-H. Lin, I. H.-R. Jiang, and C. Chiang, "Machine-learning-based hotspot detection using topological classification and critical feature extraction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 3, pp. 460–470, 2015.

[5] P. H. Wu, C. W. Chen, C. R. Wu, and T. Y. Ho, "Triangle-based process hotspot classification with dummification in euvl," in *Proceedings of Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*, pp. 1–4, Hsinchu, Taiwan, April 2014.

[6] K.-J. Chen, Y.-K. Chuang, B.-Y. Yu, and S.-Y. Fang, "Minimizing cluster number with clip shifting in hotspot pattern classification," in *Proceedings of the 54th Annual Design Automation Conference 2017 (DAC'17)*, pp. 63:1–63:6, ACM, Austin, TX, USA, June 2017.

[7] J. W. Park, R. Todd, and X. Song, "Geometric pattern match using edge driven dissected rectangles and vector space," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 2046–2055, 2016.

[8] F. Yang, S. Sinha, C. C. Chiang, X. Zeng, and D. Zhou, "Improved tangent space-based distance metric for lithographic hotspot classification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1545–1556, 2017.

[9] D. Ding, X. Wu, J. Ghosh, and D. Z. Pan, "Machine learning based lithographic hotspot detection with critical-feature extraction and classification," in *Proceedings of 2009 IEEE International Conference on IC Design and Technology*, pp. 219–222, Austin, TX, USA, May 2009.

[10] J.-Y. Wuu, F. G. Pikus, A. Torres, and M. Marek-Sadowska, "Rapid layout pattern classification," in *Proceedings of 16th Asia and South Pacific Design Automation Conference (ASPDAC'11)*, pp. 781–786, IEEE Press, Yokohama, Japan, January 2011.

[11] Y.-T. Yu, Y.-C. Chan, S. Sinha, I. H.-R. Jiang, and C. Chiang, "Accurate process-hotspot detection using critical design rule extraction," in *Proceedings of 49th Annual Design Automation Conference (DAC'12)*, pp. 1167–1172, ACM, San Francisco, CA, USA, June 2012.

[12] Y.-T. Yu, G.-H. Lin, I. H.-R. Jiang, and C. Chiang, "Machine-learning-based hotspot detection using topological classification and critical feature extraction," in *Proceedings of 50th Annual Design Automation Conference (DAC'13)*, pp. 67:1–67:6, ACM, Austin, TX, USA, May 2013.

[13] W.-C. Chang, I. H.-R. Jiang, Y.-T. Yu, and W.-F. Liu, "iClaire: a fast and general layout pattern classification algorithm," in *Proceedings of the 54th Annual Design Automation Conference 2017 (DAC'17)*, pp. 64:1–64:6, ACM, Austin, TX, USA, June 2017.

[14] W. C. J. Tam and R. D. Blanton, "LASIC: layout analysis for systematic ic-defect identification using clustering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1278–1290, 2015.

[15] S. V. Dongen, *Graph clustering by flow simulation*, Ph.D. thesis, University of Utrecht, Utrecht, Netherlands, 2000.

[16] R. R. Takkala, "CHIP: clustering hotspots in layout using integer programming," *Creative Components*, vol. 33, 2018, https://lib.dr.iastate.edu/creativecomponents/33.

[17] IBM CPLEX, *Ibm Ilog Cplex Optimizer*, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[18] M. Woo, S. Kim, and S. Kang, "GRASP based metaheuristics for layout pattern classification," in *Proceedings of 36th International Conference on Computer-Aided Design*, pp. 512–518, IEEE Press, Irvine, CA, USA, November 2017.