UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ANDRÉ SALDANHA OLIVEIRA

Pattern Classification for Layout Hotspots

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Engineering

Advisor: Prof. Dr. Ricardo Reis
Coadvisor: Dra. Carolina Metzler

Porto Alegre
January 2018

ABSTRACT

The final objective of an integrated circuit design is to produce a layout, that is, a geometrical representation of the circuit where the geometrical shapes correspond to patters that will be formed by layers of metal, oxide, and semiconductors. These patterns are essentially descriptions that will be used to print the circuit through chemical, thermal and photographic processes.

To ensure the layout can be used to print the circuit with no defects, it is necessary to run design rules check. This verification searches for patterns that violate design rules, which makes it impossible to guarantee defect-free printing. However, some layout patterns may present printability problems even when design rules are respected. To solve this problem, physical verification flows are applied to the layout with the objective of detecting and treating such patterns. The sheer number of these layout printability hotspots and the fact that they are sometimes similar to each other suggests that the physical verification flow can be sped up by clustering together similar patterns.

In this work, we address the problem of complex shape partitioning, incorporating an algorithm with complexity $\mathcal{O}(n^{5/2})$ into the layout hotspot clustering flow, which allows for clustering of hotspots in benchmarks with complex polygons. Furthermore, a study of the viability of a machine learning flow for incremental clustering is conducted, covering the choice of features and analysis of candidate models.

Keywords: Physical Design. Verification. Machine Learning. Layout. EDA. Partitioning. Microelectronics.

RESUMO

O objetivo final do fluxo de projeto de um circuito é produzir um leiaute, uma representação geométrica do circuito, onde as formas geométricas correspondem aos padrões que serão formados por camadas de metal, óxido e semicondutores. Esses padrões são essencialmente descrições que serão usadas para imprimir o circuito através de processos químicos, térmicos e fotográficos.

Para garantir que o leiaute possa ser usado para impressão de um circuito integrado sem defeitos, é necessário executar verificações de regras de projeto. Essa verificação encontra padrões que violam regras que inviabilizariam a garantia de impressão sem defeitos. Porém, alguns padrões do leiaute podem apresentar problemas na impressão mesmo quando a checagem das regras de projeto não encontra erros. Para solucionar esse problema, fluxos de verificação física são aplicados no leiaute com o objetivo de detectar e tratar tais padrões. A grande quantidade de regiões com problemas de impressão e a similaridade entre elas sugere que o fluxo de verificação física pode ser acelerado ao se agrupar padrões similares.

Neste trabalho, o problema de particionamento de polígonos complexos é abordado, e um algoritmo de particionamento de complexidade $\mathcal{O}(n^{5/2})$ é incorporado ao fluxo de classificação e agrupamento de regiões de interesse, permitindo que casos de teste com polígonos complexos tenham suas regiões de interesse agrupadas. Além disso, um estudo sobre a viabilidade de um fluxo de aprendizado de máquina é conduzido, cobrindo a escolha de atributos e a análise de diferentes modelos candidatos.

Palavras-chave: Verificação. Síntese Física. Aprendizado de Máquina. Leiaute. EDA. Particionamento. Microeletrônica.

LIST OF ABBREVIATIONS AND ACRONYMS

CAD	Computer Aided Design
EDA	Electronic Design Automation
DRC	Design Rule Check
ICCAD	International Conference on Computer Aided Design
UFRGS	Universidade Federal do Rio Grande do Sul
VLSI	Very Large Scale Integrated
SVM	Support Vector Machine
RF	Random Forest
NN	Neural Network

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

While the semiconductor design shrinks, the whole physical design flow grows more dependant on lithography constraints. The main pattern printing process still uses 193nm lithographic process (SHIN; LEE, 2016), even though the patterns drawn are much smaller. The techniques that allow such thin patterns to be drawn can disturb patterns in the surroundings. In order to detect the patterns that could suffer lithographic printing problems, also called lithographic hotspots or simply hotspots, the standard typically flow uses optical simulations. This technique has great hotspot detection, but it requires high computational costs to be spent in the optical simulation. As the authors describe in (SHIN; LEE, 2016), the geometric verification methods have been introduced as an alternative. In their work, they use the layout image as input to a convolutional neural network in order to detect hotspots.

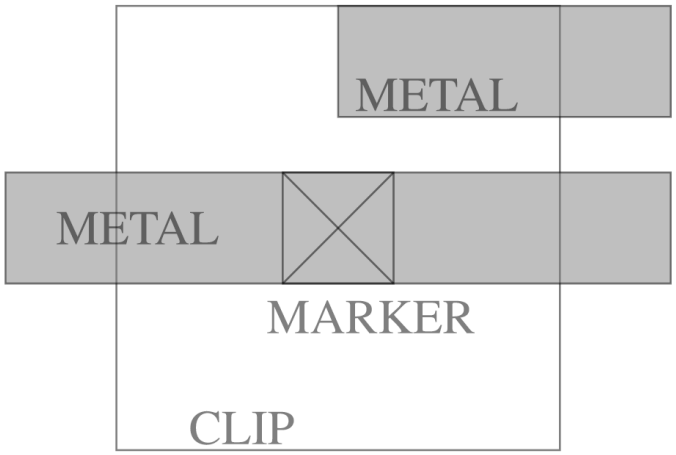
Regardless of how the hotspots were detected, in order to continue the flow, any pattern with a detected layout printability problem is marked in the layout. Because of the large number of problematic patterns found in general, classifying the patterns based on geometrical similarity has an important role in determining the efficiency of hotspot detection. In the hotspot pattern classification flow, similar hotspots are classified into a cluster based on geometrical similarity (CHEN et al., 2017).

1.1 Problem Formulation

The tool developed follows the problem specification from 2016 CAD contest at ICCAD for pattern classification(TOPALOGLU, 2016). The contest provides as input a circuit layout. The layout, following the contest definition, has two relevant layers. The first layer has the metal polygons. The last layer contains polygons that represent markers, but have no physical meaning. The markers are generally small polygons on which a pattern where a layout printabilty problem has been detected. These markers are simply small rectangles that point where a layout printability problem was detected. In an area centered in any point within these markers, a rectangle is created, and the shapes that are contained within this rectangle are cut. The result of this extraction is called a clip. The clip contains only the part of the shapes that intersects with its boundary. This clipping process is illustrated in figure 1.1, where a clip is being extracted with its center point

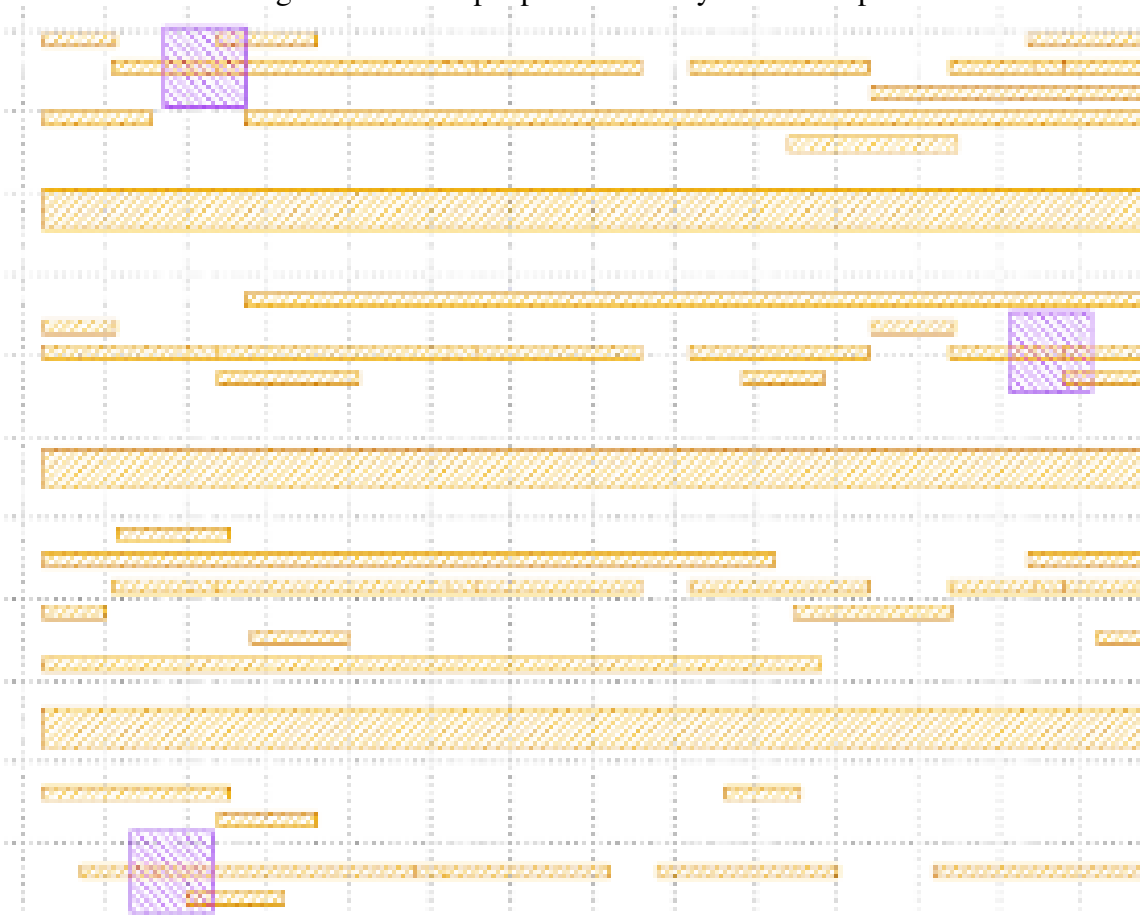
in a marker. The two metal rectangles have their area that is contained within the clip extracted and added to the clip. In this particular example, both metal rectangles are cut, and only a sub-rectangle of each is added to the actual clip. The clip can be centered in any point inside the marker.

Figure 1.1: Structure of a hotspot in the layout.



The program input also specifies the width and height of the clip. Given a marker, one can choose a point contained in it and generate a clip centered in such point with the given dimensions. Figure 1.2 shows this exact process. The white canvas represents lack of any material. The yellow polygons represent metal, and the purple polygons are the clips. Note that the clips are centered in the markers. The main objective is to provide a reduced set of representative layout clips contained on these markers. The tool developed that implements the clustering flow is described in Chapter 2.

Figure 1.2: Example picture of a layout with clips.



To perform the pattern classification two parameters are selected by the user: (A) Area match constraint that selects the clustering by area constraint and (B) Edge displacement constraint that performs the clustering by edge constraint. The objective of clustering is to provide sets of clips (clusters) that resemble each other according to these parameters.

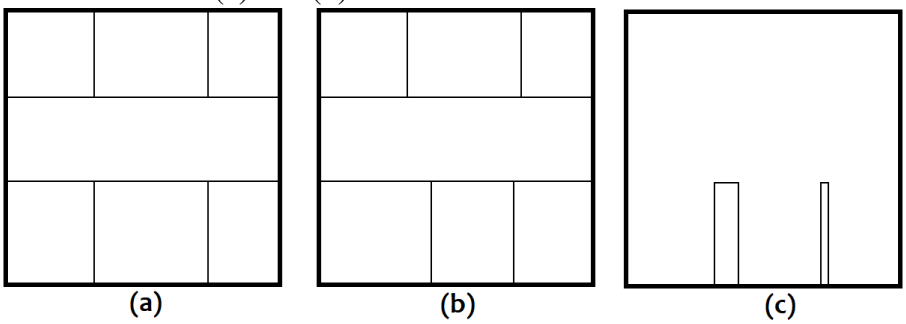
1.1.1 Area Constrained Clustering

In area constrained clustering (ACC) mode, a topological classification is performed based on the area match constraint parameter α. This parameter defines the maximum area match percentage. The overlap area between any clip of a cluster (A) and the representative clip of a cluster (B) must satisfy this parameter, as shown in the equation 1.1, bellow. In Figure 1.3 a graphical example is provided. In the figure, there are two clips (a) and (b) that need to be compared in terms of area. The result of the geometric XOR applied upon them is represented in the resulting clip (c).

$$\frac{[Area(XOR(A,B))]}{w*h} \leq (1-\alpha) \tag{1.1}$$

The Area() function gives the total area of the polygons and the XOR gives the geometric difference between the clips A and B. And (w, h) are the dimensions of the clip.

Figure 1.3: Geometric XOR example. (a) and (b) are two clips, and (c) is the result of the geometric XOR between (a) and (b)

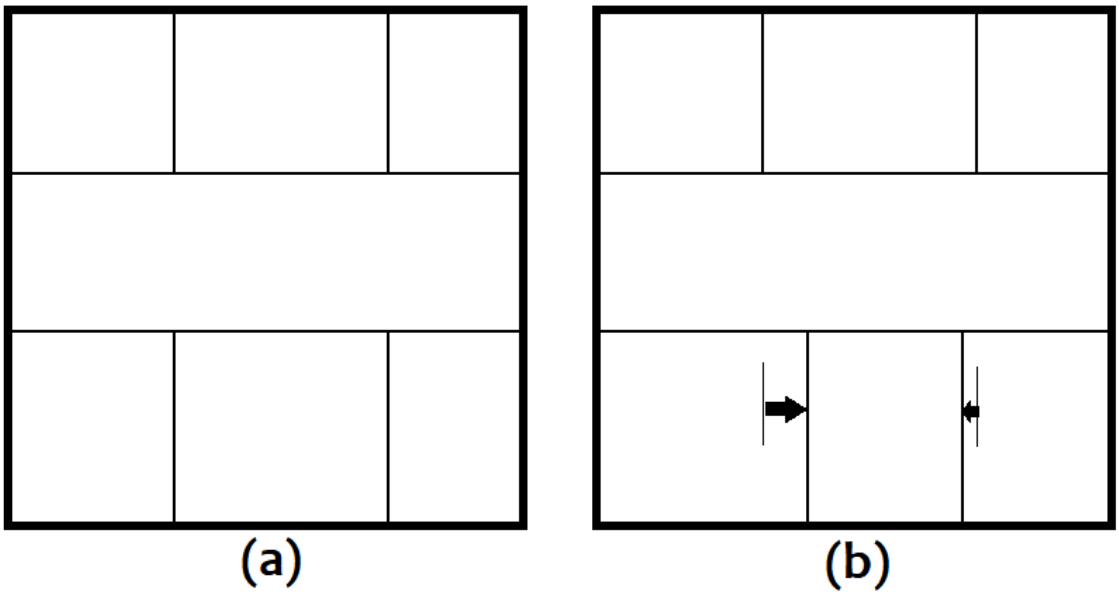


1.1.2 Edge Constrained Clustering

The edge constrained clustering (ECC) mode receives a parameter e in nanometers that indicates how much the edge will be shifted, the shifting can be in both directions: inward and outward. This parameter e defines the tolerance that the clip pattern can shift. Figure 1.4 shows an example of the edge being shifted. Multiple edges can shift by different amounts since it is limited by e value.

Edges only shift in orthogonal projections and the representative clip of a cluster after the shifting becomes another clip of the same cluster.

Figure 1.4: Edge displacement example. (a) and (b) are two clips. (b) differs to (a) in edge displacement, showed by the arrows.



1.2 Complex Polygons

During the layout hotspot clustering flow, the presence of complex polygons in some layouts pose a problem. Efficient clip extracting and clip comparison algorithms require all polygons in the layout and within each clip to be a rectangle. For that reason, classifying a polygon as complex in this context means that it isn't a rectangle. In order to make possible for the flow to support layouts with complex polygons, there is the need of an algorithm capable of dividing a complex polygon into a set of rectangles.

The complex polygon partitioning problem is addressed in Chapter 3. In order to better understand the problem of complex polygon partitioning, Section 3.2 presents a study of the literature. Since there are multiple applications for complex polygon partitioning that extend beyond the domain of VLSI, the problem is well known and has efficient solutions in the literature. The incorporation of an efficient algorithm to the flow is described in Section 3.3.

The improved flow and the results obtained in the 2016 ICCAD contest benchmark suite (TOPALOGLU, 2016) are shown in Chapter 4. First, in 4.1 the benchmarks are analyzed in order to obtain relevant statistics such as the number of clips. Then, in Section 4.2 the methodology used is described. The results and their interpretations are in Section 4.3 and the conclusion of the experiment is in Section 4.4.

1.3 Machine Learning Incremental Flow

Another idea suggested by looking into the particularities of the problem, such as the high similarity between clips within the same cluster and the strong emphasis in geometrical characteristics of the comparison process, is the development of a machine learning flow for incremental clustering. Given an initial clustering, a classifier could be trained to cluster any additional clip into its most probable cluster. If such an incremental flow were to be implemented, after a database of clustered clips was generated, there would be no need to run the clustering flow again for a new layout. All clips extracted from the new layout would be assigned to its cluster in constant time after the feature extraction.

The machine learning flow study will be addressed in Chapter 5. Section 5.2 describes how information relevant to classification was extracted from the clips, and Section 5.3 how the model choice was made. Finally, the results of the experiments and the conclusion are discussed in Section 5.5.

2 CLUSTERING FLOW

The clustering flow described in 1.1 has been implemented in a tool. The tool starts by parsing the input layout file, where it finds the markers and initializes its layout data structure with the material polygons. The markers are provided as an input and since they are not points, but shapes, the clip centering can be arranged in order to reduce the number of clusters. In this situation, given the layout in GDS file containing the markers and the clip size, the clips are centered in the midpoint and for each marker a clip is created. The extraction of the clips uses the points of the shapes instead of a grid matrix. Our flow targets a fast and accurate solution.
In order to achieve competitive runtime, we implemented a greedy algorithm to cluster the clips together. The list of clips are shuffled, then the first unclustered clip is selected as a representative of a new cluster. This representative is compared with every other unclustered clip in the list, and if there is a rotation or mirroring of the second clip that satisfies the constraint the clip is added to the cluster. This is repeated until every clip is clustered, and the list of clusters is returned. The clips' total material area is compared before considering a full comparison, because if the material area of the two clips is too different there is no reason to compare all different configurations because they will not be compatible regardless of the rotation. The clustering pseudo-algorithm 1 shows the algorithm used for clustering. The function needs to receive a list of the previously extracted clips, the constraint and the operation mode. For example, it could run on area mode with a constraint of 0.95, or 95%, or edge mode with a 2nm constraint.

The algorithm first shuffles the list of clips to obtain a new order. It also initializes an empty list of clusters to be populated during the algorithm. These two initialization steps can be seen in the lines 2 and 3 of the algorithm.
In line 4, the algorithm starts iterating through the clips. For each clip it iterates through, if the clip is already part of a cluster it is skipped, as seen in lines 5 and 6. If not, a new cluster is created, the current clip is clustered in it and is set as the cluster's representative, all of which is done in lines 8 to 10.
From line 11 to 18, all clips starting from the next in the list of clips are compared to the representative. The function compare_clips() in line 12 performs the comparison. It keeps the first clip constant and generates all possible arrangements applying rotations and mirrorings on the second clip and then performing the comparison, returning the smallest

Algorithm 1 Greedy clustering algorithm.

```
1:  procedure CLUSTER(list_of_clips, constraint, mode)
2:      clips ← shuffle(list_of_clips)
3:      clusters ← {}
4:      for all clip in clips do
5:          if clip.get_clustered() then
6:              continue
7:          else
8:              new_cluster ← create_cluster()
9:              new_cluster.representative ← clip
10:             clip.set_clustered()
11:             for all other_clip in clips do
12:                 difference ← compare_clips(clip, other_clip, mode)
13:                 if difference > constraint then
14:                     continue
15:                 else
16:                     new_cluster.add_clip(other_clip)
17:                     other_clip.set_clustered()
18:                 end if
19:             end for
20:             clusters.add_cluster(new_cluster)
21:         end if
22:     end for
23:     return clusters
24: end procedure
```

difference possible. In area mode, a geometric XOR is applied between the first clip and all the arrangements of the second clip, and the smallest difference is returned. In the edge mode, the biggest edge difference of the best rotation is returned. If the difference is smaller than the constraint requires, the second clip is added to the cluster.

Finally, the cluster being generated, it is added to the list of clusters. After all the clips have been iterated through, the list of clusters is returned.

3 COMPLEX POLYGON PARTITIONING

3.1 Introduction

As previously stated in Chapter 1, efficient clip extraction and comparison requires all polygons in the layout to be rectangles. Furthermore, it is required that the rectangles have two sides aligned with the y-axis and two sides aligned with the x-axis. However, some of the benchmarks, namely testcase3 and testcase4, have complex polygons. The specifics of each benchmark will be discussed in Section 4.1.
As defined in Chapter 1, a complex polygon in this context is any polygon that is not a rectangle. These polygons need to be partitioned into rectangles before the clip extraction takes place. This process is called Rectilinear Polygon Partitioning.
Efficient rectilinear polygon partitioning is a quite difficult problem, even if intuition says otherwise. The objective of the partitioning is to generate as few rectangles as possible. As a secondary objective, the runtime of the partitioning may also be important, but in this context it is not the main objective since we consider the polygon partitioning as a pre-processing of the layout and time performing the partitioning isn't taken into account for runtime metrics.

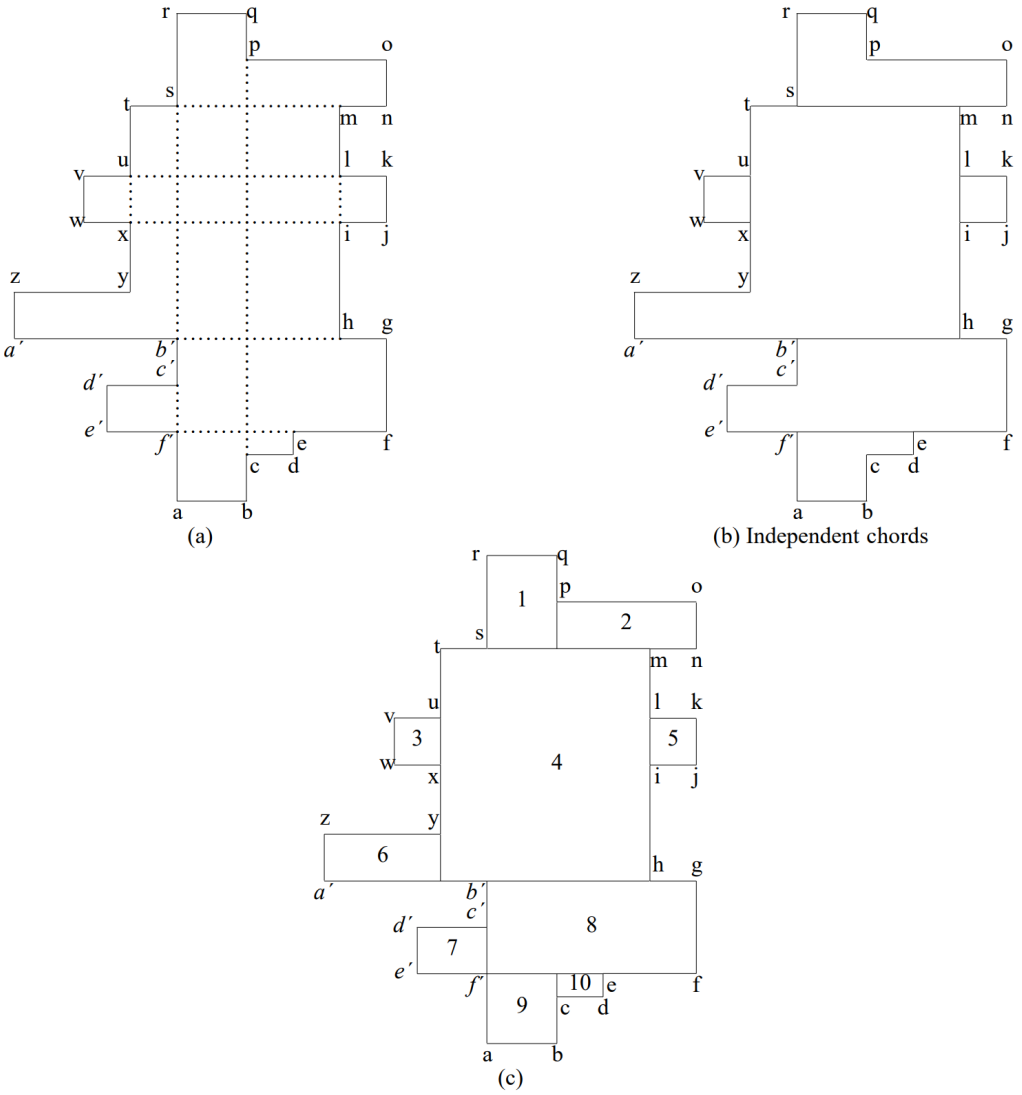
3.2 State of the Art

In his work, (OHTSUKI, 1982) has proposed an algorithm with complexity $\mathcal{O}(n^{5/2})$, where n is the number of vertices of the polygon. One of the main contributions of the work is the use of both vertical and horizontal cuts. Their algorithm runs in three steps:

1. Find all chords - line segments contained within the polygon - that connects two covertical or cohorizontal concave vertices. The chords are then used to create a graph, where each node is a chord and there are connections between any two chords that intersect. Figure 3.1 (a) shows all of these chords.
2. Compute the maximum independent set of this graph, and all chords belonging to this set are drawn, cutting the polygon. Figure 3.1 (b) shows the polygon with these chords drawn, cutting the polygon.

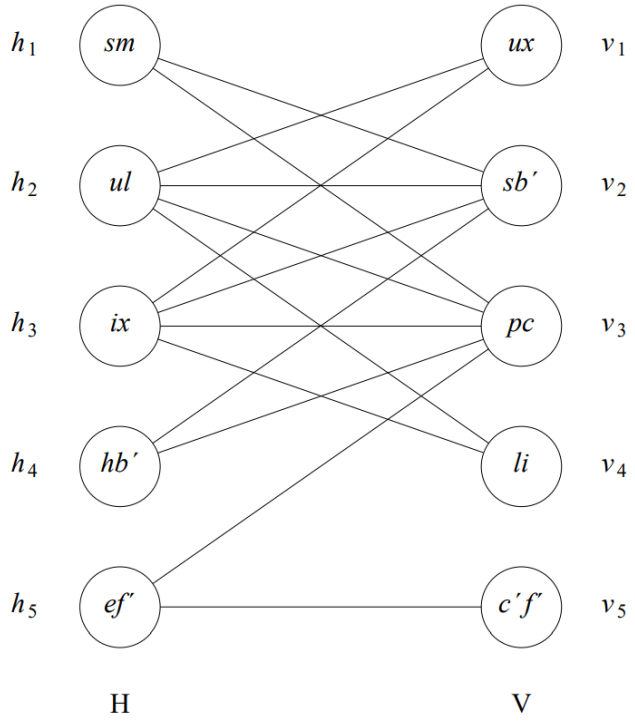
3. From each of the concave vertices from which a chord was not drawn in step 2 draw a maximum length vertical line that is wholly within the smaller rectilinear polygon created in step 2 that contains this vertex. Figure 3.1 shows the final partitioning.

Figure 3.1: Polygon being partitioned by (OHTSUKI, 1982)'s algorithm.



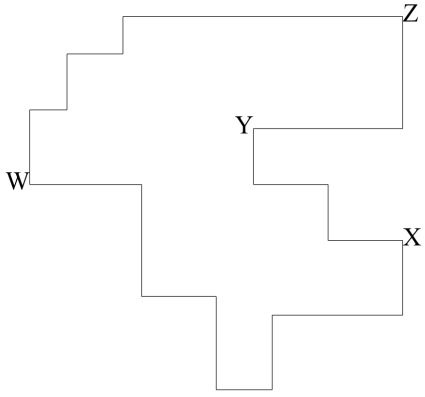
Note that any horizontal (vertical) chord is independent of all other horizontal (vertical) chords. The graph produced in step 2 is, therefore, a bipartite graph. Figure 3.2 is the graph generated from 3.1 (a). The left vertices of the graph are the horizontal chords, and the right vertices are the vertical chords. As stated in step 1, a connection between two nodes in this graph means the two chords intersect each other in the original polygon.

Figure 3.2: Intersection graph for the polygon in figure 3.1.



In their work, (NAHAR; SAHNI, 1988) defined a measure for polygon complexity. This measure uses the number of inversions, denoted by k. It is defined by the number of inversions of a polygon while walking along its sides. For example, on figure 3.3, when starting on vertex W and walking counterclockwise, the horizontal direction changes from right to left on vertex X, back to right on vertex Y and back to left on vertex Z. This means the number of horizontal inversions is 4, and therefore $k_h = Horizontal\_Inversions/2 = 2$. The vertical complexity is calculated analogously, and the vertical complexity $k_v = 1$. The polygon's complexity is then defined by $k = min\{k_h, k_v\}$. The number of inversions of the polygon on figure 3.3 is, therefore, 1.

Figure 3.3: Example of a complex polygon and its horizontal complexity relevant points.



The authors went further and analyzed 2896 polygons from VLSI mask data, provided by Sperry Corporation, and calculated the statistics in regards to tje number of inversions. They found out that 99.4% of the polygons had a k smaller than 4, 86% of the polygons having a k of 1, which leads to the conclusion that most polygons found in practice have a small number of inversions.
(WU; SAHNI, 1994) proposes two algorithms for rectilinear polygon partitioning. The complexity of the two algorithms are $\mathcal{O}(nk)$ and $\mathcal{O}(n \log k)$, where n is the number of vertices of the polygon and k is the number of inversions as defined previously. With some very complex polygons, where k is a linear function of n, the worst case complexity is $\mathcal{O}(n^2)$. However, as (NAHAR; SAHNI, 1988)'s analysis found out, for most practical polygons k is small, thus making the algorithm quite fast in practice. Their work describes a way of classification of some sides of the polygon as support or reflex edges. An edge is called a support edge if its two vertices are both convex, and it's called a reflex edge if both vertices are concave. Using these definitions, the authors prove that $k_h = number\_of\_right\_support\_edges + number\_of\_left\_reflect\_edges$. Analogously, it can be proved the same for the number of vertical inversions. Polygons with $k \leq 3$ are partitioned using an algorithm with complexity $\mathcal{O}(n)$. When $k > 3$, the polygon is decomposed into 2k subpolygons with k = 1, thus the complexity of $\mathcal{O}(n \log k)$.

3.3 Implementation

The algorithm proposed by (OHTSUKI, 1982) has been implemented by (LYSENKO, 2014) and published on GitHub, licensed under MIT. This implementation was incorporated into the flow because of the lack of a good implementation of a better algorithm, such as (WU; SAHNI, 1994)'s. Since the code was developed in JavaScript, the actual integration into the flow is more difficult.
The flow that was implemented was quite convoluted, integrating code in three different languages in order to achieve the final objective. Figure 3.4 shows the new parsing flow. The C++ code reads the GDSII input file and uses our own GDSII library to read the polygon descriptions. If the polygon is a rectangle, it is directly added to the list of shapes of the layout. If it's complex. the complex polygon is translated into a string representation and a pipe is opened to call the partitioning script. The script then outputs its own string representation of the rectangles. The rectangles are read and added to the layout's list

of rectangles, and an entry is created in the mapping file. This mapping file contains all complex polygons and the rectangle partitioning that was applied by the script. Using this file, the secondary flow can be optionally run. In this flow, the mapping file is processed by a Python script. This script uses (HUNTER, 2007)'s library to translate the points into a polygon and finally generate a file containing the graphical representation of all complex polygons alongside the partitioning that was previously generated. This optional flow allows for verification of the partitioning algorithm's correctness and analysis of the complex polygons of the layout.

Figure 3.4: Flow diagram of the parsing, including the partitioning script call.

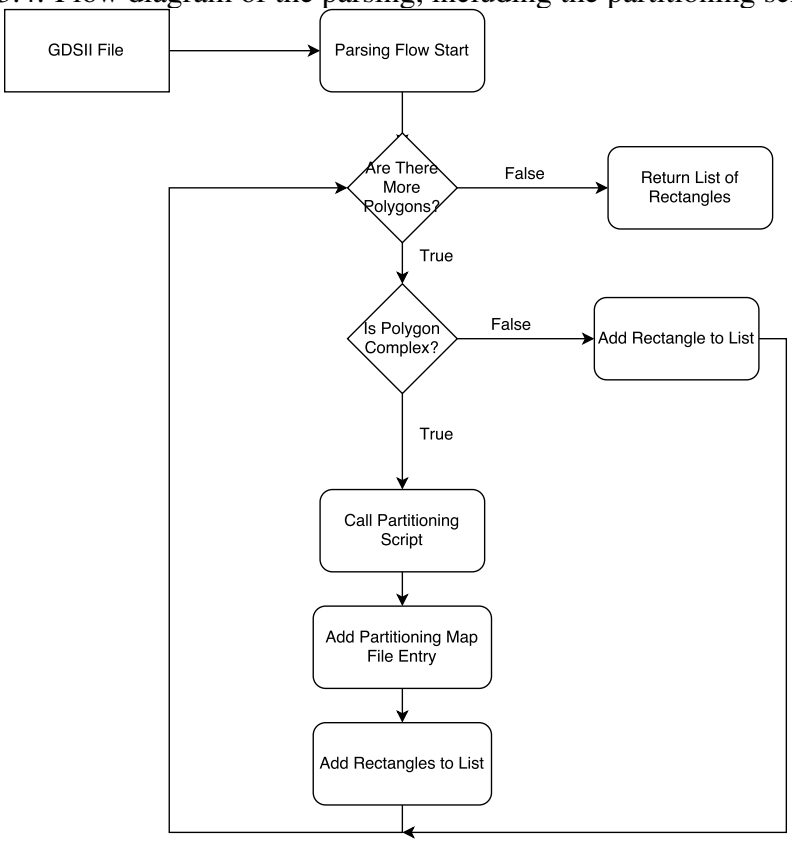
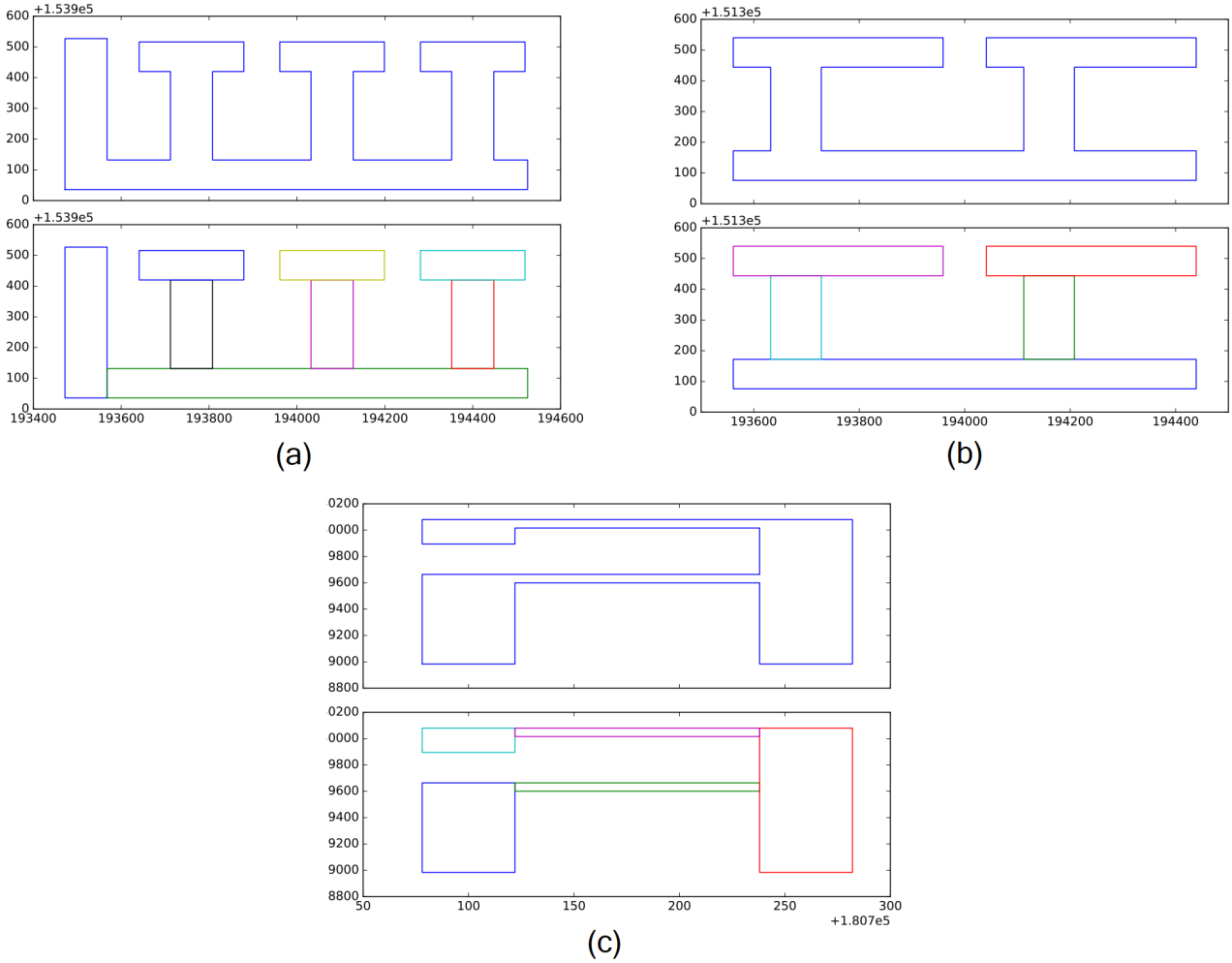


Figure 3.5 contains three examples of the output of the second flow. Figure 3.5 (a) contains a polygon with 30 vertices and k=4, (b) contains a polygon with 20 vertices and k=2, and (c) contains a polygon with 14 vertices and k=2. In fact, as observed by (WU; SAHNI, 1994), these polygons have a relatively low k when compared to the number of vertices, and, therefore, would allow for near linear complexity.

Figure 3.5: Three complex polygons and their partitioning.



4 IMPROVED FLOW

4.1 Introduction

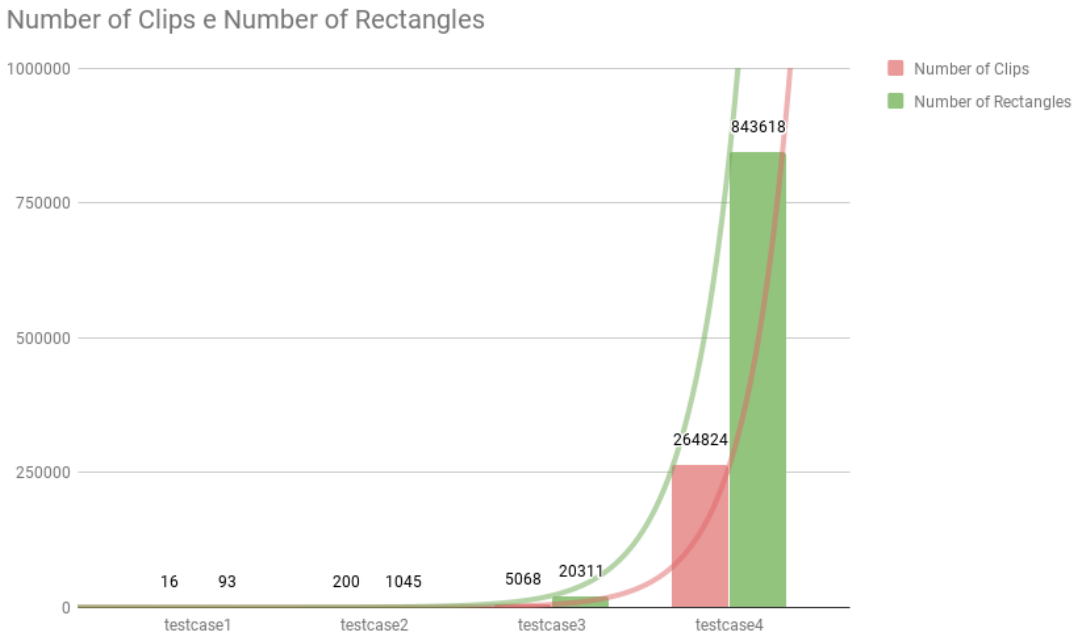
With the incorporation of a polygon partitioning algorithm in the flow, the tool can now run on layouts with complex polygons. The benchmark suite used was obtained from 2016 CAD contest at ICCAD for pattern classification(TOPALOGLU, 2016). There are four different benchmarks, which have their statistics are shown in table 4.1. Only testcase3 and testcase4 have complex polygons.

Table 4.1: Statistics of the four benchmarks.

Benchmark	Number of Clips	Number of Complex Polygons	Number of Rectangles in Layout
testcase1	16	0	93
testcase2	200	0	1045
testcase3	5068	2256	20311
testcase4	264824	1400	843618

The number of rectangles in the layout is counted after the complex polygons are partitioned. From this table, it can be seen that testcase3 and testcase4 are dramatically larger than testcase1 and testcase2. In order to illustrate this increase, figure 4.1 shows a plot of the number of clips and number of rectangles.

Figure 4.1: Graph showcasing the size progression of the benchmarks.



The two tendency lines seen are approximations of the size progression. The green tendency line maps to the number of rectangles, and is defined by the equation $N_r = 2.26 * e^{3.89x}$ and the red tendency line maps to the number of clips, and is defined by the equation $N_c = 15.2 * e^{3.64x}$, where x is the benchmark index in both equations. To put into perspective, if the trend continued and there was a fifth benchmark, its number of clips and rectangles would be $1.3 * 10^7$ and $3.2 * 10^7$ respectively. Even though it is not expected that layout sizes would scale according to these equations, this example is used to showcase the sheer size of the last two test cases, specially testcase4.

4.2 Methodology

The methodology used had as an objective achieve 95% confidence interval with 5% significance. This means 95% of all sets of independent experiments have the results within a 5% range of the result obtained. The tool executions were all done completely independently, and the time measurements were done with microsecond resolution. The machine used has 48GB of RAM. Its processor is an Intel®i7-3930K that runs at a frequency of 3.2GHz. The operating system used is Ubuntu 16.10.

In order to achieve target confidence level of 95% with a significance of 5%, all benchmarks with all their configurations were first run 10 times. From this experiment, we

calculate the number of experiments necessary in order to achieve the confidence interval desired. The tool was then executed this number of times for each configuration.

4.3 Results

Table 4.3 shows the results with respect to maximum cluster size, table 4.2 shows the number of clusters obtained for each benchmark, and finally table 4.4 shows the runtimes in miliseconds. We compare our results with the top 3 tools in the ICCAD 2016 contest and (CHANG et al., 2017) for maximum cluster sizes, and with the top 1, (CHANG et al., 2017) and (CHEN et al., 2017) for cluster counts.

Table 4.2: Cluster count results.

	TOP 1	Chen17	iClaire	Ours
testcase1	8	8	8	8
testcase1ap95	4	3	3	4
testcase1e4	5	5	5	5
testcase2	26	20	26	26
testcase2ap95	13	5	11	12
testcase2ap90	7	-	7	8
testcase2e4	18	13	18	18
testcase3	70	70	70	70
testcase3a85	39	15	13	20
testcase3e8	57	51	37	76
testcase4	72	72	72	72
testcase4a99	69	26	24	26
testcase4e2	56	64	46	184

The results obtained with the benchmarks with no complex polygons are all very similar, except for (CHEN et al., 2017) on testcase2 default, area 95% and edge modes, where it obtains fewer clusters than the other tools.
On testcase3, all results were the same in the default run. In the area 85% run, our results were better than the contest winner but worse than both (CHEN et al., 2017) and (CHANG et al., 2017). In the edge mode run, we performed worse than the other tools.

On testcase4, all tools performed the same in the default run. When running in area mode, our result is almost as good as (CHANG et al., 2017) and ties with (CHEN et al., 2017), while being much better than the contest winner. Finally, when running on edge mode, our tool doesn't perform well, generating many more clusters than the others.

Table 4.3: Results of maximum cluster size.

	TOP 1	TOP 2	TOP 3	iClaire	Ours
testcase1	5	5	5	5	5
testcase1ap95	6	5	6	9	6
testcase1e4	5	5	5	5	5
testcase2	104	104	104	104	104
testcase2ap95	106	104	106	106	106
testcase2ap90	114	104	111	112	112
testcase2e4	104	138	104	104	104
testcase3	792	792	792	792	792
testcase3a85	2056	2080*	1344	2608	2176
testcase3e8	792	1056*	792*	1056	792
testcase4	193370	193370	192240	193370	193370
testcase4a99	197830	197830	197660	197830	197830
testcase4e2	193370	197830*	192240	193710	192240

In the largest cluster size table, results that was obtained but are invalid are denoted by *, and are not used in any comparison. The validity of the runs were obtained from (CHANG et al., 2017).

Once again, in the benchmarks with no complex polygons, all tools performed similarly. The only run that stands out is top 2's run on testcase2 with edge mode, where the largest cluster obtained was dramatically larger than the other tools'. However, given that this tool generated invalid results for all edge mode runs except for testcase1's, this result is doubtful at best.

On testcase3, the default run's results were all the same. With area 85% mode, our tool performed better than every other except for (CHANG et al., 2017)'s. Running in edge mode, once again our result is only beaten by (CHANG et al., 2017)'s, while we tied with first place's tool.

For testcase4, the default run has every tool tied except for the third place's tool that per-

formed slightly worse. In the area mode, our tool ties with the top 1, top 2 and (CHANG et al., 2017) as best results. For edge mode, our tool is tied with the top 3 as the worst performance.

Table 4.4: Runtime results in ms on all benchmarks and configurations.

	TOP 1	Chen17	iClaire	Ours
testcase1	5	10	1	0.165
testcase1ap95	5	10	5	0.057
testcase1e4	5	30	5	0.508
testcase2	10	30	4	4.523
testcase2ap95	12	60	11	5.950
testcase2ap90	11	-	11	6.097
testcase2e4	9	200	11	11.51
testcase3	60	220	60	366.1
testcase3a85	400	370	80	747.6
testcase3e8	70	1340	100	771.0
testcase4	21000	136000	193370	1479178
testcase4a99	268000	152000	197830	1393994
testcase4e2	20000	169000	193710	1044263

The last table showcases the runtimes obtained by the first place, (CHEN et al., 2017), (CHANG et al., 2017) and ours. For the benchmarks with no complex polygons, our tool runs beat every other tool except for testcase2 on edge mode, where we are slower than top 1 and slightly slower than (CHANG et al., 2017).

On testcase3, our runtimes start to ramp up. Our tool is in the order of 10 times slower than both the contest winner's and (CHANG et al., 2017) and slower than (CHEN et al., 2017)'s except when running in edge mode.

Finally, on testcase4, our tool is, once again, one order of magnitude times slower than both (CHANG et al., 2017) and (CHEN et al., 2017), and two orders of magnitude slower than the contest winner for the default run.

4.4 Conclusion

Based on the runtime results, we can conclude that there are scalability issues that need to be addressed. The extraction time of clips on testcase3 are very close to the algorithm's runtime, and the extraction time of clips on testcase4 dominates completely the runtime, taking on average 70% of the runtime. This is because the clip extraction doesn't expect such gigantic number of rectangles, and simply checks for intersections between a clip being extracted and every rectangle in the layout. This module is a strong candidate for optimization.

Taking into account only the qualitative results, that is, the number of clusters and maximum cluster sizes, we conclude that the tool, while performing well, has plenty of room for improvement. One aspect of the problem that was partially neglected by the algorithm is the choice of a good representative. The first clip inserted in a cluster becomes the representative, but a clip inserted later could become the representative and make the clustering better. This problem remains open in our algorithm and could be a source of improvement of the quality.

5 MACHINE LEARNING FLOW

5.1 Introduction

In Chapter 1 it was suggested that a machine learning flow could provide a fast incremental classification. The interest in such a tool is quite simple. After training a classifier, new clips can simply be classified into its probable cluster in constant time. After building a good database of clusters and clips, the classifier can incrementally classify the layout hotspot instead of running the whole flow again.

There are advantages to both the standard clustering flow and the machine learning incremental one. If, for instance, a clip that doesn't belong in any of the clusters appears, the classifier will classify the clip in one of the existing clusters, potentially causing problems. Furthermore, after many iterations and new clips being generated, the clustering could be inefficient or wrong, and the classifier would not be able to even detect such a problem. Of course, if the classification flow is executed only as an incremental tool, the main flow can be used again when necessary on the new database and then the classifier can be trained with the new clusters.

In order to implement the machine learning flow, there are two fundamental steps. The first is the feature extraction, where information is obtained from the clips in the database. Second, the machine learning model must be chosen carefully. The details about how these two steps were approached will be discussed in the Sections 5.2 and 5.3.

5.2 Feature Extraction

Features are individual measurable values of what is being observed (BISHOP; PARK, 2012). In this context, we are interested in generating values based on the clip that can be used to classify it. For example, the total area of material in the clip could be useful because within a cluster all clips are expected to have a similar area of material. The number of shapes could be useful as well, however, since two clips can be exactly equal in the context of area/edge clustering but have a different number of rectangles, the classifier may choose not to take it into account since the gain of information could not be meaningful. Some models allow for the feature importance to be extracted. This could be useful in order to reduce the complexity of the flow as a whole, since irrelevant features

can be discarded without loss in classification accuracy.
The goal with the feature extraction at this level is to generate enough information for the model to use. The model will base its classification taking the feature importance into account during the training. It can be expected that after the training any unimportant features will have almost no weight in the classification, whereas important features would have a large impact.
Extracting meaningful information is fundamental in order to create an accurate machine learning flow. We chose six different features:

- The total area of material.
- The number of rectangles contained within the clip.
- Area of material remaining after geometric Xor with a clip consisting of a horizontal strip.
- Area of material remaining after geometric Xor with a clip consisting of a vertical strip.
- Edge displacement difference between the clip and a clip consisting of a horizontal strip.
- Edge displacement difference between the clip and a clip consisting of a vertical strip.

The choice of using the comparison between the clip whose features are being extracted and two seemingly arbitrary clips is used as to generate a measure of the resemblance of the two clips. Two clips that have a similar comparison result when compared with the two stripped clips then they likely belong to the same cluster. The clip consisting of a horizontal strip is shown in figure 5.1 and the clip with a vertical strip is shown in figure 5.2

Figure 5.1: Artificial clip consisting of a horizontal strip of material crossing the middle of the clip.

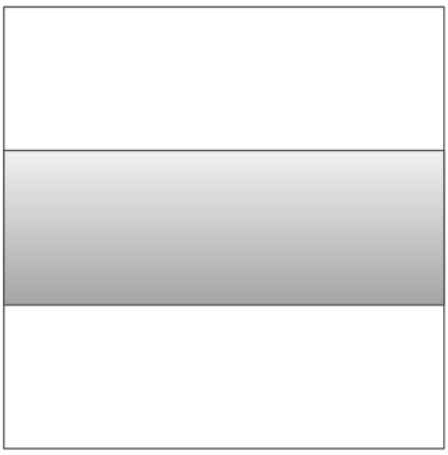
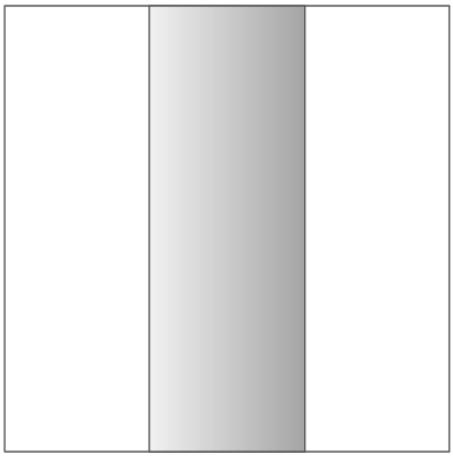


Figure 5.2: Artificial clip consisting of a vertical strip of material crossing the middle of the clip.



5.3 Model Choice

When choosing a model, we are mainly interested in the power of generalization of the classifier. The main threat to this is the phenomenon called overfitting. Overfitting happens when the classifier essentially becomes so tuned to the training set that it can classify it perfectly, but has poor accuracy with other instances. In order to analyze different models while avoiding overfitting, we utilize a technique called cross-validation (KOHAVI et al., 1995).

This technique partitions the instances into n disjoint groups. The classifier is trained with n-1 of these groups, and, after the training, it is tested with the group that wasn't used for training. The process is repeated until all groups have been used as the test in one run. If

in any set of training data the classifier suffers from overfitting, it will likely be detected when testing, because in the test set there are instances that the classifier has not seen during training, and the classifier will have a poor result.
The objective at this stage is merely choosing the model that experimentally had the best performance on the data that was available. After the best model has been selected, the training would then done with all the instances in order to produce the final classifier.

5.4 Implementation

The machine learning flow was implemented using Python with the library SciKit-learn (PEDREGOSA et al., 2011). Three different models were considered: support vector machines (SVM) (CORTES, 1995), random forests (RF) (BREIMAN, 2001) and neural network (NN) (HAYKIN, 1994).
An SVM maps the input vector to a very high-dimensional feature space, where linear decision surfaces are constructed (CORTES, 1995). In this specific application, a generalized model is used to classify into multiple classes, as opposed to the original implementation that performs binary classification.
An RF is a combination of multiple decision trees, where each tree performs its prediction based on a sampling of the original attributes. Given the small number of attributes, we chose the number of trees in the RF to be ten.
Finally, a NN is a network of simple nodes organized in layers, where each layer is completely connected to the previous and next layer, although the weights of the connections are generally different. The network computes its classification by propagating the data through each layer. At every propagation, the nodes sums all the weighted inputs from the previous layer, applies a sigmoid function and outputs to the next layer. In this experiment the neural network had five hidden layers, each having two nodes.
The clips used for training and test were extracted from testcase3, running on area mode with a constraint of 0.85. The number of groups for cross-validation was set to 5. Furthermore, the features extracted were all normalized with the objective of making all features having the same initial importance regardless of magnitude.
Since the samples were obtained from the benchmarks, many of the clips are equal. It can, therefore, happen that one instance that is used in training appears also in the test set. To better compare the models given the particularity of this domain, the flow was executed once again without any clip repetitions. This is the reason the cross-validation

used 5 groups instead of the usual 10 because of the number of samples in the second dataset, which contained too few samples

5.5 Results

Table 5.1 contains the results of the test for each model. These models were trained with a total of 5068 instances. Based on this test, we can conclude that the random forest is the best model of the three, having the very high mean accuracy of 99.51%. Of course, the number of instances is important, and a database with enough examples would allow for a model with even higher accuracy.

Table 5.1: Results for each model on the dataset with all instances.

	SVM	RF	NN
Mean Accuracy	94.81%	99.51%	82.67%
Variance	0.03%	0.00002%	0.03%

The second run, where no repeated instances are allowed, the results were different. There are 91 unique clips in the 5060 from the dataset. As it can be seen in table 5.2, the accuracy is considerably smaller, and the variance is much higher. This can be attributed to the considerably smaller number of instances used to train the models and the fact that there is no chance of an instance appearing in the training set and test set. Cross-validation with 5 groups was used for this test as well.

Table 5.2: Results for each model on the dataset with no repeated instances.

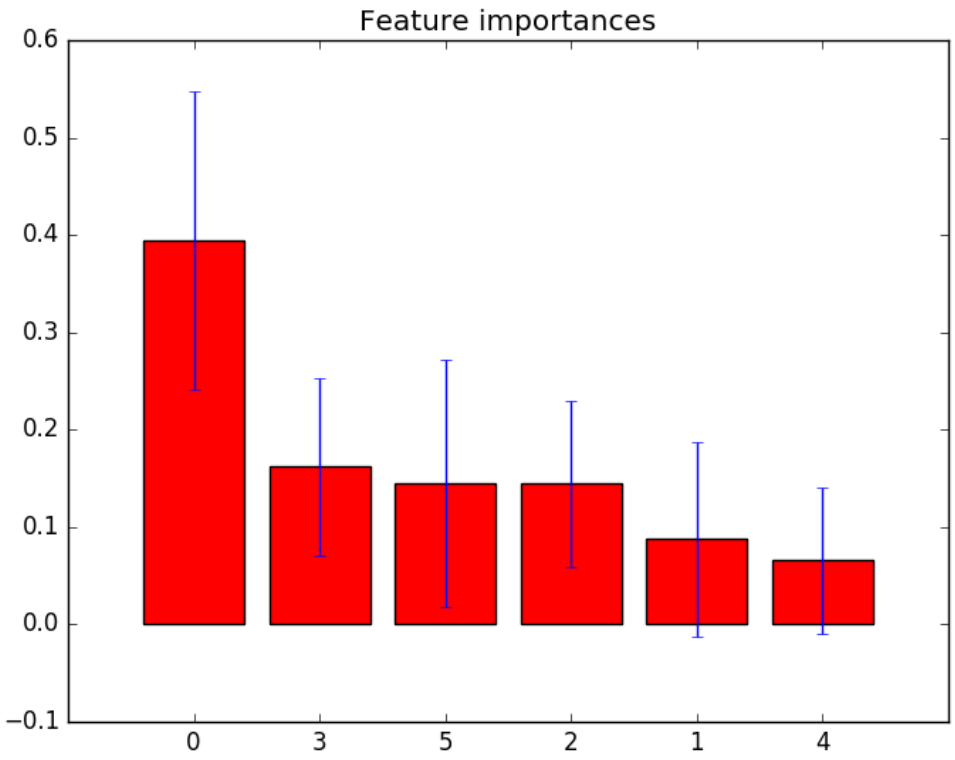
	SVM	RF	NN
Mean Accuracy	53%	63%	41%
Variance	11.36%	14.76%	11.24%

Both tests indicate that the best model for this domain is random forest, obtaining a mean accuracy of 99.51% with negligible variance in the test with repeated clips and 63% mean accuracy with 14.76% of variance with no repetitions.

The feature importance has been extracted from the random forest generated for both datasets. The results for the dataset with repeated instances is shown in figure 5.3 and the results for the dataset with no repetition are in figure 5.4. In both figures, the x

axis maps to the feature index in the same order as stated in Section 5.2, and the y-axis contains the feature importance in percent. Feature 0 is the material area, feature 1 is the number of rectangles, feature 2 and 3 are area comparisons with horizontal and vertical stripped clips respectively, and features 4 and 5 are edge comparisons with horizontal and vertical stripped clips respectively.

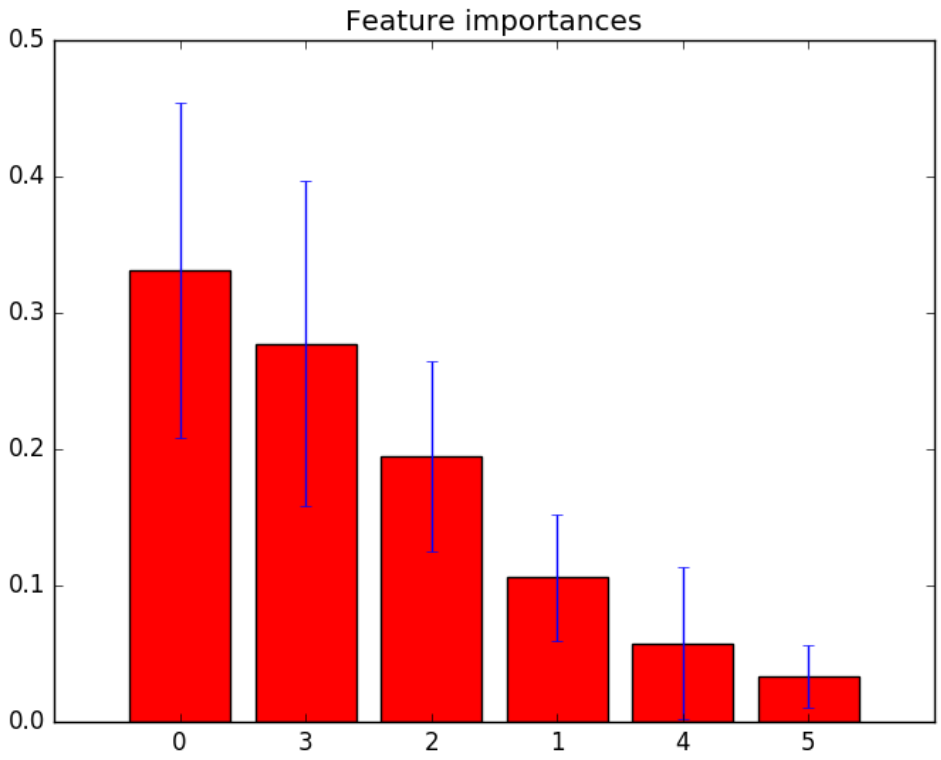
Figure 5.3: Feature importance for the dataset with repetition.



From both figures it can be inferred that the total material area of the clips is the most important feature, scoring 0.39 and 0.33 in the datasets with repetition and no repetition respectively. It can also be observed that in the dataset without repetition, the comparison with respect to area with both the vertical and horizontal stripped clips provide substantial information for the model, with 0.27 and 0.19 respectively. In the dataset with repetitions, the two comparisons score 0.16 for the vertical and 0.14 for ly stripped clips, which is not as high as it scored in the dataset with no repetitions but it is still non-negligible.

For both datasets, the number of rectangles and the edge comparisons scored low. For the no repetition, horizontal (vertical) stripped clip edge comparisons scored 0.05 (0.03), and the number of rectangles scored 0.10. In the dataset with repetitions, the horizontal (vertical) stripped clip edge comparisons scored 0.06 and 0.14, and the number of rectangles

Figure 5.4: Feature importance for the dataset without repetition.



scored 0.08. Even though the vertical stripped clip score is reasonably high, the suspicion of bias on the dataset could contribute towards deviation of this value. Therefore, it can be concluded that these three features add little information to the model and their removal should be considered.

6 CONCLUSIONS

The integration of the complex polygon was succesful. With it, the two benchmarks that contain complex polygons could be used to evaluate our tool's results. It was possible to validate the algorithm's correctness and identify problems with the scalability, mainly with the clip extraction. As for the quality of the result, the lack of a robust choice of representative clips caused the edge mode runs to produce unsatisfactory results. In order to improve the quality, a better choice of representative looks very promising study. The algorithm used for clip extraction has a complexity of $\mathcal{O}(n * m)$, where n is the number of rectangles in the layout and m is the number of markers. This is because all clips are compared with every rectangle, regardless of how far apart both are. With a better algorithm, only the polygons that have a chance of intersecting with the clip boundaries could be considered, speeding up the bigger benchmarks. Also, the clustering algorithm has a worst case complexity of $\mathcal{O}(n^2)$, where n is the number of clips.This makes the very large benchmark runs to be very slow. A different strategy that sorts the clips first could be employed, potentially reducing the run times for the larger benchmarks.

The machine learning flow's results, presented in Section 5.5 were satisfactory. The results reinforce the idea that this problem is compatible with a machine learning flow. More features could be tried in the future, as well as other models. As a future work, a further application of the classifier could be trained with a large benchmark such as testcase4, and then a whole smaller benchmark such as testcase2 would have its layout hotspots incrementally classified in the clusters obtained from testcase4, but using the machine learning flow to classify. This experiment is essentially the complete implementation of the incremental flow, whereas in this work the viability of such incremental flow was studied and validated.

REFERENCES

BISHOP, C. M.; PARK, W. S. Pattern recognition and machine learning, 2006. **Korean Society of Civil Engineers**, v. 60, n. 1, p. 78–78, 2012.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

CHANG, W.-C. et al. iclaire: A fast and general layout pattern classification algorithm. In: IEEE. **Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE**. [S.l.], 2017. p. 1–6.

CHEN, K.-J. et al. Minimizing cluster number with clip shifting in hotspot pattern classification. In: IEEE. **Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE**. [S.l.], 2017. p. 1–6.

CORTES, C. Support-vector network. **Machine learning**, v. 20, p. 1–25, 1995.

HAYKIN, S. **Neural networks: a comprehensive foundation**. [S.l.]: Prentice Hall PTR, 1994.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing In Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

KOHAVI, R. et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: STANFORD, CA. **Ijcai**. [S.l.], 1995. v. 14, n. 2, p. 1137–1145.

LYSENKO, M. **Rectangle Decomposition**. [S.l.]: GitHub, 2014. <https://github.com/mikolalysenko/rectangle-decomposition>.

NAHAR, S.; SAHNI, S. Fast algorithm for polygon decomposition. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 7, n. 4, p. 473–483, 1988.

OHTSUKI, T. Minimum dissection of rectilinear regions. In: **Proc. IEEE International Symposium on Circuits and Systems, Rome**. [S.l.: s.n.], 1982. p. 1210–1213.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

SHIN, M.; LEE, J.-H. Cnn based lithography hotspot detection. **International Journal of Fuzzy Logic and Intelligent Systems**, Korean Institute of Intelligent Systems, v. 16, n. 3, p. 208–215, 2016.

TOPALOGLU, R. O. Iccad-2016 cad contest in pattern classification for integrated circuit design space analysis and benchmark suite. In: IEEE. **Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on**. [S.l.], 2016. p. 1–4.

WU, S.-Y.; SAHNI, S. Fast algorithms to partition simple rectilinear polygons. **VLSI Design**, Hindawi Publishing Corporation, v. 1, n. 3, p. 193–215, 1994.

Pattern Classification for Layout Hotspots

André Saldanha Oliveira, Ricardo Augusto da Luz Reis

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{andre.oliveira, reis}@inf.ufrgs.br

Abstract. *This work's objective is to introduce the chosen subject for my graduation thesis. The chosen topic is within the area of physical verification, specifically classification of layout hotspots. We implemented a greedy algorithm for hotspot clustering that is competitive with the state of the art algorithms. Furthermore, we will discuss about the possibility of implementing a machine learning flow to enable incremental hotspot classification. This work's results are validated using the ICCAD 2016 contest's benchmark suite, and compared to state of the art works that published results obtained in the same benchmarks.*

Glossary

clip set of extracted polygons in a specific area of the layout, typically representing a hotspot..

DRC Design Rule Check.

hotspot region of the layout where an error detection tool has detected a problem. For this work, we assume a hotspot if a region with poor lithographic printability..

layout representation of an integrated circuit in terms of geometric shapes stacked in layers made of different materials..

1. Introduction

The widening gap of the sub-wavelength lithography in advanced process technology causes unintended distortions on the shape of the printed layout patterns[ITR]. Thus, there is an increasing interest in the circuit synthesis field for manufacturability-driven design and physical verification. The urge for scalability and faster run times for tools keeps increasing, while the size of the designs are getting larger, which presents an even more challenging task when it comes to keep the tools fast. The heuristics applied by most of the physical synthesis tools during the flow often disregard some constraints in order to achieve a good result within reasonable time. Furthermore, the advanced fabrication technology and the process variation brought with it new challenges. Lithography hotspots, for example, are a set of layout patters with poor printability, even though it passes DRC[Chen et al. 2017].

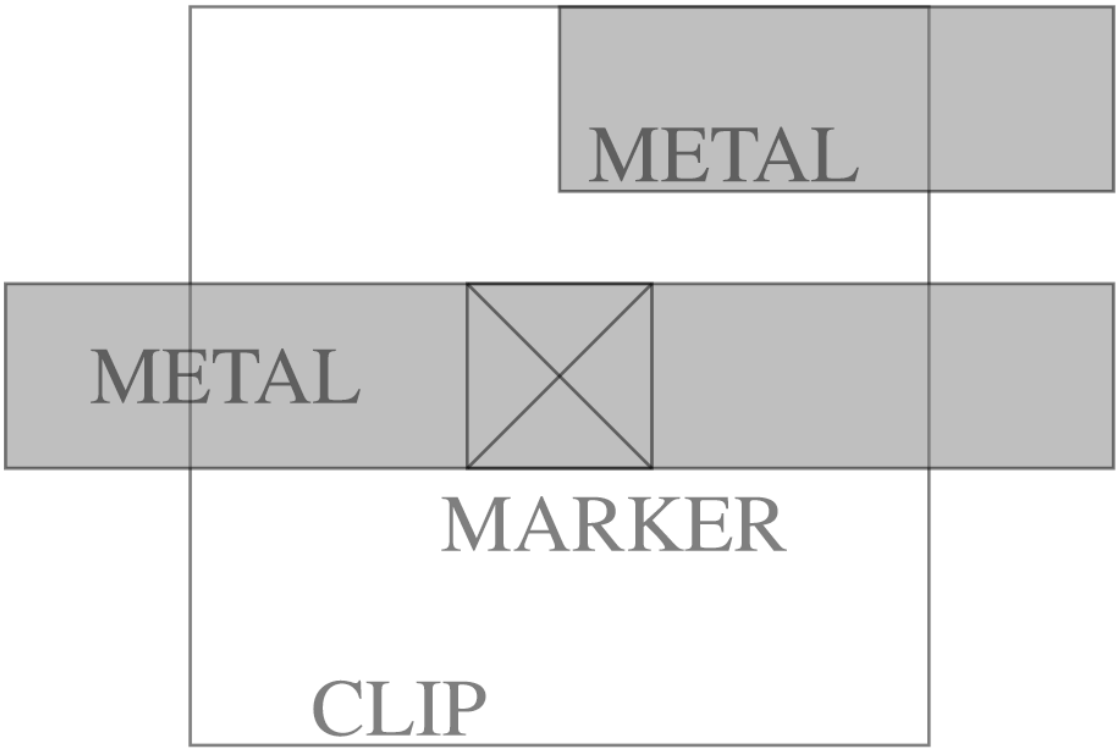
The clustering of hotspots - which in this scope are assumed to be lithography hotspots - has the potential of accelerating other error correction and classification flows. By clustering together clips based on their geometric similarity, an error correction flow

could merely analyze one clip of each cluster, so called the representative, cluster and apply any changes to all of the cluster members, resulting in a faster overall flow.
What this work proposes is a flow that, given layout files, generates clusters respecting the specified constraints. Alongside the clustering flow, we also propose a machine learning approach for incremental classification. After the initial solution, new hotspots can be classified into existing groups using a classifier trained using the data from previous executions of the flow.

2. Problem Formulation

As previously mentioned, the hotspot detection is crucial in physical verification. We follow the problem specification from 2016 CAD contest at ICCAD for pattern classification[Topaloglu 2016]. The contest provides as input a circuit layout. The layout, following the contest definition, has two relevant layers. The first layer has the metal polygons. The last layer contains polygons that represent markers, but have no physical meaning. The markers are generally small polygons on which a clip will be centered. The clip can be centered in any point inside the marker. Figure 1 is an example of this configuration.

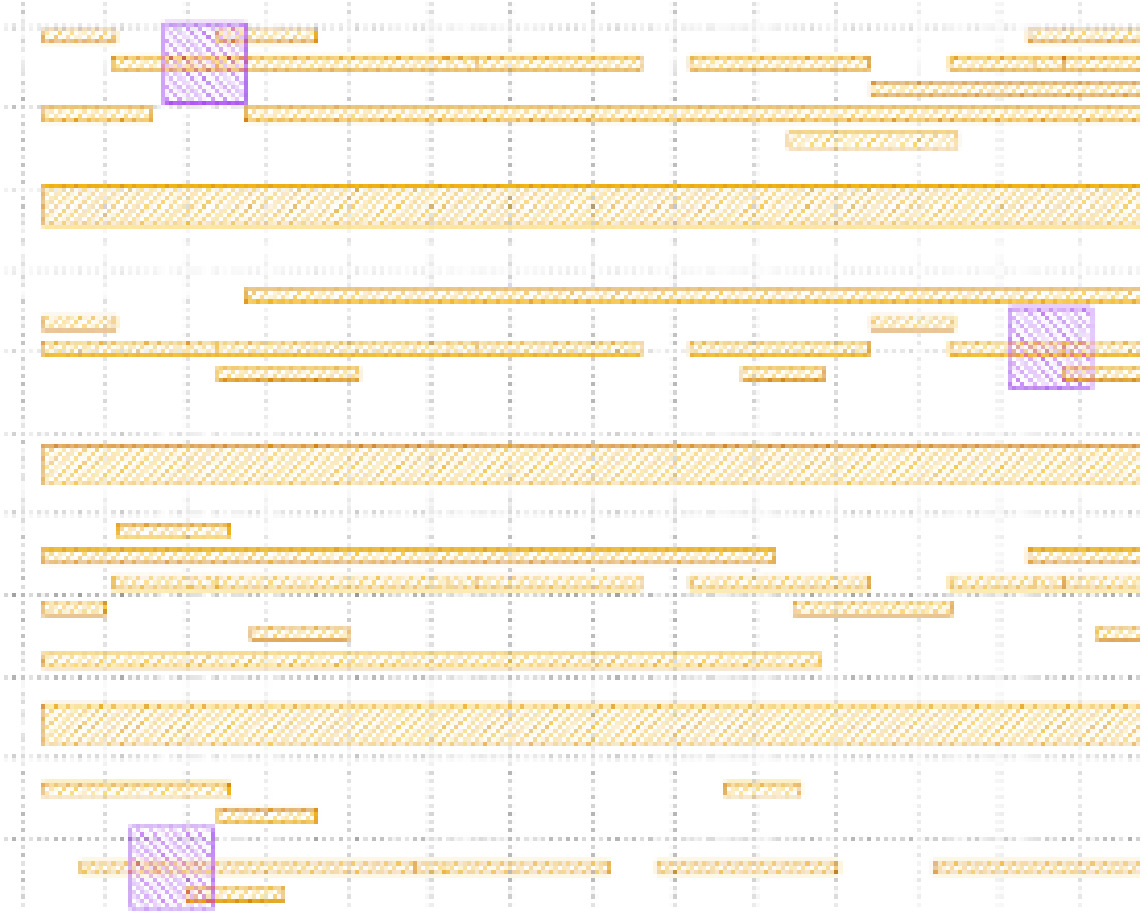
Figure 1. Example picture of a clip configuration in a layout.



The program input also specifies the width and height of the clip. Given a marker, one can choose a point contained in it and generate a clip centered in such point with the given dimensions. Figure 2 shows this exact process. The white canvas represents lack of any material. The yellow polygons represent metal, and the purple polygons are the

clips. Note that the clips are centered in the markers. The main objective is to provide a reduced set of representative layout clips contained on these markers.

Figure 2. Example picture of a layout with clips.



To perform the pattern classification two parameters are selected by the user: (A) Area match constraint that selects the clustering by area constraint and (B) Edge displacement constraint that performs the clustering by edge constraint. The objective of clustering is to provide sets of clips (clusters) that resemble each other according to these parameters.

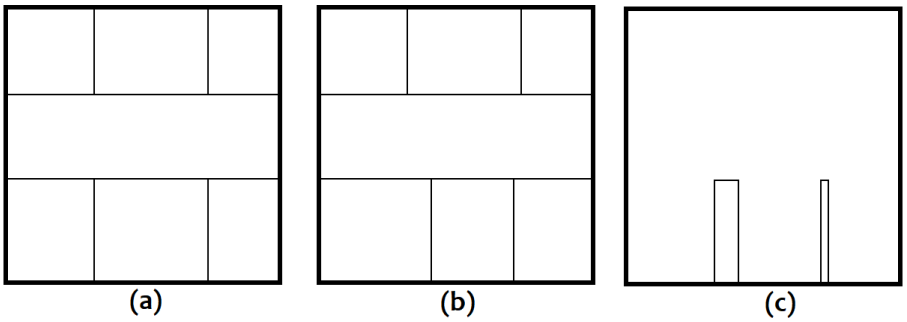
2.1. Area Constrained Clustering

In area constrained clustering (ACC) mode, a topological classification is performed based on the area match constraint parameter α. This parameter defines the maximum area match percentage. The overlap area between any clip of a cluster (A) and the representative clip of a cluster (B) must satisfy this parameter, as shown in the equation 1, bellow. In Figure 3 a graphical example is provided. In the figure, there are two clips (a) and (b) that need to be compared in terms of area. The result of the geometric XOR applied upon them is represented in the resulting clip (c).

$$\frac{[Area(XOR(A,B))]}{w*h} \leq (1-\alpha) \quad (1)$$

The Area() function gives the total area of the polygons and the XOR gives the geometric difference between the clips A and B. And (w, h) are the dimensions of the clip.

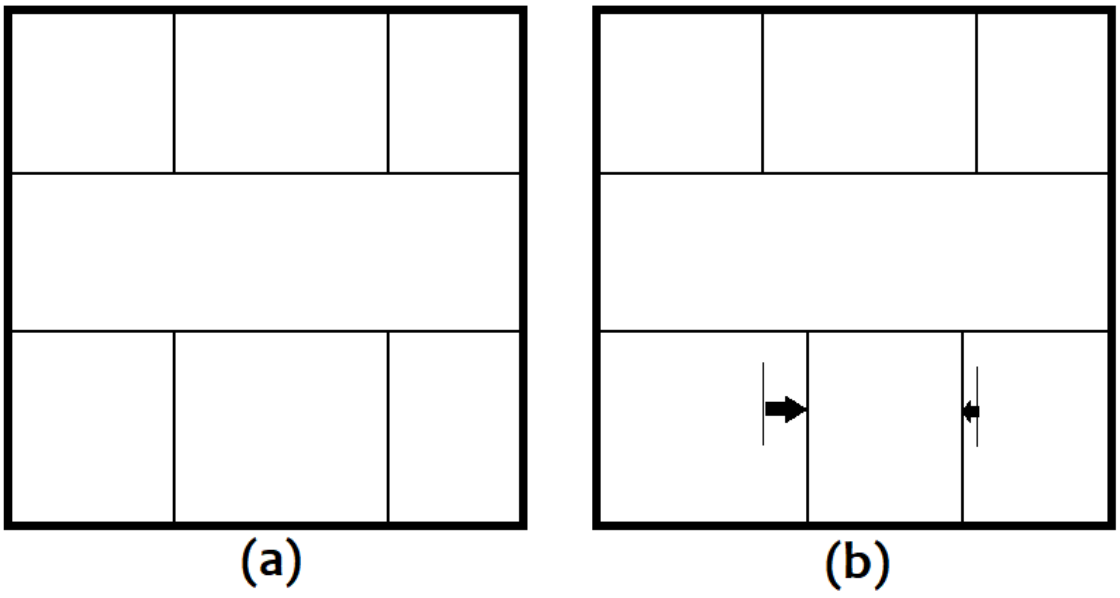
Figure 3. Geometric XOR example. (a) and (b) are two clips, and (c) is the result of the geometric XOR between (a) and (b)



2.2. Edge Constrained Clustering

The edge constrained clustering (ECC) mode receives a parameter e in nanometers that indicates how much the edge will be shifted, the shifting can be in both directions: inward and outward. This parameter e defines the tolerance that the clip pattern can shift. Figure 4 shows an example of the edge being shifted. Multiple edges can shift by different amounts since it is limited by e value.
Edges only shift in orthogonal projections and the representative clip of a cluster after the shifting becomes another clip of the same cluster.

Figure 4. Edge displacement example. (a) and (b) are two clips. (b) differs to (a) in edge displacement, showed by the arrows.



3. Related Work

As stated in section 2, this work is related to the 2016 CAD contest at ICCAD. There have been more works published that were developed based on the same contest, these being [Chen et al. 2017] and [Chang et al. 2017] .

[Chen et al. 2017]'s contributions are very significative. In order to save time, we consider the center of the marker as the center of the clip, whereas [Chen et al. 2017] selects a different point within the marker. The candidates generated are more effective in generating less clusters, results which we will analyze later. Furthermore, they describe an encoding method to create a string representation of a clip, which could prove useful for machine learning.

[Chang et al. 2017]'s main contributions are a technique of clip representation based on topology and density and the proposal of a two-level clustering, where any identical clips are merged together before considering any comparisons. This idea could be applied to our tool very easily, and should generate even better runtimes.

[Yu et al. 2015] consists in a very good base for attribute extraction for machine learning. The authors propose a critical feature extraction and topological classification that, when combined, allows for their machine learning flow to achieve high accuracy. However, the objective of their work is not clustering, but rather identify false detections. For this reason, we do not compare to their results.

[Yang et al. 2017] contributed with an improved distance metric between two clips, applied to hotspot classification in terms of type. For this reason, we do not compare to their results. The accuracy achieved by their tool is generally better than the works it compares itself to. However, their runtime is sometimes too large, especially for smaller benchmarks. Nonetheless, given that it does scale better than the other works it cites, it shall be considered for future implementations.

We can compare ourselves with the related work that has published results of runs on the contest benchmarks. Table 1 contains the information pertinent to the benchmarks. The first column has the benchmark names, the second column has the total number of polygons in that benchmark and the third column has the number of markers, which will become the center of clips.

Table 1. Benchmarks and their characteristics.

Benchmark	Polygons	Markers
testcase1	77	16
testcase2	845	200

Currently, our tool works on benchmarks 1 and 2 of the contest. In these benchmarks, our results are quite competitive with [Chen et al. 2017] and [Chang et al. 2017],

as well as the contest winners. In section 6 we will compare the current results with the related work and the contest winner.

4. Clustering Flow

In this section, we detail our approach to cluster the clips together. Figure 4 shows the pattern classification flow. The markers are provided as an input and since they are not points, but shapes, the clip centering can be arranged in order to reduce the number of clusters. Here, given the layout in GDS file containing the markers and the clip size, the clips are centered in the midpoint and for each marker a clip is created. The extraction of the clips uses the points of the shapes instead of a grid matrix. Our flow targets a fast and accurate solution. But there is space for improvements in scalability and complex shape partitioning.

In order to achieve competitive runtime, we implemented a greedy algorithm to cluster the clips together. The list of clips are shuffled, then the first unclustered clip is selected as a representative of a new cluster. This representative is compared with every other unclustered clip in the list, and if there is a rotation or mirroring of the second clip that satisfies the constraint the clip is added to the cluster. This is repeated until every clip is clustered, and the list of clusters is returned. We also compare the clips total material area before anything else, because if the material area of the two clips is too different there is no reason to compare different configurations because they will not be compatible regardless of the rotation. The clustering pseudo-algorithm 1 shows the algorithm used for clustering. The function needs to receive a list of the previously extracted clips, the constraint and the operation mode. For example, it could run on area mode with a constraint of 0.95, or 95%, or edge mode with a 2nm constraint.

The algorithm first shuffles the list of clips to obtain a new order. It also initializes an empty list of clusters to be populated during the algorithm. These two initialization steps can be seen in the lines 2 and 3 of the algorithm.

In line 4, the algorithm starts iterating through the clips. For each clip it iterates through, if the clip is already part of a cluster it is skipped, as seen in lines 5 and 6. If not, a new cluster is created, the current clip is clustered in it and is set as the cluster's representative, all of which is done in lines 8 to 10.

From line 11 to 18, all clips starting from the next in the list of clips are compared to the representative. The function compare_clips() in line 12 performs the comparison. It keeps the first clip constant and generates all possible arrangements applying rotations and mirrorings on the second clip and then performing the comparison, returning the smallest difference possible. In area mode, a geometric XOR is applied between the first clip and all the arrangements of the second clip, and the smallest difference is returned. In the edge mode, the biggest edge difference of the best rotation is returned. If the difference is smaller than the constraint requires, the second clip is added to the cluster.

Finally, the cluster being generated, it is added to the list of clusters. After all the clips have been iterated through, the list of clusters is returned.

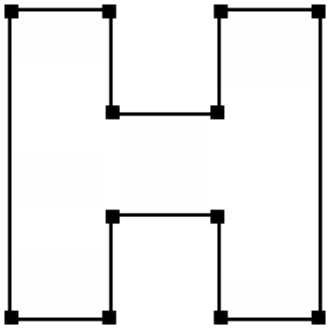
4.1. Complex Polygons

Some of the shapes in the layout are complex polygons. A complex polygon in this context is any polygon that isn't a rectangle. Figure 5 is an example of a complex polygon.

Algorithm 1 Greedy clustering algorithm.

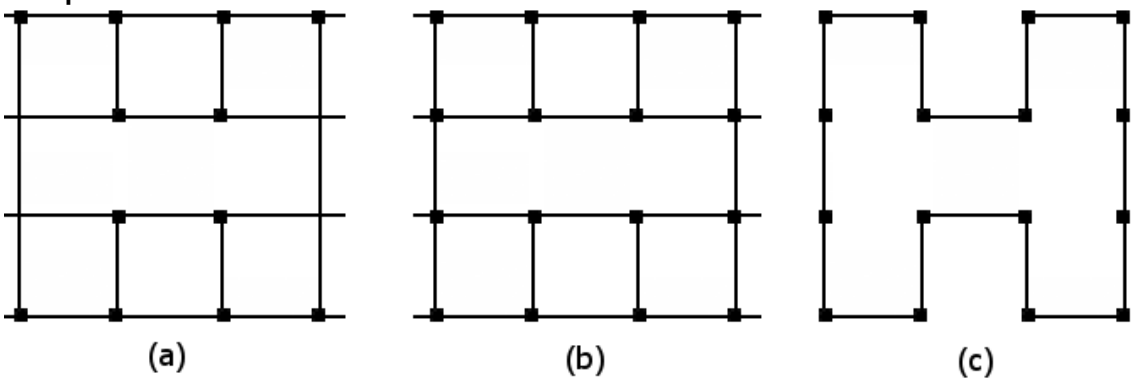
```
1:  procedure CLUSTER(list_of_clips, constraint, mode)
2:      clips ← shuffle(list_of_clips)
3:      clusters ← {}
4:      for all clip in clips do
5:          if clip.get_clustered() then
6:              continue
7:          else
8:              new_cluster ← create_cluster()
9:              new_cluster.representative ← clip
10:             clip.set_clustered()
11:             for all other_clip in clips do
12:                 difference ← compare_clips(clip, other_clip, mode)
13:                 if difference > constraint then
14:                     continue
15:                 else
16:                     new_cluster.add_clip(other_clip)
17:                     other_clip.set_clustered()
18:                 end if
19:             end for
20:             clusters.add_cluster(new_cluster)
21:         end if
22:     end for
23:     return clusters
24: end procedure
```

Figure 5. Example of a complex polygon.



In order go make faster comparisons, the algorithm requires that any polygon that isn't a rectangle is divided into a set of rectangles. The current algorithm we use for this task is based on the concavity of each vertex. First, all horizontal points become horizontal lines, as seen in figure 6 (a). Then, new points are added in the intersection of these horizontal lines with vertical lines, just like figure 6 (b) shows. The final result with the new dummy points is shown in figure 6 (c).

Figure 6. First three stages of polygon division. (a) represents the horizontal line creation. (b) represents the new points being drawn in the intersections of vertical and horizontal lines. (c) shows the final image with the dummy new points.



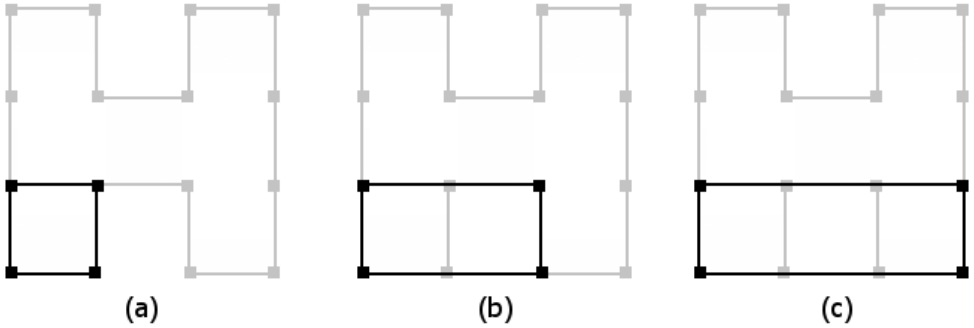
Finally, the division process starts. All points starting from the bottom left, are compared in groups of four. Whenever any group that generates a rectangle is found, a candidate subpolygon is generated. This candidate goes through a process to determine if it's a valid part of the original polygon.

Given this new representation of the complex polygon with the dummy points, the algorithm generates candidate rectangles, that may or may not be part of the final set of rectangles. Rectangles initially accepted may be removed because they are contained entirely by another valid rectangle. The rules are as follow:

1. If the polygon is entirely contained by another polygon it is discarded as redundant.
2. If the candidate doesn't pass the concavity test it is discarded because it covers a region that isn't present on the original polygon, and the candidate is marked as invalid.
3. If the candidate contains any invalid polygon it is discarded.
4. The candidate is included in the set of polygons.

Figure 7 (a) shows a first candidate, which is accepted. Figure 7 (b) shows a second candidate that is not accepted, because the concavity of the two corners further to the right aren't compatible with rectangle. Figure 7 (c) shows a rectangle that would have been accepted based on concavity, but it is discarded because it contains a part of the rectangle discarded in Figure 7 (b).

Figure 7. Candidate rectangles generation. (a) represents a valid rectangle candidate that is accepted. (b) represents an invalid rectangle that is discarded because it doesn't belong to the shape. (c) shows a candidate thas is discarded because it contains part of the rectngle discarded on (b).



After the shape is split, any intersection between any number of polygons is treated. Figure 8 shows a case where two valid polygons share some area. In this case, one of the n intersecting polygons keeps this region and all the other are trimmed. This is repeated until no polygons are intersecting. The result of this is a set of rectangles that are equivalent to the complex polygon.

5. Machine Learning Flow

After the initial solution given by the clustering flow has been achieved, we intent on using a machine learning flow for incremental classification. Figure 9 shows the clustering flow. The input is the layout file, which is parsed int he first state. The second state, given the layout, extracts the clips. The final state runs the clustering algorithm and outputs the clusters. All these steps have been covered previously. What the machine learning flow can accomplish is an incremental flow after the initial clusters have been generated.

Usually, after a new run, the new clips would be compared to all the representatives and, if the representative isn't well chosen, one may need to create more clusters. What we propose is training a classifier with the data acquired from the clusters and the

Figure 8. Example of two rectangles overlapping with each other.

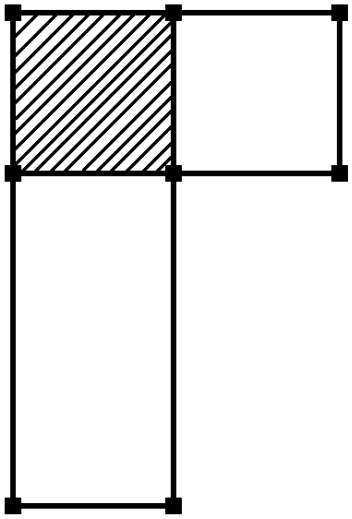
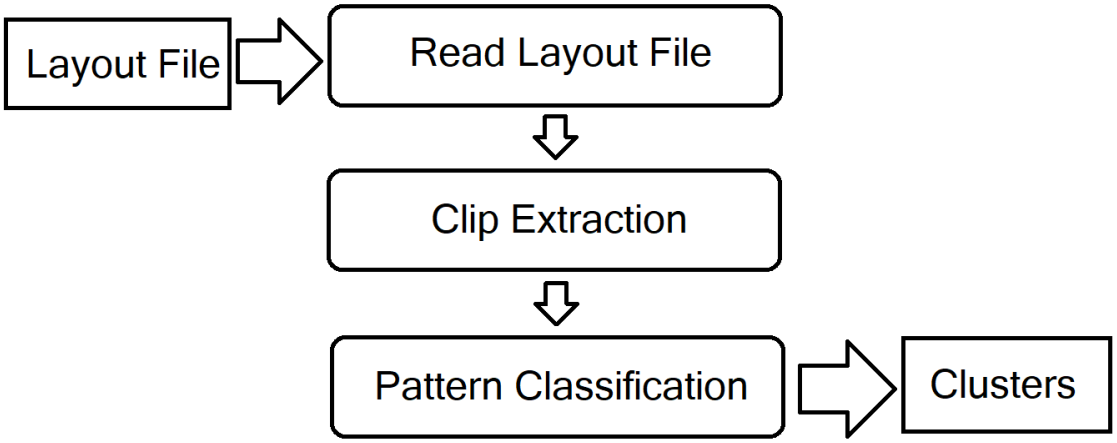


Figure 9. clustering flow.

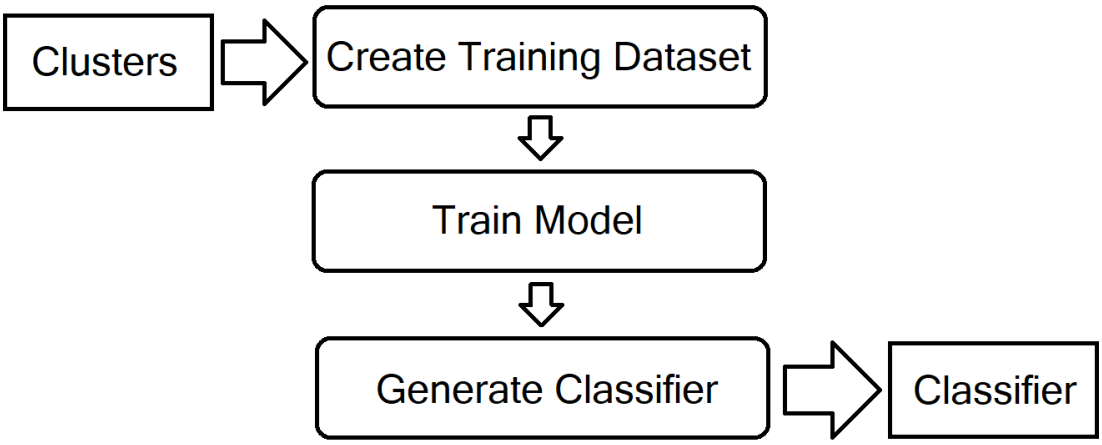


clips they contain and make a classifier capable of assigning new clips to existing clusters in a very short time. The drawback is the training time of the classifier. However, once a good classifier has been trained, the complete flow never needs to be run again. Furthermore, depending on the attributes of the clips given to the classifier, it may be no longer necessary to divide complex polygons into rectangles.

The machine learning flow is illustrated briefly in figure 10. The initial state is fed with the clusters generated by the clustering flow. Each observation is a set of attributes, for example, the number of polygons, and the expected result. Some of the observations aren't used in training, but rather on testing the model. This is to check if the model is suffering of overfitting, where the classifier essentially memorizes the training set but isn't useful at classifying new observations.

Once the classifier is properly trained and tested, a classifier is generated, and it can be used to classify any new clip into a cluster.

Figure 10. Machine learning flow.



6. Conclusion

The developed tool is compared with the related work discussed in section 3 and the contest winners. The configurations of the runs are showed in the first column. For example, testcase1ap95 means the benchmark used was the testcase1, and it was run with an area constraint mode of 95%, and testcase2e4 means the benchmark used was the testcase2, and it was run with an edge constraint mode of 4 nanometers.

Table 2 shows the results in terms of runtime. In all cases of benchmark 1, our tool outperforms every other tool, except iClaire and only for the case of no constraint. As for testcase2, our tool is slightly slower than top 1, except for edge constriant mode, where it performs slower. When compared to iClaire, our tool is outperformed slightly in all configurations. The tool is faster than Chen for every configuration.

Table 2. Runtime Results (ms)

	TOP 1	Chen17[Chen et al. 2017]	iClaire[Chang et al. 2017]	Ours
testcase1	5	10	1	3
testcase1ap95	5	10	5	3
testcase1e4	5	30	5	3
testcase2	10	30	4	11
testcase2ap95	12	60	11	13
testcase2ap90	11	-	11	13
testcase2e4	9	200	11	19

Tables 3 and 4 showcase the quality of the results rather than the runtime. On table 3, our results are almost equal to the top 1 in all configurations, we generating one less cluster on testcase2 with area 95% and their tool generating one less on testcase 2 with edge 4nm. The results are also very similar to iClaire's, generating the same results except for testcase1 95% area, testcase2 95% area and testcase2 90% area, where it generates one less cluster. Chen generates less clusters for most configurations, and

clearly outperforms the rest of the tools.

Table 3. Cluster Count Results

	TOP 1	Chen17[Chen et al. 2017]	iClaire[Chang et al. 2017]	Ours
testcase1	8	8	8	8
testcase1ap95	4	3	3	4
testcase1e4	5	5	5	5
testcase2	26	20	26	26
testcase2ap95	13	5	11	12
testcase2ap90	7	-	7	8
testcase2e4	18	13	18	18

Finally, on table 4, we can see the number of clips that were clustered in the largest cluster. All tools generated a very consistent result, with small variations. The most notable variations are iClaire generating a cluster with 9 clips for testcase1 area 95% while the second best had 6. Also, top 2 managed to generate a cluster with 138 clips while every other tool had 104.

Table 4. Maximum Cluster Size Results

	TOP 1	TOP 2	TOP 3	iClaire[Chang et al. 2017]	Ours
testcase1	5	5	5	5	5
testcase1ap95	6	5	6	9	6
testcase1e4	5	5	5	5	5
testcase2	104	104	104	104	104
testcase2ap95	106	104	106	106	106
testcase2ap90	114	104	111	112	112
testcase2e4	104	138	104	104	104

Given the results in comparison to the related works and contest winners, we conclude that the results are quite promissing, We perform a competitive clustering in all configurations for the benchmarks stated.
For the final version of this work, we will study different approaches to the representative generation and polygon division.

References

International technology roadmap for semiconductors, available at http://www.itrs2.net/.

Chang, W.-C., Jiang, I. H.-R., Yu, Y.-T., and Liu, W.-F. (2017). iclaire: A fast and general layout pattern classification algorithm. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pages 1–6. IEEE.

Chen, K.-J., Chuang, Y.-K., Yu, B.-Y., and Fang, S.-Y. (2017). Minimizing cluster number with clip shifting in hotspot pattern classification. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pages 1–6. IEEE.

Topaloglu, R. O. (2016). Iccad-2016 cad contest in pattern classification for integrated circuit design space analysis and benchmark suite. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*, pages 1–4. IEEE.

Yang, F., Sinha, S., Chiang, C. C., Zeng, X., and Zhou, D. (2017). Improved tangent space-based distance metric for lithographic hotspot classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(9):1545–1556.

Yu, Y.-T., Lin, G.-H., Jiang, I. H.-R., and Chiang, C. (2015). Machine-learning-based hotspot detection using topological classification and critical feature extraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):460–470.