VIPER: A Versatile and Intuitive Pattern GenERator for Early Design Space Exploration

Gaurav Rajavendra Reddy, Mohammad-Mahdi Bidmeshki and Yiorgos Makris
gaurav.reddy@utdallas.edu, bidmeshki@utdallas.edu, yiorgos.makris@utdallas.edu
Department of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson, TX 75080, USA

Abstract*—Contemporary technology nodes exhibit high defectivity due to complex interactions between the process and certain layout topologies/patterns. Foundries identify such patterns during diagnosis, Scanning Electron Microscope (SEM) inspections, Failure Analysis (FA), etc., and create a database to restrict their presence in future designs. However, such a database can be generated only after fabricating a few products, hence making this process *reactive*. Ideally, foundries would prefer to have a *proactive* approach, where such sensitive patterns are available up-front during technology development. Thereby, they can build accurate Hotspot Detection models and offer a robust Product Design Kit (PDK) to even the earliest of customers, either by ensuring that the process is immune to such patterns or by including them in the Design For Manufacturability Guidelines (DFMGs). To enable this, Early Design Space Exploration (EDSE) can be performed, wherein an Electronic Design Automation (EDA) tool generates synthetic layout patterns. In this work, we introduce VIPER, a novel, controlled random walk-based pattern generation method, which not only generates realistic and Design Rule-clean layout patterns, but which also offers versatility so that the generated patterns can be intuitively customized to specific needs. To ensure that the generated patterns are representative of real designs, we data mine designs in previous technology nodes and we learn some of their typical characteristics. Effectiveness of the proposed method is contrasted against the state-of-the-art, commercially available EDA tool.

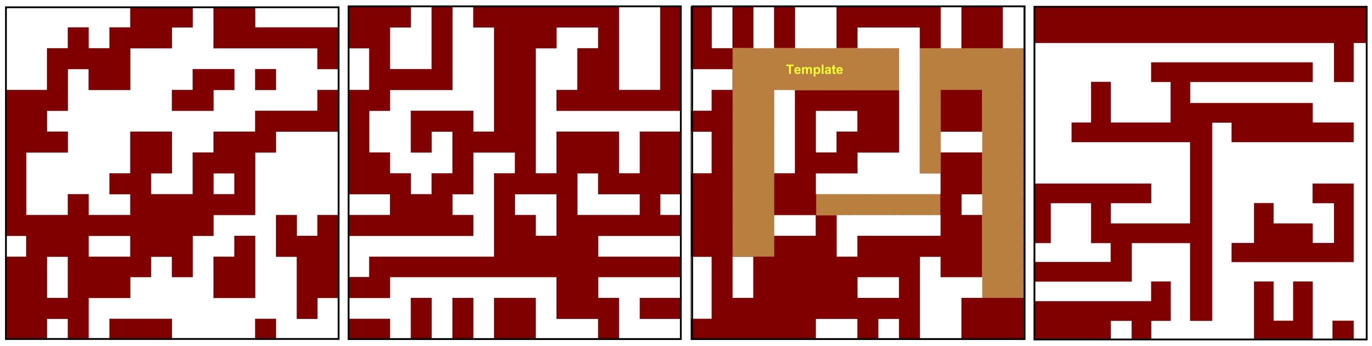
I. Introduction

Lithography is an extremely complex task and one of the major challenges during technology development in advanced process nodes. Construction of Optical Proximity Correction (OPC) keywords and lithography recipes begins in the early stages of technology development when only basic Design Rules (DRs) are defined and very little, if any, real layout content is available [1]. Therefore, during this process, lithographers must depend on traditional test structures and/or layout examples from previous technology nodes. As a result, the process/recipe is exposed to and optimized for only a small set of patterns among a vast array of options in the design space. When the technology goes into production and large scale designs are fabricated, however, the process is exposed to many new patterns which were never seen during the development phase. Interestingly, as shown in [2], [3], such previously unseen patterns continue to appear as more designs are fabricated throughout the lifetime of a process. Inevitably, this results in high systematic defectivity, as the process is not optimized for all patterns found in real designs. Unfortunately, once the process is in production, root cause analysis of all pattern-related defects and correction through process changes becomes expensive, time-consuming and, often, intractable. Therefore, common practice is to categorize defect-causing patterns as risky/hard-to-manufacture and restrict their use in future designs.

As an alternative, Early Design Space Exploration (EDSE) can be performed. In EDSE, a large number of synthetic patterns which resemble real layouts is generated up-front during technology development, using only the basic DRs [4]. A database of such patterns enables the foundry to develop a more robust technology node by: (i) using layout-like patterns during lithography recipe development and ensuring that the process is amenable to a large variety of patterns [1], (ii) designing and characterizing pattern-based test structures on silicon [5], (iii) formulating Pattern Matching (PM) rules or Design For Manufacturability Guidelines (DFMGs) around high-risk patterns and offering a robust Product Design Kit (PDK) to even the earliest of customers, and (iv) performing lithographic simulations on a large dataset and building accurate hotspot detection models to identify sensitive patterns in future designs [6], [7]. For EDSE to be effective, however, the synthetic layout pattern generation method and tool should exhibit several key attributes: First and foremost, it should be able to generate a wide variety of random patterns which explore and accurately reflect the entire design space. Second, resulting patterns should obey DRs and resemble real Integrated Circuit (IC) layout snippets. Third, it should enable the user to intuitively control various aspects of the generated patterns and accommodate specific needs, such as generation of constrained pattern types or patterns from certain design corners.

To the best of our knowledge, the State-Of-The-Art (SOTA) in this area is a commercially available Electronic Design Automation (EDA) tool (i.e., Mentor's Layout Schema Generator (LSG)), the use of which for synthetic layout pattern generation is described in [4]. This tool uses a set of 'unit patterns' and randomly places them on a grid of certain size, seeking to produce a realistic design. Some operating modes of this tool are fully automated or require minimal human involvement. In such cases, however, the produced patterns tend to be rather unrealistic and of limited utility to the EDSE process. Other modes, requiring significant amount of human effort, produce very realistic and DR-clean patterns. Nevertheless, despite using best modes, settings and practices, as prescribed by the tool vendor, it tends to generate patterns which only cover a limited portion of the design space, leaving

978-1-7281-4823-6/19/$31.00 ©2019 IEEE



(a) Random (b) Weighted (c) Template-based (d) Automated and realistic

Fig. 1: Sample patterns generated by the state-of-the-art commercial tool

a lot of room for improvement to better achieve the objectives of EDSE.

To this end, we propose VIPER, a novel, controlled random walk-based pattern generation method, which not only generates realistic patterns, but which also covers better the design space. In order to increase pattern realism, we data-mine designs from previous technology nodes and learn technology-independent design characteristics. VIPER is versatile and allows the user to intuitively vary every parameter of interest within a pattern, explore various corners of the design space and generate custom patterns for specific applications.

The rest of the paper is organized as follows. The SOTA and its limitations are discussed in Section II. The proposed methodology, its algorithmic details and its various features are introduced in Section III. Experimental results comparing the performance of VIPER against the SOTA are presented in Section IV and conclusions are drawn in Section V.

II. State-Of-The-Art And Its Limitations

The SOTA tool generates synthetic layout patterns by creating a grid of a certain size and populating it with ‘unit patterns’. These are customizable ‘snippets’ or ‘blocks’ whose side dimensions are equal to the minimal pitch for a given layer. The tool randomly places such unit patterns on the grid, seeking to produce a realistic layout pattern. However, as discussed in [1] and as shown in Figure 1a, such a random placement method seldom generates realistic patterns. Nevertheless, the tool features several ‘modes’ through which the user can attempt to generate more realistic patterns:

(i) Mode 1: In this mode, weights can be added to each unit pattern. The characteristics of the resulting patterns differ based on the specified weights. A typical pattern obtained using the weighted option is shown in Figure 1b.

(ii) Mode 2: In this mode, the user can define templates which predetermine certain areas of the pattern. The tool, then, generates random polygons around the predetermined area, as shown in Figure 1c. The authors of [8] suggest the use of complex patterns or Hotspot patterns obtained from previous technology nodes “as is”, as templates for generating more realistic designs. However, using such templates reduces the area available for the tool to place random patterns. Thereby, its ability to explore new design spaces is significantly restricted.

(iii) Mode 3: In this mode, as suggested by the authors of [6] and [9], design rules and various heuristics are transformed into a special $rule\_file$ in a format understandable by the tool, in order to guide the synthetic pattern generation process. However, producing such constraint files is extremely time-consuming and requires significant amount of human effort.

(iv) Mode 4: In this mode, which is fully automated and the most promising direction, realistic patterns are produced with minimal human involvement. The algorithm and heuristics that are likely used to generate patterns in this mode are proprietary and, therefore, unknown to us. The generated patterns appear to be realistic; however, as demonstrated in the experimental results (Section IV-A), they tend to only cover a limited portion of the design space. An example of such a pattern is shown in Figure 1d.

In short, the SOTA synthetic pattern generation tool offers various modes and options, enabling the user to make a trade-off between the amount of human-effort involved and the quality/realism of patterns generated. Even when operated in its most promising mode, however, it still offers limited coverage of the design space, thereby not exploring the full potential of EDSE in supporting robust early technology development.

III. Proposed Methodology (VIPER)

A standard cell-based IC layout is not a random arrangement of polygons; rather, every one of these polygons has a specific purpose. Moreover, besides being governed by design rules, they also exhibit an additional set of features which depend on their actual purpose in the design. Examples of such features include the following: (i) polygons supplying power to transistors are often vertical rectangles with one end always branching out of power rails and the other end terminating at source/drain, (ii) stacked-via enclosures are isolated rectangles with small dimensions, and (iii) source-to-drain connections and cell-to-cell connections occur through ‘paths’ which may have multiple turns and branches. Therefore, in order to generate realistic patterns, instead of generating random polygons, we should mimic the characteristics of real layouts and generate ‘paths’, ‘power rail branches’ etc., which are

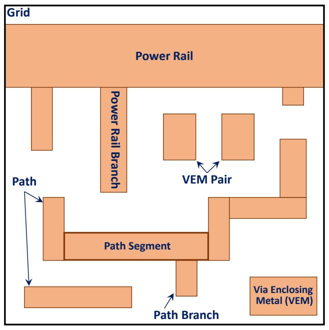


Fig. 2: Nomenclature

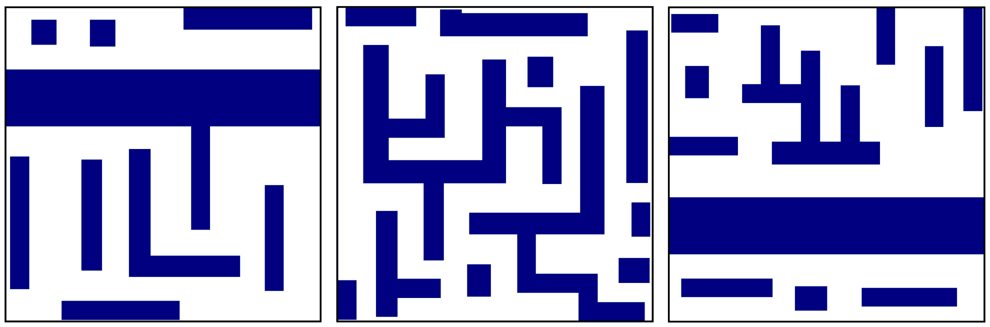


Fig. 3: Samples of patterns generated by VIPER

essentially 'polygons with a meaning'. To realize this, we propose a controlled random walk-based algorithm, where every aspect of a pattern is abstracted as a random walk performed under certain constraints. While the randomness in the proposed method ensures that design space exploration is not restricted, the constraints ensure that the 'meaning' of every polygon is preserved, thereby, making the resulting patterns realistic. The nomenclature, algorithmic details, and various aspects of the proposed method are explained in detail in the following sub-sections.

A. Nomenclature

Names assigned to the various features of a pattern are illustrated in Figure 2. The area wherein a pattern is created is called the *grid*. Wide metal lines running horizontally across the entire length of a pattern are termed *power rails*. Long metal lines, often containing multiple turns, are called *paths*. Parts of a path between turns are referred to as *segments*. Polygons branching out of power rails and paths are termed *power rail branches* and *path branches*, respectively. Small isolated rectangles are referred to as *Via Enclosing Metals (VEMs)* and VEMs which are horizontally aligned with each other are called *VEM pairs*.

B. Controlled Random Walk Algorithm

The required pattern size and the fundamental design rules, such as minimum width, space, notch, etc., are provided as inputs to the proposed pattern generation algorithm. Produced patterns may contain several of the features mentioned in Section III-A, but not every pattern must include all such features. The type and number of features to be included in a pattern is determined by sampling their corresponding Probability Density Functions (PDFs). The probabilistic nature of this method enables exploration of various corners in the design space and increases diversity of the produced patterns. Every feature is abstracted as a different type of a controlled random walk. More specifically, (i) a power rail is a random walk with only one horizontal step, (ii) a power-rail branch is a random walk with a single vertical step, with its initial point randomly chosen anywhere along an edge of the power rail, (iii) a path is implemented as a random walk with many continuous steps, where an anti-clockwise/clockwise right-angled turn is made at the end of every step, (iv) a path branch is similar to a path but its initial point always lies on an

```
def GenPattern (features) :
    Input: Pattern features, e.g. presence of power rail, power rail
           branch count, path count, VEM count, etc.
    Result: A random pattern with requested features
    /* Create a new grid                                      */
1   grid = Grid(features.gridSize)
2   if features.hasPowerRail == true:
        /* Generate power rail and its branches
           */
3       TakeRandomWalks (grid, type = PowerRail, count = 1)
4       TakeRandomWalks (grid, type = PowerRailBranch,
5       count = features.powerRailBranchCount)
    /* Generate paths                                         */
6   TakeRandomWalks (grid, type = Path,
7   count = features.pathCount)
    /* Generate path branches                                 */
8   TakeRandomWalks (grid, type = PathBranch,
9   count = features.pathBranchCount)
    /* ...                                                    */
    /* Similarly, all other pattern features
       are implemented                                        */
10  return grid
```

Algorithm 1: VIPER pattern generation

edge of a pre-existing path, and (v) VEMs and VEM pairs are implemented as single-step walks. A grid with a resolution of $1nm$ and dimensions equal to the pattern size is created and features are implemented through Algorithm 1.

At the start of every walk, depending on its type, many walk-specific parameters such as the number of steps, the width and length of every step, etc., are obtained by sampling their respective PDFs. To take the first step, a random location within the grid and a random direction are chosen. The area on the grid where the step is supposed to be taken is first checked for vacancy. The area surrounding the step is also checked in order to ensure that minimum spacing rules are not violated. If either of the checks fails, the walk is attempted in the opposite direction. If the first step is unsuccessful due to obstruction in both directions, a new initial point is obtained and the walk is restarted. If congestion is found in subsequent steps of a path, one or more steps are taken backwards, new distance values are obtained from the PDF, and the steps are reattempted in the same order. The process continues until either the walk is completed or a certain termination criterion is met. The detailed steps of this approach are shown in Algorithm 2, while examples of typical generated patterns are shown in Figure 3.

```
def TakeRandomWalks(grid, type, count):
    Input: Grid to create pattern on, walk type and count
    Result: Adds requested features to the grid
1   for $i$ in range(count):
        /* Get walk specific parameters such as
           step count, widths, distances, etc.
           by sampling corresponfing PDFs     */
2       walkParameters = SamplePDF ()
        /* get random initial point           */
3       initPoint = random ()
4       $step = 0$
5       $maxStep = 0$
6       while $step < stepCount$:
            /* get random direction            */
7           stepDirection = random ()
8           if step and surrounding is vacant:
9               take the walk and add it to the grid
10              $step+ = 1$
11              $maxStep = max(step, maxStep)$
12          else:
                /* try the opposite direction   */
13              stepDirection = DirOpposite(stepDirection)
14              if step and surrounding is vacant:
15                  take the walk and add it to the grid
16                  $step+ = 1$
17                  $maxStep = max(step, maxStep)$
18              else:
19                  if step == 0:
                        /* this is the first step
                           */
20                      repeat by going to line 3 until walk
                          completes, or termination criteria is
                          met
21                  else:
                        /* take a step back & try
                           again                  */
22                      remove last step from the grid
23                      $step = maxStep - 1$
24                      get a new distance
25                      repeat a few times by going to line 8
26                      if unsuccessful after multiple trials in
                          line 25:
                            /* take two steps back &
                               try again           */
27                          remove two last steps from the grid
28                          $step = maxStep - 2$
29                          get a new distance
30                          repeat by going to line 8 until walk
                              completes or termination criteria
                              is met
31  return
```

Algorithm 2: VIPER random walk procedure

While generating patterns with a large number of walks, some of the later walks may face increased congestion. Consequently, some parameters, such as step direction, may depend on the spatial orientation of polygons already placed by previous random walks. However, this does not reduce randomness because all the previous polygons were placed randomly and their spatial arrangement is different for every new pattern. To reduce congestion, walks are taken in a specific order. Power rails and their branches are created first. VEM pairs are created next, followed by Paths and their branches. Individual VEMs are placed last.

C. Data-Mining Previous Technology Nodes

With the introduction of every new technology node, the critical dimensions, standard cell sizes and layer pitch continue to shrink. However, fundamental characteristics of standard cells, such as the number of internal connections, type of connections, and Input/Output (IO) access methods have not changed significantly. Indeed, most of these attributes are governed by the principles of Complementary Metal Oxide Semiconductor (CMOS) logic design and are, therefore, technology independent.

To increase the 'realism' of the patterns generated for the current technology node, we data-mine designs from previous nodes and learn the statistics of various technology-independent features. Designs are decomposed into patterns of a certain size, which are then used for data-mining. Features-of-interest which we seek to learn include the number of paths in a pattern, the number of turns in a path, the length and width of individual segments, the number of branches on a path, the probability of a pattern having a power rail, the number of branches on a power rail, VEM counts, etc. The mined information is, subsequently, used to generate individual PDFs for the features-of-interest. To ensure that the generated statistics are technology-independent, various features such as widths, lengths, patterns sizes etc., are normalized to the corresponding layer pitch. Normalized PDFs are, then, used to sample various aspects of the patterns generated for the current technology node, as discussed in Section III-B.

We note that the information mined from previous technology nodes neither forces the algorithm to generate the 'same' patterns as before, nor constrains the algorithm from exploring new design spaces. The underlying reason for this is that the pattern generation process is still random; the mined information and learned feature distributions are only used as a starting point in order to guide our method towards generating realistic patterns. The user can always vary the mean and standard deviation of certain aspects to encourage the tool to generate patterns from various design corners or emphasize specific characteristics.

D. Versatility and Intuitiveness

Layout pattern generators are used not only to generate synthetic patterns which resemble real layouts, but also custom patterns targeting specific applications. For instance, while performing silicon validation for technology development, patterns from various design corners which push the process to its extremes are required. These could be patterns with a large number of corners, high-density, parallel-paths with varying widths and line-end positions, VEM clusters, low density patterns with isolated VEMs, etc. An effective synthetic pattern generation tool should be versatile enough to generate such custom patterns-of-interest.

A custom pattern generation process should also be intuitive and user-friendly. Unlike the SOTA, where the user can only control density and a few basic design rules, VIPER enables the user to generate full-custom patterns by directly controlling various parameters such as number of VEMs, corners, path

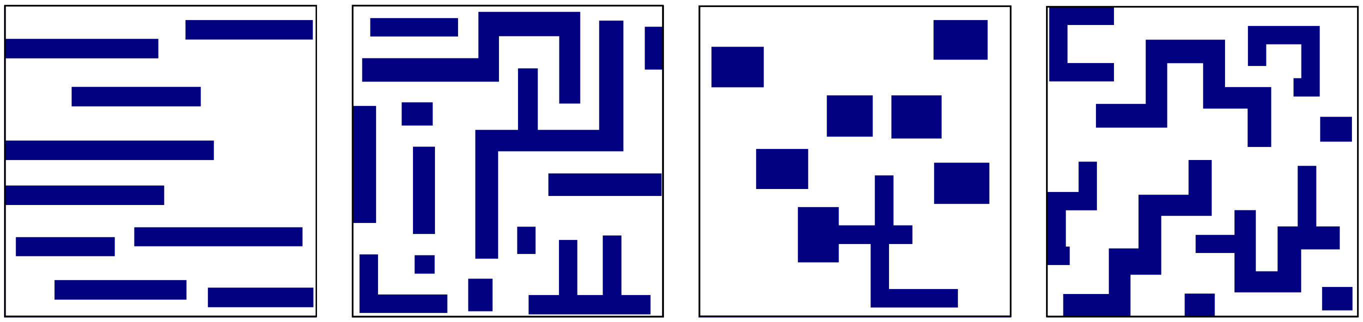

(a) Parallel paths (b) High density (c) VEM cluster (d) Multiple corners

Fig. 4: Custom patterns generated by VIPER

lengths, widths, branches etc. Examples of custom patterns generated automatically using VIPER are shown in Figure 4.

E. Adaptability to major inflections in fabrication technology

Newer nodes use advanced technologies to tackle various fabrication challenges and, in turn, introduce process-driven changes to design methodologies. An effective synthetic layout pattern generation tool must be capable of adapting to such changes. Any layout pattern generation tool can easily adapt to certain process changes, such as the shift from single exposure lithography to double/triple exposures, as it can continue to generate single color layout patterns, which can be later colored using commercial layout coloring tools [6]. Adapting to other process changes, however, such as the move to Self-Aligned Double/Triple Patterning (SADP/SATP), can be quite challenging. Unlike the SOTA, VIPER is designed to seamlessly handle many major technology inflections without any changes to its source code. Examples illustrating such capabilities of VIPER include:

(i) Some newer nodes feature additional layers dedicated to local interconnections between power-rails and transistor source/drains, thereby, eliminating such design features from lower Back-End-Of-Line (BEOL) layers. When used to generate BEOL layer patterns, VIPER can accommodate such changes by simply setting the number of power-rail branches to zero.

(ii) In process nodes using SADP/SATP, drawn metal lines are required to be increasingly uniform and only a few, if any, corners are allowed. VIPER can adhere to such design requirements by simply setting the number of corners on a path to zero, or a small value.

IV. Experimental Results

The performance of VIPER is evaluated and contrasted to the SOTA based on the two primary objectives of synthetic pattern generation methods, namely (i) generation of a diverse set of patterns which cover a large portion of the design space, and (ii) generation of realistic patterns which closely resemble actual IC layouts.

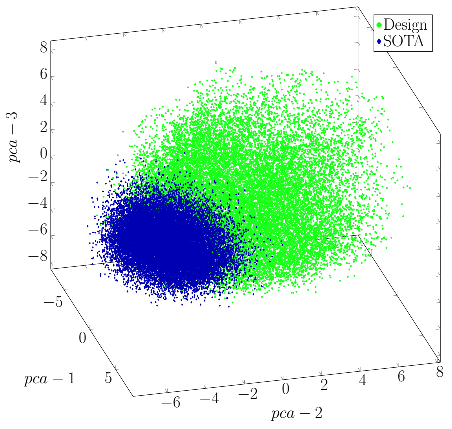
Our experiments are performed in a 45nm technology node [10]. An open-source circuit is placed and routed using the Nangate open cell library [11] to obtain baseline designs. The basic design rules from the same PDK are provided as inputs to both VIPER and the SOTA tool. As a prior node, we consider a 65nm technology and we use patterns from a single design at this node to learn the technology independent PDFs.

In the rest of the paper, we refer to patterns captured from real designs as 'Design patterns', patterns obtained from the SOTA tool as 'SOTA patterns', and patterns generated by VIPER as 'VIPER patterns'. The Design patterns were captured from the baseline design using a moving window scheme [12]. All patterns correspond to $Metal1$ and have their side dimensions equal to 8.5*$layer\_pitch$, which translates to about 1100nm. All generated patterns were subjected to a full Design Rule Check (DRC). On average, about 14% of the SOTA patterns and about 7% of VIPER patterns failed DRC. Most such DRC failures were found on complex rule-checks, both for VIPER patterns and for SOTA patterns. Only DR-clean patterns were used in the rest of the analysis.

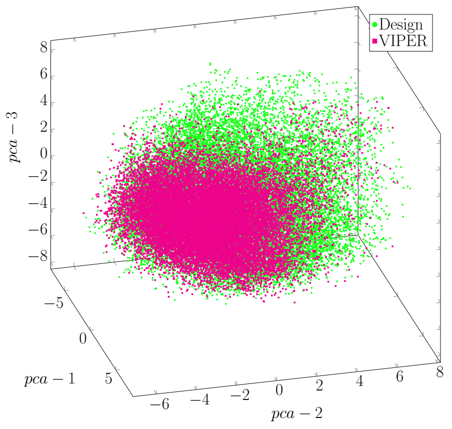
Both VIPER, and the SOTA tool were operated in their fully automated modes which require minimal human involvement. For the SOTA tool, this translates to mode 4, which is its most promising mode, as discussed in Section II.

A. Design Space Exploration

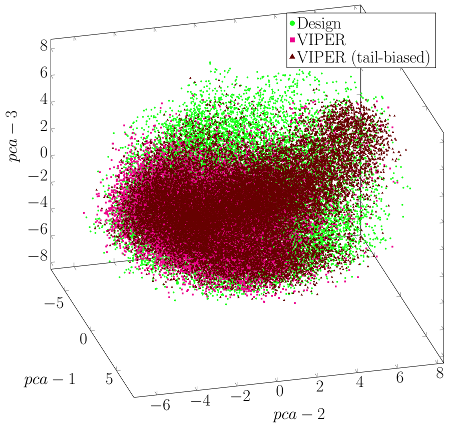
To determine the amount of design space explored by a pattern generation tool, we can use One-Class Classifiers (OCCs). OCCs are Machine Learning (ML) based models which are trained using a dataset containing samples belonging to a single class and learn a decision boundary that encompasses them. When a test set is presented to the trained OCC, the samples which fall within the learned boundary are classified as 'inliers', while the rest are classified as 'outliers' [13]. Prior to presenting layout snippets to an ML model, we must first convert them into numerical 'feature vectors'. For this purpose, we use the coordinate transform method [12]. We then use a one-class Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel as our OCC. While training ML models, such as OCCs, some model parameters may require fine-tuning. Such hyper-parameters control, for example, the 'conservativeness' of the classifier while learning the decision boundary. To preserve fairness, we have ensured that the same hyper-parameter values are used throughout our experiments.



(a) SOTA



(b) VIPER



(c) Tail-Biased VIPER

Fig. 5: Distribution of synthetic patterns w.r.t. design patterns

TABLE I: OCC results from design space coverage analysis

Training dataset	Testing dataset	Inliers	Outliers
SOTA patterns	Design patterns	26.61%	73.39%
VIPER patterns	Design patterns	73.5%	26.5%
VIPER patterns (tail-biased)	Design patterns	84.71%	15.29%

The rationale of our experiment is that, if we train an OCC model using patterns produced by a synthetic pattern generation tool and test it using actual patterns from the entire design space, the percentage of inliers provides a good estimate of the synthetic pattern generation tool effectiveness in covering the design space. To this end, we generated 50,000 SOTA patterns and 50,000 VIPER patterns, and used them to train two separate OCCs, which we then tested using a common dataset of 50,000 design patterns. The results are reported in Table I. To better visualize the results, we perform Principal Component Analysis (PCA) [14] on the design patterns, project VIPER and SOTA patterns onto the same space, and plot the first three principal components in Figure 5. As observed in Figure 5a, which contrasts the distribution of SOTA patterns against the distribution of Design patterns, the overlap is rather small, resulting in a low percentage of inliers (i.e., 27%) in our experiment. On the other hand, this overlap is significantly larger when we contrast the distribution of VIPER patterns against the distribution of Design patterns. As shown in Figure 5b, VIPER patterns are spread out in a larger area, thereby explaining the higher percentage of inliers (i.e., 73%) in our experiment.

In order to further guide VIPER towards exploring a broader design space, we use the PDFs learned from the previous technology node as a starting-point and bias them towards generating more samples at the tails of the distribution for various parameters, such as line widths, number of corners, etc. The biased PDFs are, then, used to generate a new set of 50,000 patterns, referred to as $VIPER_{Tail-Biased}$ patterns. Using this set of patterns, the classification procedure is repeated and the results are reported in Table I. In Figure 5c, we contrast the distribution of tail-biased patterns against both the Design patterns and the initial set of VIPER patterns. Evidently, tail-biased patterns explore design spaces well beyond the initial set of VIPER patterns, resulting in even higher design space coverage and inlier percentage (i.e., 85%), which is approximately 3X larger than the SOTA[1]. Since, the tail-biased patterns achieve the highest design space coverage, we use them as VIPER patterns for the rest of the analysis.

B. Realistic Pattern Generation

While the results of the previous subsection corroborate that the synthetically generated layout patterns cover a large portion of the design space, it is also important to ensure that they are realistic. In other words, they should originate from the same distribution as actual design patterns. Since, to the best of our knowledge, there is no metric for quantifying ‘realism’ in IC layout patterns, we resort once again to OCCs in order to evaluate the effectiveness of VIPER and the SOTA tool in generating realistic patterns.

The rationale of our experiment is that all realistic design patterns are governed by certain common characteristics and originate from a certain –yet unknown– distribution. Therefore, if we train an OCC to learn the boundaries enclosing that distribution in the numerically transformed layout feature space, we can then use the trained model to determine whether the patterns generated by the synthetic pattern generation tools fall within the learned boundary. Patterns resembling realistic layout snippets would be, then, classified as inliers, while unrealistic patterns would be classified as outliers. For instance, if a synthetically generated pattern is blank, or if it only contains one polygon covering the entire pattern area, it would likely be substantially different from the majority of real design patterns in the numerically transformed layout feature space and would, therefore, be classified as an outlier.

[1]While the SOTA could also potentially benefit from similar biasing, it does not offer a mechanism for such fine-grained adjustment.

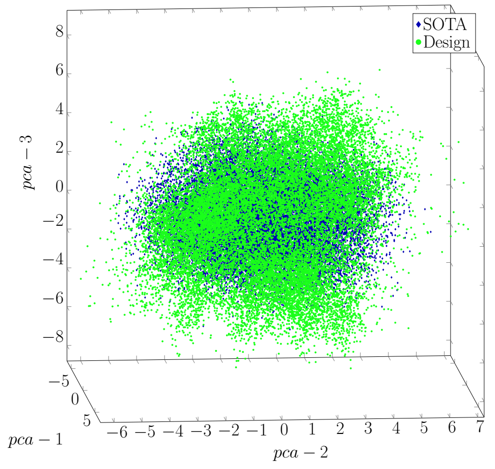


Fig. 6: Design patterns projected onto SOTA pattern space

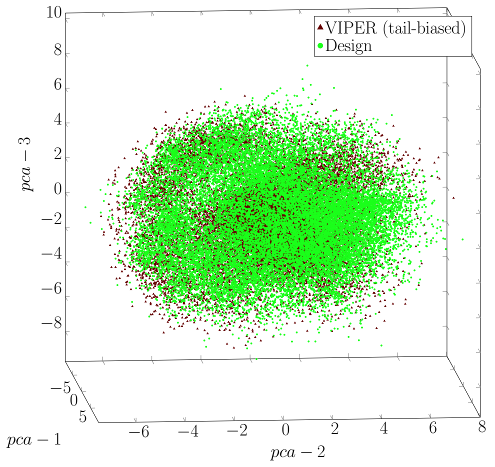


Fig. 7: Design patterns projected onto VIPER pattern space

TABLE II: OCC results from pattern realism analysis

Training dataset	Testing dataset	Inliers	Outliers
Design patterns	SOTA patterns	85.09%	14.91%
Design patterns	VIPER patterns (tail-biased)	90.57%	9.43%

In this analysis, we train an OCC using as known 'real patterns' the same set of design patterns which were described in Section IV-A. We then use the same trained classifier to test both the SOTA patterns and the VIPER patterns. The results are summarized in Table II. For better visualization, we perform PCA on the SOTA patterns and project the real design patterns onto the same space, as shown in Figure 6. A similar plot, where PCA is performed on VIPER patterns and real design patterns are projected onto its space, is shown in Figure 7. In both cases, we observe that the vast majority of SOTA and VIPER patterns, i.e., 85% and 90%, respectively, lie within the boundary of real design patterns, signifying that they belong to the same distribution. These results confirm that the patterns generated by both the SOTA tool and VIPER are realistic and resemble the characteristics of actual designs. Moreover, we note that VIPER tends to perform slightly better in generating realistic patterns, despite exploring a 3x larger portion of the design space.

V. Conclusion

The use of comprehensive and representative layout patterns during EDSE can play a catalytic role in robust development of advanced technology nodes. To this end, we introduced VIPER, a novel synthetic layout pattern generation method which is based on a controlled random-walk algorithm. VIPER leverages statistical design characteristics from prior technology nodes in order to not only generate realistic layout patterns but also to cover extensively the possible design space. As demonstrated experimentally, VIPER covers an approximately 3x larger design space in comparison to the SOTA commercial EDA tool, while ensuring realism of the generated layout patterns across the entire space.

Acknowledgment

This research has been partially supported by Semiconductor Research Corporation (SRC) through task 2709.001.

References

[1] A. Hamouda, M. Bahnas *et al.*, "Enhanced opc recipe coverage and early hotspot detection through automated layout generation and analysis," in *Optical Microlithography XXX*, vol. 10147, 2017.
[2] J. P. Cain, Y.-C. Lai *et al.*, "Methodology for analyzing and quantifying design style changes and complexity using topological patterns," in *Design-Process-Technology Co-optimization for Manufacturability X*, vol. 9781, 2016.
[3] P. Pathak, K. Krishnamoorthy *et al.*, "Methodology to extract, data mine and score geometric constructs from physical design layouts for analysis and applications in semiconductor manufacturing," in *Design-Process-Technology Co-optimization for Manufacturability X*, vol. 9781, 2016.
[4] H. Li, E. Zou *et al.*, "Design space exploration for early identification of yield limiting patterns," in *Design-Process-Technology Co-optimization for Manufacturability X*, vol. 9781, 2016.
[5] G. Rajavendra Reddy, J. Wallner *et al.*, "Pattern matching rule ranking through design of experiments and silicon validation," in *ASM International Symposium for Test and Failure Analysis*, 2018, pp. 443–448.
[6] Y. Chen, T. Gai *et al.*, "Hybrid hotspot library building based on optical and geometry analysis at early stage for new node development," in *Design-Process-Technology Co-optimization for Manufacturability XII*, vol. 10588, 2018.
[7] D. Ding, J. A. Torres *et al.*, "High performance lithography hotspot detection with successively refined pattern identifications and machine learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1621–1634, 2011.
[8] J.-W. Jeon, J. Song *et al.*, "Early stage hot spot analysis through standard cell base random pattern generation," in *Design-Process-Technology Co-optimization for Manufacturability XI*, vol. 10148, 2017.
[9] M. Zhang, G. Deng *et al.*, "A weak pattern random creation and scoring method for lithography process tuning," in *Design-Process-Technology Co-optimization for Manufacturability XII*, vol. 10588, 2018.
[10] FreePDK45. (2018) https://www.eda.ncsu.edu/wiki/FreePDK. [Online; accessed 1-Nov-2018].
[11] Nangate OCL. (2017) https://www.nangate.com/?page_id=22. [Online; accessed 1-May-2017].
[12] G. Rajavendra Reddy, C. Xanthopoulos *et al.*, "Enhanced hotspot detection through synthetic pattern generation and design of experiments," in *IEEE VLSI Test Symposium (VTS)*, 2018, pp. 1–6.
[13] M. M. Moya, M. W. Koch *et al.*, "One-class classifier networks for target recognition applications," pp. 797–801, 1993.
[14] S. Wold, K. Esbensen *et al.*, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.