Accurate Process-Hotspot Detection Using Critical Design Rule Extraction*

Yen-Ting Yu[1], Ya-Chung Chan[2], Subarna Sinha[3], Iris Hui-Ru Jiang[1], and Charles Chiang[4]
[1]Dept. of Electronics Engineering and Inst. of Electronics, National Chiao Tung University, Hsinchu, Taiwan
[2]MStar Semiconductor, Inc., Chupei, Taiwan
[3]Computer Science Department, Stanford University, Stanford, CA, USA
[4]Synopsys, Inc., Mountain View, CA, USA

ABSTRACT
In advanced fabrication technology, the sub-wavelength lithography gap causes unwanted layout distortions. Even if a layout passes design rule checking (DRC), it still might contain process hotspots, which are sensitive to the lithographic process. Hence, process-hotspot detection has become a crucial issue. In this paper, we propose an accurate process-hotspot detection framework. Unlike existing DRC-based works, we extract only critical design rules to express the topological features of hotspot patterns. We adopt a two-stage filtering process to locate all hotspots accurately and efficiently. Compared with state-of-the-art DRC-based works, our results show that our approach can reach 100% success rate with significant speedups.

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Design Aids

General Terms
Algorithms, Design.

Keywords
Design for manufacturability; process hotspot; pattern matching, design rule checking; lithography.

1. INTRODUCTION
As the printed feature size becomes smaller than the actual lithographic wavelength in advanced fabrication technology, the sub-wavelength lithography gap leads to unwanted shape distortions of the printed patterns. Even in a DRC-clean layout, some layout patterns are still sensitive to the lithographic process. These potentially problematic layout patterns, referred to as *process hotspots*, should be replaced with yield-friendly configurations. Therefore, detecting process hotspots in a layout is of particular importance to enable this correction process.

Recently, many research endeavors have been devoted to process-hotspot detection (also known as *pattern matching*) [1–8]. These approaches can be classified into four categories:

* This work was partially supported by Synopsys and NSC of Taiwan under Grant No. NSC 100-2220-E-009-047.

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

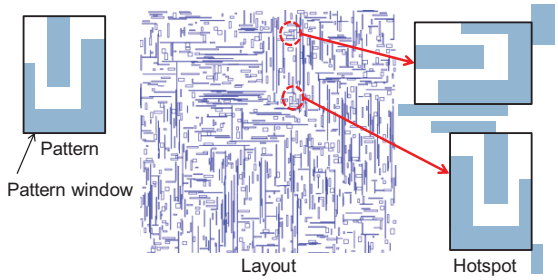


Fig. 1. Hotspots are accurately identified by our approach. We consider eight possible orientations and can handle the hotspots surrounded by arbitrarily-shaped polygons.

1) Graph-based hotspot detection: Kahng *et al.* present a pioneering work on hotspot detection in [1]. They create a dual graph to represent a given layout. Then, they filter out over-weighted edges and faces according to a user-specified threshold value. However, this method may generate false alarms due to the simplified error model.

2) Machine-learning-based hotspot detection: Ding *et al.* propose a machine learning kernel in [2], where they extract hotspot features to train their artificial neural network. Recently, Ding *et al.* extend it to a hierarchical learning framework in [3]. Wuu *et al.* devise another machine learning engine in [4]. This approach suffers from long training time and false alarms.

3) String-matching-based hotspot detection: In [5] and [6], Yao *et al.* and Xu *et al.* use worm-like movement to investigate all possible windows within a layout. Each window is converted to a layout matrix, and the matrix and pattern are encoded by strings. String matching is then applied to identify hotspots. This approach is accurate, but the layout matrix conversion is time consuming because the grid size is very small for advanced technology nodes.

The above three approaches may suffer from inaccuracy or long running time. Different from them, the fourth approach is *DRC-based hotspot detection*, which leverages on DRC to improve the accuracy of the detection process. Typically, DRC-based hotspot detection first converts the topological features of process hotspots to design rules and then analyzes the DRC report to identify hotspots [7][8].

In [7], Pikus and Collins extract all lengths of polygon edges and distances between adjacent polygons inside a given pattern as the topological features. At the analysis step, they construct a search graph to record the locations reported by DRC. They traverse the search graph to identify hotspots. In [8], Gennari *et al.* exploit two techniques, 2D image-based DRC and hashing, to improve DRC-based hotspot detection. They use a hash table to store the

location and the configuration around each edge or corner in a layout. They then compute match factors between the pattern and layout to determine hotspots. However, a sophisticated hash function is required to prevent hash collisions.

To avoid inaccuracy and long running time, in this paper, we propose an accurate process-hotspot detection framework based on the DRC-based approach. We consider eight possible orientations (including combinations of four rotations (0°, 90°, 180°, 270°) and two mirrors (horizontal and vertical mirrors)) and allow that arbitrarily-shaped polygons surround around the pattern window as shown in Fig. 1.

To achieve this goal, unlike existing DRC-based works, we target to find the topological features that can express a given hotspot pattern sufficiently and efficiently. Consequently, first of all, we adopt an efficient representation to model the pattern, and then we extract only critical topological features from the representation and convert them to design rules. Second, considering eight orientations, we apply DRC to find all locations that match any of these rules within a layout. Finally, we propose a two-stage filtering process to identify all hotspot locations accurately and efficiently. The key features of our approach include:

- We extract only critical rules: The classic way is to interpret all topological relations of edges within a pattern to design rules. However, doing so may generate numerous design rules and induce tremendous locations reported by DRC, thus making subsequent analysis difficult. On the other hand, too few extracted rules may also result in tremendous locations reported and complicated analysis. Hence, we extract only critical design rules to facilitate DRC and subsequent analysis.
- Our hotspot detection is accurate: We propose a two-stage filtering process; pre-filtering indicates potential locations, while finalization verifies exact locations. Compared with other analysis techniques adopted by state-of-the-art DRC-based approaches, our results show that our approach can detect hotspots 100% accurately with significant speedups.

The remainder of this paper is organized as follows. Section 2 briefly introduces design rule checking and gives the problem formulation. Section 3 details our process-hotspot detection framework. Section 4 shows our experimental results. Finally, Section 5 concludes this paper.

2. PRILIMINARIES

In this section, we briefly introduce design rule checking (DRC) and give the problem formulation.

2.1 Design Rule Checking

Design rules are a set of parameters to guarantee the manufacturability of a layout. For a specific manufacturing process, foundries provide the corresponding set of rules to ensure sufficient margins to compensate the variability during manufacturing. If these rules are violated, the design may not operate correctly. Fig. 2 indicates the most fundamental design rules. For a single layer, a width rule specifies the minimum width of any shape in the layout, while a spacing rule specifies the minimum distance between two neighboring objects. For two layers, an enclosure rule specifies an object should be covered with some additional margin by some object on the other layer.

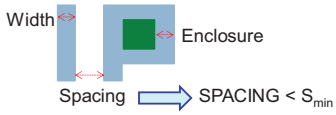


Fig. 2. Basic design rules: width, spacing, and enclosure.

In addition to the fundamental rules, modern DRC tools can perform general dimensional checks within a single polygon (including length, width, area, overlap, ratio, and density calculations) or between polygon edges (including intersecting polygon spacings, enclosure spacings and external polygon spacings). Given a *runset* file (design rules for a specific process) and a layout, a DRC tool reports design rule violations (indicating locations and violated rules). Basically, design rules can be expressed by equations and/or inequalities. For example, the minimum spacing rule can be described as the spacing between any two adjacent polygon edges is smaller than the specified value as shown in Fig. 2. The DRC tool then indicates the locations where there exist some edges violate the minimum spacing value.

2.2 Problem Formulation

As mentioned in Section 1, detecting process hotspots in a layout is a crucial issue. The hotspot detection problem is formulated as follows.

The Hotspot Detection Problem:
Given a hotspot pattern and a layout, our goal is to report all hotspot locations with eight possible orientations in the layout.

Hotspot patterns are patterns with exact dimensions which are provided by foundries. A hotspot location means the layout configuration at this location exactly matches that of the pattern inside the pattern window.

3. OUR HOTSPOT DETECTION FRAMWORK

In this section, we detail our DRC-based hotspot detection framework as shown in Fig. 3. First of all, we extract only the critical topological features of a given hotspot pattern and convert them to design rules. We devise an efficient representation–Modified TCG–to facilitate the extraction. Second, we apply these rules to DRC. DRC reports all locations that fit any generated rule considering eight possible orientations. Finally, we adopt a two-stage filtering process to analyze the DRC results: Pre-filtering indicates all potential locations that match the given pattern, while finalization identifies all true locations. We can easily extend our approach to consider multiple patterns simultaneously.

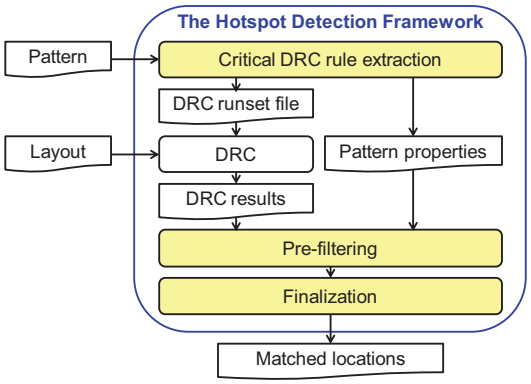


Fig. 3. The overview of our hotspot detection framework.

3.1 Modified TCG and Critical DRC Rule Extraction

To use the aid of DRC to realize hotspot detection, we shall extract design rules from the given pattern. The classic way is to interpret all topological relations of edges within a pattern to design rules. However, doing so may generate numerous design rules and induce tremendous locations reported by DRC thus making subsequent analysis difficult. On the other hand, too few extracted rules may also result in tremendous locations reported and complicated analysis. Hence, our goal is to extract *only* critical design rules to facilitate DRC and subsequent analysis.

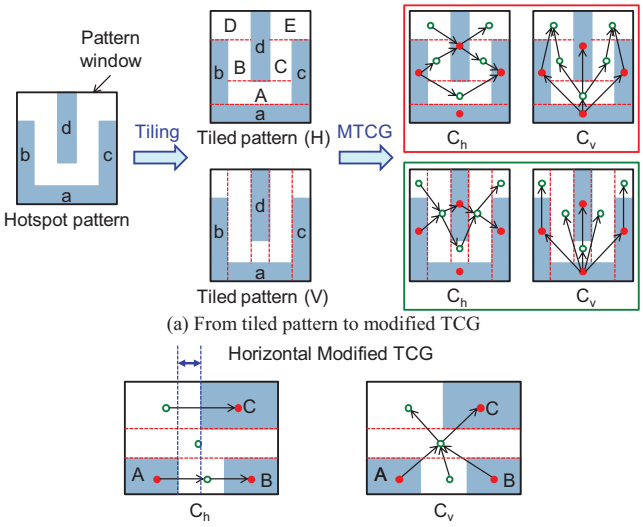
There are two tasks: 1) to model the given pattern by a good representation that can reflect topological features, and 2) to select critical features from the representation and translate them to design rules.

To accomplish the first task, we extend *transitive closure graph* (TCG) representation proposed by Lin and Chang in [9]. TCG is widely used to represent a compact placement; it uses a pair of constraint graphs, C_h and C_v, to record geometric relations among modules. However, hotspots may not be in a compact form because of spacing among polygons (see Fig. 4(a)). The spacing among polygons (i.e., white spaces) is essential for hotspot detection, which contains topological features of a pattern. In order to consider spacing by TCGs, we tile the pattern. We stretch the horizontal edges of each polygon until they reach other polygons or the window boundaries (see Fig. 4(a)). After horizontal tiling, a pattern is composed of block tiles (contributed by polygons) and space tiles (contributed by white spaces), and the pattern becomes compact.

We convert the horizontally tiled pattern to a horizontal modified TCG (MTCG). It is known that *each compact placement can be represented by a unique TCG*; a modified TCG inherits this property, i.e., MTCG can represent a unique tiled pattern. In an MTCG, each vertex represents a block tile (dot) or a space tile (circle), while each edge represents some topological relation among tiles.

MTCGs can be constructed by the sweep line algorithm. In the vertical constraint graph C_v, a directed edge is added between any two adjacent tiles if their projections on x-axis overlap. Similarly, in the horizontal constraint graph C_h, a directed edge is added between any two adjacent tiles if their projections on y-axis overlap. Moreover, the diagonal relations among block tiles can be extracted when a space vertex with one incoming and one outgoing edges connected to the same pair of block vertices in C_v and C_h. The diagonal relation between two corner-touched block tiles is directly checked from constraint graphs. Since spacing is considered, and the tiled pattern is compact, the transitive edges (which are redundant) can be simplified during MTCG construction. Without these transitive edges, our MTCGs are thus sparse.

To fully represent a given pattern, we adopt not only a horizontal MTCG but also a vertical MTCG. Consider the example shown in Fig. 4(b). If we use only the horizontal MTCG, we cannot extract



(a) From tiled pattern to modified TCG

(b) One-way tiling

Fig. 4. Modified TCG construction.

the horizontal distance between block tiles A and C. The properties of MTCGs are summarized as follows.

Lemma 1: MTCG is compact for a given tiled pattern.

Theorem 1: MTCG is unique for a given tiled pattern.

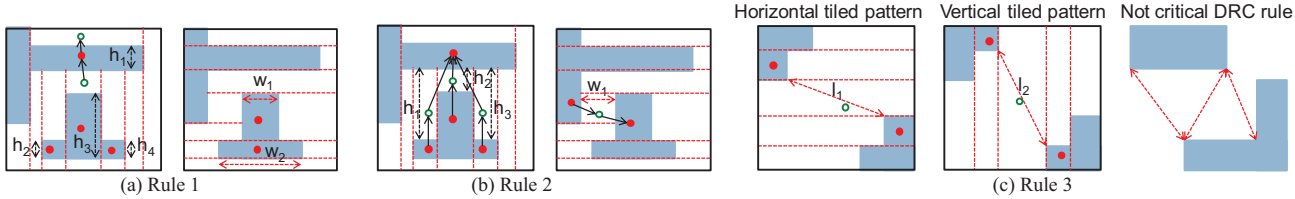
Moreover, recall that in the aforementioned classic way, a complete (dense) graph is needed to record all topological relations. In contrast, since redundant relations are not included, our MTCGs are sparse. The good representation–MTCGs–facilitates the second task, critical feature selection.

To accomplish the second task, we extract the following critical topological features. One of our goals is to handle patterns that are surrounded by arbitrarily-shaped polygons. Hence, we first focus on the internal topological relations (see Fig. 5). These primary rules can be expressed by equations.

1) Rule one–the width and height of a block tile: As shown in Fig. 5(a), we find the dimension of each block tile that does not touch the window boundary. Given an MTCG, we extract all block vertices whose incoming and outgoing edges are connected to space vertices.

2) Rule two–the distance between two adjacent block tiles: As shown in Fig. 5(b), we find the dimensions of all space tiles that do not touch the window boundary and are located in between block tiles. Given an MTCG, we extract any space vertex which lies in between exactly two block vertices.

3) Rule three–the diagonal relations between two convex corners of block tiles: As shown in Fig. 5(c), we find the diagonal relations between any two convex corners of block tiles. Given an MTCG, we extract space vertices whose in and out degrees are larger than two and also check their diagonal relations and distance.



(a) Rule 1 (b) Rule 2 (c) Rule 3

Fig. 5. Primary critical rules.

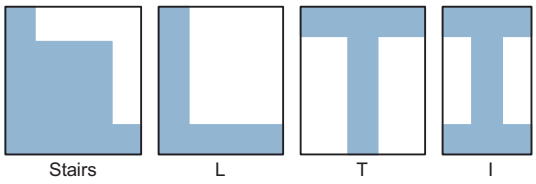


Fig. 6. Special cases.

The primary rules can handle most patterns. However, the primary rules may be insufficient for some special cases. As shown in Fig. 6, we cannot extract any primary rules for patterns "Stairs", "L", and "T", and we have only one rule for pattern "I". Too few rules imply that too many redundant locations could be reported. To overcome this difficulty, we add two secondary rules for tiles that touch the window boundary. The secondary rules can be expressed by inequalities.

4) Rule four–the space or block tile with one edge touching the window boundary: As shown in Fig. 7(a), we identify boundary tiles. Given an MTCG, we extract a space vertex that has zero indegree or outdegree in C_v (C_h) and has nonzero incoming and outgoing edges connected to block vertices in C_h (C_v).

5) Rule five–the space tile with two edges touching the window boundary or space tiles: As shown in Fig. 7(b), we extract the dimensions of space boundary tiles.

The secondary rules can handle the cases that the primary rules cannot, e.g., rule 4 can handle "T" and "I", while rule 5 can handle "Stairs" and "L". However, rule 5 is too general and may induce too many design rules. Hence, if we can extract critical rules based on the first four types of rules, we do not generate rules for rule 5 to speed up the subsequent process.

So far, we convert a pattern to MTCG and extract critical rules. It can be seen that MTCG is an efficient representation and can capture critical topological features well. Later, our results will show that we can achieve a 100% success rate to detect all hotspots.

A pattern may have eight possible orientations as shown in Fig. 8(a). Based on MTCG, the extracted critical rules express vertical and horizontal geometric relationships. Rules extracted from C_h are always perpendicular to rules extracted from C_v (see Fig. 8(b)). Consequently, we divide these eight orientations into two sets (see Fig. 8(a)). We generate a runset file for each set and run DRC twice to obtain the locations that hit any generated rule.

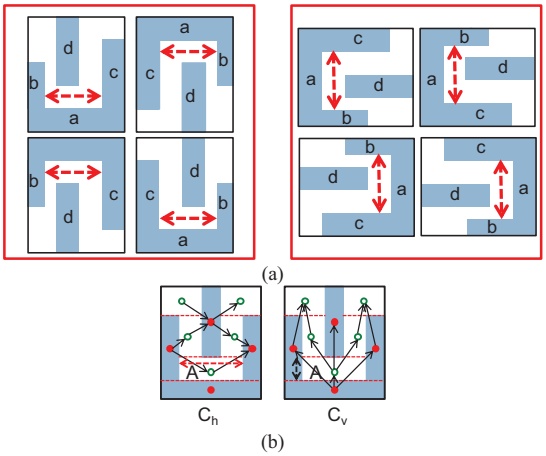


Fig. 8. (a) Eight orientations. (b) Rules extracted from C_h and C_v are mutually perpendicular, e.g., space tile A.

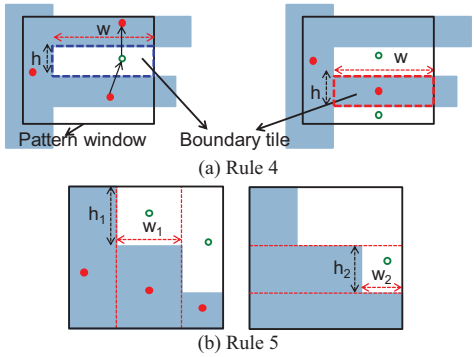


Fig. 7. Secondary critical rules.

3.2 Pre-filtering

Based on the DRC results and pattern properties, pre-filtering is applied to find the potential hotspot locations. Based on the way we set primary and secondary rules, we have the following property:

Theorem 2: Each extracted critical rule corresponds to a rectangle.

DRC reports the locations and dimensions of all polygons or spaces in the layout that match some specified critical rule. Based on Theorem 2, these reported block/spacing polygons are *rectangles*.

Hence, we create *rule rectangles* to record the pattern properties as follows. Given a pattern, a *reference point* is set to the bottom-left corner of its pattern window. Each extracted rule is modeled as a rule rectangle: A rule rectangle is associated with a width, a height, the relative distance (d_x, d_y) between the reference point and the bottom-left corner of this rectangle. As shown in Fig. 9(a), the spacing between polygons b and d is extracted based on rule 2 and is recorded by a rule rectangle. Totally five rule rectangles are recorded for this pattern.

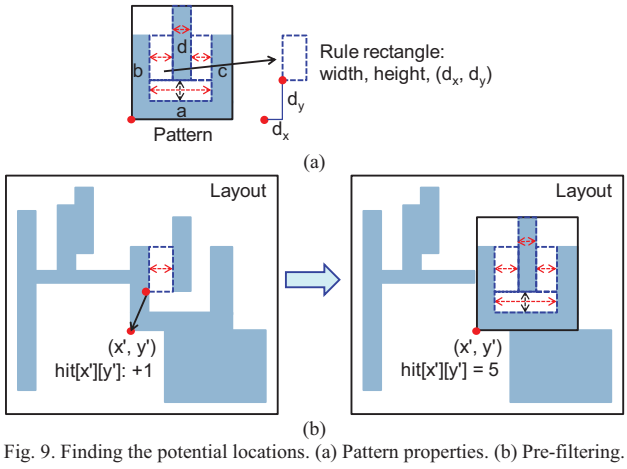


Fig. 9. Finding the potential locations. (a) Pattern properties. (b) Pre-filtering.

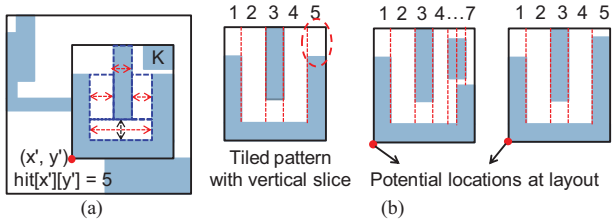


Fig. 10. Identifying the true hotspot locations. (a) Polygon K is not identified during pre-filtering. (b) Finalization. These two potential locations are not hotspots.

Pre-filtering indicates the potential hotspot locations by analyzing the rectangles reported by DRC and the rule rectangles (pattern properties) of the given pattern. To facilitate the analysis, we use a variable *hit*[*x*][*y*] to record the total number of rules matched at coordinate (*x*, *y*) and use a queue *Q* to store all *hit*[*x*][*y*] values. When parsing the DRC results, for each reported rectangle, we calculate the corresponding reference point (x', y') in the layout according to (d_x, d_y) set by the rule rectangle and increment its *hit*[x'][y'] value by 1. Finally, once the *hit* value in *Q* is equal to or greater than the number of rule rectangles, we find a potential hotspot location. For example, as shown in Fig. 9(b), we collect 5 matched rules at (x', y'), and thus (x', y') is a potential hotspot location.

3.3 Finalization

Pre-filtering indicates all locations in layout that match the extracted critical rules. However, some non-hotspot locations might pass pre-filtering. For example, in Fig. 10(a), (x', y') is reported by pre-filtering; nevertheless, there exists an extra polygon *K*, and (x', y') should be excluded. Hence, finalization is indeed necessary to identify true hotspot locations.

The finalization stage verifies the potential locations reported by pre-filtering. Since arbitrarily-shaped polygons may surround hotspots, first of all, we frame a checking window based on each potential location (which is the bottom-left corner of the window). Second, we vertically slice the layout inside the window. If the number of generated slices or the area of each tile within each slice is different from the given pattern, it is not a hotspot (see Fig. 10(b)).

4. EXPERIMENTAL RESULTS

Our algorithm was implemented in the C++ programming language on a Linux platform with a 2.4 GHz CPU with 16 GB RAM. The experiments were conducted with two layouts and with seven hotspot patterns. Table I summarizes the statistics of these layouts. Fig. 11 lists 7 patterns: "Stair 1", "I", and "S" are killer cases used to test the capability of our approach, "Mountain" and "Stair 2" are from [5], and "Ind1" and "Ind2" are extracted from real designs. These layouts are from real designs, and we randomly inject 6,400 and 9,600 exact hotspots for each pattern into Layout1 and Layout2, respectively. In addition, we also inject similar-shaped hotspots for patterns "Stair 1", "Stair 2", "I" to test the robustness of our framework. We adopt a state-of-the-art industrial DRC engine into our framework. The pattern and layout are described in GDS format. We have two experiments as follows.

4.1 The Impact of Critical Rule Extraction

Since rule five may induce too many design rules, *hierarchical* rule extraction used in our experiments means that rule 5 is applied only if no rules are extracted for rules 1 to 4. In the first experiment, we show the impact of the hierarchical rule extraction on the running time of the detection flow.

Table II compares hierarchical critical rule extraction with complete critical rule extraction in terms of running time, the number of matched rules reported by DRC, the number of finalized hotspot locations, and the success rate. First of all, the success rates of these two rule extraction strategies are both 100%. Among these patterns, "Stair 1" contains only rule 5. As expected, the rules matched by DRC are tremendous, and thus pre-filtering and finalization are slow. It is reasonable that hierarchical critical rule extraction performs equally well as complete critical rule extraction for "Stair 1". Patterns "S" and "Ind1" have primary and secondary DRC rules. In these cases, the hierarchical extraction achieves significant speedups.

4.2 Comparison with Other DRC-based Approaches

The second experiment compares our flow with other DRC-based approaches. For fair comparison, we incorporate the ideas of edge-based rule extraction and corner-based hotspot analysis proposed in [8] into our detection framework.

Table III lists the results of using edge-based rule extraction. Edge-based rule extraction models the lengths of all polygon edges within the pattern window as design rules. Redundant and duplicated rules are removed in our experiments. Although the edge-based method extracts fewer rules, it incurs tremendous locations reported by DRC. In contrast, our critical rule extraction identifies critical topological features well (fewer locations reported by DRC) and thus leads to faster analysis (shorter running times of pre-filtering and finalization).

Table IV lists the results of using the corner-based hotspot analysis instead of using our finalization. The corner-based hotspot analysis uses a hash table to store the feature of each polygon corner. The feature used here is the polygon area inside a small window centered at the investigated corner, and the window size is set to 10% of the height and 10% of the width of the given pattern window. It can be seen that both analysis methods are correct, but our method outperforms corner-based hotspot analysis.

5. CONCLUSION

In this paper, we propose an accurate process-hotspot detection framework. Unlike existing DRC-based approaches, we extract only critical design rules to express the topological features of hotspot patterns. We adopt a two-stage filtering process to locate all hotspots accurately and efficiently. Our results show that our approach not only reaches 100% success rate but also results in a short DRC report and superior efficiency. Future work includes handling multi-layer, range, and incompletely-specified patterns.

6. REFERENCES

[1] A. B. Kahng *et al.* Fast dual graph based hotspot detection. In *Proc. SPIE*, vol. 6349, pp. 628–635, 2006.

[2] D. Ding *et al.* Machine learning based lithographic hotspot detection with critical feature extraction and classification. In *Proc. ICICDT*, pp. 219–222, 2009.

[3] D. Ding *et al.* High performance lithographic hotspot detection using hierarchically refined machine learning. In *Proc. ASP-DAC*, pp. 775–780, 2011.

[4] J.-Y. Wuu, *et al.* Rapid layout pattern classification. In *Proc. ASP-DAC*, pp. 781–786, 2011.

[5] H. Yao *et al.* Efficient process-hotspot detection using range pattern matching. In *Proc. ICCAD*, pp.625–632, 2006.

[6] J. Xu *et al.* Accurate detection for process-hotspots with vias and incomplete specification. In *Proc. ICCAD*, pp. 839–846, 2007.

[7] F. G. Pikus and T. W. Collins, Jr. Topological pattern matching. *US Patent Application* 2010/018594 A1, Jul. 2010.

[8] F. E. Gennari *et al.* Fast pattern matching. *US Patent* 7818707, Oct. 2010.

[9] J.-M. Lin and Y.-W. Chang. TCG: A Transitive closure graph based representation for non-slicing floorplans. In *Proc. DAC*, pp. 764–769, 2001.

TABLE I. LAYOUTS

	Layout1	Layout2
Area (mm^2)	1.2×1.2	1.2×1.2
#Polygon*	661,056	1,028,622

*#Polygon: the number of polygons.
The layouts are based on 32nm process.

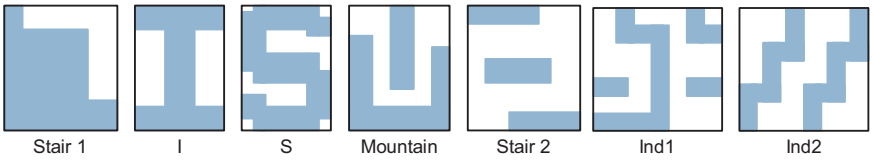


Fig. 11. Hotspot patterns.

TABLE II. HIERARCHICAL VS. COMPLETE CRITICAL RULE EXTRACTION

	Pattern	Layout	Critical rule extraction time (s)	Critical rule extraction #ruleC* (R)	DRC time (s)	DRC #ruleD** (A)	Pre-filtering time (s)	Pre-filtering #hotspot (B)	Finalization time (s)	Finalization #hotspot*** (C)	Total time (s)	Ratio (A/RB)	Ratio (B/C)	Success rate
Hierar-chical	Stair 1	Layout1	<0.01	(0,0,0,0,3)	41	2,396,343	106.68	38,400	57.86	6,400	205.54	20.80	6.00	100%
		Layout2	<0.01		63	3,710,439	164.20	57,592	125.42	9,600	352.62	21.48	6.00	100%
	I	Layout1	<0.01	(1,0,0,1,0)	9	45,004	1.51	25,600	24.79	6,400	35.30	0.88	4.00	100%
		Layout2	<0.01		9	67,640	2.27	38,400	56.25	9,600	67.52	0.88	4.00	100%
	S	Layout1	<0.01	**(1,3,0,7,0)**	**17**	**609,013**	**39.81**	**6,400**	**1.60**	**6,400**	58.41	8.65	1.00	**100%**
		Layout2	<0.01		**35**	**914,217**	**61.16**	**9,600**	**3.55**	**9,600**	99.71	8.66	1.00	**100%**
	Mountain	Layout1	<0.01	(3,0,0,1,0)	8	64,698	1.92	6,400	1.60	6,400	11.52	2.53	1.00	100%
		Layout2	<0.01		11	97,418	2.98	9,600	3.54	9,600	17.52	2.54	1.00	100%
	Stair 2	Layout1	<0.01	(2,1,0,0,0)	6	77,159	2.49	12,800	6.26	6,400	14.75	2.01	2.00	100%
		Layout2	<0.01		9	115,950	3.99	19,200	14.18	9,600	27.17	2.01	2.00	100%
	Ind1	Layout1	<0.01	**(3,3,0,6,0)**	**23**	**596,472**	**31.91**	**6,400**	**1.59**	**6,400**	56.50	7.77	1.00	**100%**
		Layout2	<0.01		**35**	**895,377**	**50.44**	**9,600**	**3.61**	**9,600**	89.05	7.77	1.00	**100%**
	Ind2	Layout1	<0.01	(2,4,0,4,0)	24	512,941	25.53	6,400	1.58	6,400	51.11	8.01	1.00	100%
		Layout2	<0.01		37	769,949	40.10	9,600	3.53	9,600	80.63	8.02	1.00	100%
Ratio					**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**			
Complete	Stair 1	Layout1	<0.01	(0,0,0,0,3)	41	2,396,343	106.68	38,400	57.86	6,400	205.54	20.80	6.00	100%
		Layout2	<0.01		63	3,710,439	164.20	57,592	125.42	9,600	352.62	21.48	6.00	100%
	I	Layout1	<0.01	(1,0,0,1,0)	9	45,004	1.51	25,600	24.79	6,400	35.30	0.88	4.00	100%
		Layout2	<0.01		9	67,640	2.27	38,400	56.25	9,600	67.52	0.88	4.00	100%
	S	Layout1	<0.01	**(1,3,0,7,2)**	**39**	**2,876,939**	**420.66**	**6,400**	**1.59**	**6,400**	**461.25**	**34.58**	**1.00**	**100%**
		Layout2	<0.01		**89**	**4,431,688**	**652.42**	**9,600**	**3.56**	**9,600**	**744.98**	**35.51**	**1.00**	**100%**
	Mountain	Layout1	<0.01	(3,0,0,1,0)	8	64,698	1.92	6,400	1.60	6,400	11.52	2.53	1.00	100%
		Layout2	<0.01		11	97,418	2.98	9,600	3.54	9,600	17.52	2.54	1.00	100%
	Stair 2	Layout1	<0.01	(2,1,0,0,0)	6	77,159	2.49	12,800	6.26	6,400	14.75	2.01	2.00	100%
		Layout2	<0.01		9	115,950	3.99	19,200	14.18	9,600	27.17	2.01	2.00	100%
	Ind1	Layout1	<0.01	**(3,3,0,6,1)**	**56**	**2,479,937**	**105.02**	**6,400**	**1.67**	**6,400**	**162.69**	**29.81**	**1.00**	**100%**
		Layout2	<0.01		**84**	**3,835,998**	**167.63**	**9,600**	**3.58**	**9,600**	**255.21**	**30.74**	**1.00**	**100%**
	Ind2	Layout1	<0.01	(2,4,0,4,0)	24	512,941	25.53	6,400	1.58	6,400	51.11	8.01	1.00	100%
		Layout2	<0.01		37	769,949	40.10	9,600	3.53	9,600	80.63	8.02	1.00	100%
Ratio					**1.48**	**1.98**	**3.17**	**1.00**	**1.00**	**1.00**	**2.13**			

TABLE III. EDGE-BASED RULE EXTRACTION (VS. OUR CRITICAL RULE EXTRACTION)

	Pattern	Layout	Critical rule extraction time (s)	Critical rule extraction #ruleC (R)	DRC time (s)	DRC #ruleD (A)	Pre-filtering time (s)	Pre-filtering #hotspot (B)	Finalization time (s)	Finalization #hotspot (C)	Total time (s)	Ratio (A/RB)	Ratio (B/C)	Success rate
Edge-based	Stair 1	Layout1	<0.01	(0,0,0,0,3)	41	2,396,343	106.68	38,400	57.86	6,400	205.54	20.80	6.00	100%
		Layout2	<0.01		63	3,710,439	164.20	57,592	125.42	9,600	352.62	21.48	6.00	100%
	I	Layout1	<0.01	**(0,0,0,0,2)**	29	**1,533,876**	139.97	25,600	25.08	6,400	194.05	29.96	4.00	100%
		Layout2	<0.01		43	**2,409,841**	221.92	38,400	55.64	9,600	320.56	31.38	4.00	100%
	S	Layout1	<0.01	**(0,0,0,0,5)**	48	**2,742,895**	503.23	6,400	1.59	6,400	552.82	85.72	1.00	100%
		Layout2	<0.01		91	**4,431,688**	650.17	9,600	3.51	9,600	744.68	92.33	1.00	100%
	Mountain	Layout1	<0.01	**(0,0,0,0,5)**	45	**2,538,307**	151.36	6,400	1.59	6,400	197.95	79.32	1.00	100%
		Layout2	<0.01		67	**3,923,568**	241.98	9,600	3.53	9,600	312.51	81.74	1.00	100%
	Stair 2	Layout1	<0.01	**(0,0,0,0,3)**	41	**2,319,777**	150.35	12,800	6.25	6,400	197.60	60.41	2.00	100%
		Layout2	<0.01		60	**3,607,091**	239.57	19,200	14.10	9,600	313.67	62.62	2.00	100%
	Ind1	Layout1	<0.01	**(0,0,0,0,4)**	46	**2,614,473**	397.87	6,400	1.62	6,400	445.49	102.13	1.00	100%
		Layout2	<0.01		67	**4,037,635**	643.34	9,600	3.55	9,600	713.89	105.15	1.00	100%
	Ind2	Layout1	<0.01	**(0,0,0,0,3)**	25	**1,307,058**	174.33	6,400	1.61	6,400	200.94	68.08	1.00	100%
		Layout2	<0.01		36	**2,025,831**	277.28	9,600	3.56	9,600	316.84	70.34	1.00	100%
Ratio					**2.15**	**3.64**	**7.59**	**1.00**	**1.00**	**1.00**	**4.34**			

TABLE IV. CORNER-BASED ANALYSIS (VS. OUR FINALIZATION)

	Pattern	Layout	Critical rule extraction time (s)	Critical rule extraction #ruleC (R)	DRC time (s)	DRC #ruleD (A)	Pre-filtering time (s)	Pre-filtering #hotspot (B)	Finalization time (s)	Finalization #hotspot (C)	Total time (s)	Ratio (A/RB)	Ratio (B/C)	Success rate
Corner-based	Stair 1	Layout1	<0.01	(0,0,0,0,3)	41	2,396,343	106.68	38,400	**72.61**	6,400	220.29	20.80	6.00	100%
		Layout2	<0.01		63	3,710,439	164.20	57,592	**158.03**	9,600	385.23	21.48	6.00	100%
	I	Layout1	<0.01	(1,0,0,1,0)	9	45,004	1.51	25,600	**30.49**	6,400	41.00	0.88	4.00	100%
		Layout2	<0.01		9	67,640	2.27	38,400	**70.03**	9,600	81.30	0.88	4.00	100%
	S	Layout1	<0.01	(1,3,0,7,0)	17	609,013	39.81	6,400	**1.92**	6,400	58.73	8.65	1.00	100%
		Layout2	<0.01		35	914,217	61.16	9,600	**4.37**	9,600	100.53	8.66	1.00	100%
	Mountain	Layout1	<0.01	(3,0,0,1,0)	8	64,698	1.92	6,400	**2.00**	6,400	11.92	2.53	1.00	100%
		Layout2	<0.01		11	97,418	2.98	9,600	**4.49**	9,600	18.47	2.54	1.00	100%
	Stair 2	Layout1	<0.01	(2,1,0,0,0)	6	77,159	2.49	12,800	**7.85**	6,400	16.34	2.01	2.00	100%
		Layout2	<0.01		9	115,950	3.99	19,200	**17.58**	9,600	30.57	2.01	2.00	100%
	Ind1	Layout1	<0.01	(3,3,0,6,0)	23	596,472	31.91	6,400	**1.98**	6,400	56.89	7.77	1.00	100%
		Layout2	<0.01		35	895,377	50.44	9,600	**4.53**	9,600	89.97	7.77	1.00	100%
	Ind2	Layout1	<0.01	(2,4,0,4,0)	24	512,941	25.53	6,400	**1.94**	6,400	51.47	8.01	1.00	100%
		Layout2	<0.01		37	769,949	40.10	9,600	**4.52**	9,600	81.62	8.02	1.00	100%
Ratio					**1.00**	**1.00**	**1.00**	**1.00**	**1.25**	**1.00**	**1.07**			

*#ruleC: the number of critical rules extracted (#rule1, #rule2, #rule3, #rule4, #rule5). **#ruleD: the number of locations reported by DRC
***#hotspot: the number of hotspots