Signature Indexing of Design Layouts for Hotspot Detection

Cristian Andrades
Department of Computer Science
Universidad de Concepción
Concepción 4070409, Chile
Email: crandrades@udec.l

M. Andrea Rodríguez
Department of Computer Science
Universidad de Concepción
Concepción 4070409, Chile
Email: andrea@udec.l

Charles C. Chiang
Synopsys Inc.
Mountain View, CA 94043
Email:clc@synopsys.com

***Abstract*—This work presents a new signature for 2D spatial configurations that is useful for the optimization of a hotspot detection process. The signature is a string of numbers representing changes along the horizontal and vertical slices of a configuration, which serves as the key of an inverted index that groups layout' windows with the same signature. The method extracts signatures from a compact specification of similar exact patterns with a fixed size. Then, these signatures are used as search keys of the inverted index to retrieve candidate windows that can match the patterns. Experimental results show that this simple type of signature has 100% recall and, in average, over 85% of precision in terms of the area effectively covered by the pattern and the retrieved area of the layout. In addition, the signature shows a good discriminate quality, since around 99% of the extracted signatures match each of them with a single pattern.**

I. Introduction

Indexing is a strategy that constructs a data structure to allow efficient searching. Indexes are typically defined as structures that group data by some specific search criteria. The work in this paper proposes to index the content of design layouts by using a signature as the search criteria. The main motivation of this work is to contribute to more efficient process-hotspot detection systems by selecting regions of the design layout to be used as inputs of the hotspot detection process.

A process-hotspot is a layout pattern that is sensitive to lithographic process [1]. Process-hotspot detection is of particular importance to enable the replacement of potentially problematic layout patterns. Different strategies for process-hotspot detection are machine-learning-based hotspot detection [2], [3], [4], string-matching-based hotspot detection [5], [6], and extraction of critical design rules [1]. Several of the approaches extract features from layouts or search patterns. The work in [3], [4] proposes a high performance hotspot detection based on a machine learning strategy that uses a 1D vector representing features such as corner information (convex or concave) and distance to externally and internally facing polygons. Similarly, the work in [2] uses a machine-learning kernel based on three major features in a binary pixel image: bounded rectangles, T-shape metal features, and L-shape metal features. The work in [7] extracts situations for optical proximity correction. A situation is a collection of shapes within a radius of extraction, which are described based on corners and edges. The work in [1] extracts critical topological features of a pattern and converts them into design rules. To do so, patterns are tiled and then represented in a modified *transitive closure graph* [8] (MTCG). In this graph, vertices are block tiles or space tiles, and edges are topological relations among tiles. In the vertical constraint graph, a directed edge is added between adjacent tiles if their projections on the x-axis overlap. This is similar for the horizontal constraint graph with respect to the y-axis. A basic property of a MTCG is that there is a unique MTCG for a given tile pattern.

Unlike previous work in process-hotspot detection, we propose to pre-process layouts in order to extract signatures that characterize windows within the layouts. The idea is to use these signatures to extract portions of the layout that could be considered for the hotspot detection process. The signature can be used as a search criteria and, therefore, be organized in an index structure. To illustrate the usefulness of the signature, we consider a pattern-matching process where the signatures of a set of patterns can be used as search keys of an index structure to return only possible layout's windows that should be analyzed in a hotspot-detection process. The signature, therefore, should be general enough to consider different possible patterns. Then, we consider the representation of range patterns as in [5], which enables a compactly representation of similar exact patterns of the same size.

A signature corresponds to a feature that represents a shape by a 1-dimensional function derived from shape boundary points. There are several possible shape signatures found in the literature, such as angle boundary signature, curvature or tangent boundary signature, distance boundary signature [9], and distance from boundary to centroid [10]. These signatures, however, do not characterize configurations composed of several geometric objects. In summary, the key contributions of this work are: (i) the definition of a signature for configurations composed of non-overlapping rectangles, (ii) the implementation of algorithms for the extraction of the signature from layouts by windows and from range patterns and (iii) the use

978-3-9815370-2-4/DATE14/

of the signature with an indexing structure to efficiently search for candidate windows that can match a hotspot pattern.

The organization of this paper is as follows. Section II presents the proposed signature. Section III shows the layout representation and the extraction of its signatures by window slicing. Section IV describes the range pattern specification and its signature extraction. The strategy for searching candidate windows containing process-hotspots is in Section V. Experimental evaluation is given in Section VI, followed by final conclusions and future research directions in Section VII.

II. Signature

A signature is a mapping from a geometric configuration composed of non-overlapping rectangles to one or more numbers. Unlike signatures characterizing single shapes such as those in [9] and [10], a signature here has to characterize a set of shapes (rectangles). We propose a simple but effective signature that characterizes 2D configurations composed of non-overlapping rectangles located along or perpendicular to x- or y-axis. To formalize this signature, let us first define the concepts of grid and slices of a representation of 2D configurations.

***Definition** 1:* A $N \times M$ *regular grid* representation of a configuration is composed by $N \times M$ of cells of the same size in a 2D space, such that there exist M vertical units and for each of these M units, there exist N cells along the horizontal axis. The value of a cell of the grid is 1 if the cell overlaps a rectangle of the configuration.

***Definition** 2:* Horizontal(vertical) slices of a grid is a set of contiguous horizontal or vertical cells (rows or columns, respectively) of a regular grid that are equal. Each slice contains a number of *fragments* that group equal cells in the orthogonal axis. Horizontal and vertical slices compose an *irregular grid* where cells can be of different sizes that represent cells of the regular grid that are equal.

Example 1: Figure 1 shows the 6×6 regular and 3×4 irregular grids that represent a configuration composed of 3 non-overlapping rectangles. In this example the granularity of the regular cells is such that each cell completely overlaps a rectangle or it does not overlap the rectangle at all. As this example shows, there are 3 horizontal slices, each of then composed of fragments along the x-axis. For instance, for horizontal slice 1, there are 2 segments: $F_{1,0}$ and $F_{1,1}$. Similarly, there are 4 vertical slices. This example shows that a regular grid is of larger dimension than an irregular grid, since the latter compacts regular cells which are equal in terms of the horizontal and vertical slices.

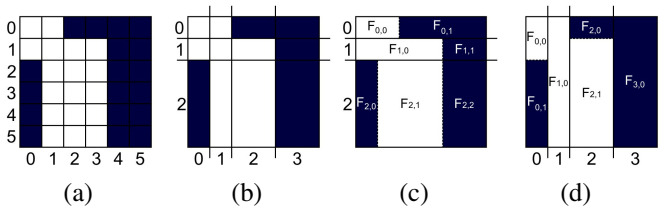

Fig. 1. Example of a configuration representation: (a) regular grid, (b) irregular grid, (c) horizontal slices, and (d) vertical slices

Using a grid representation, a change-based signatures is defined as follows.

***Definition** 3:* Let $N \times M$ be a grid representation of a 2D configuration of non-overlapping rectangles. The change-based signature ($\mathcal{S}^+$) is a tuple of the form $([h_0, \ldots, h_{N-1}], [v_0, \ldots, v_{M-1}])$, where h_i $(0 \leq i \leq N-1)$ is the number of changes of values along the horizontal slice i and v_j $(0 \leq j \leq M-1)$ is the number of changes of values along the vertical slice j.

Notice that the definition of the signature applies to grid representations and, therefore, to regular or irregular representations. In what follows, and unless we indicate the opposite, we will apply the signature to irregular grids.

Example 2: (Cont. Example 1.) For configuration in Figure 1, its $\mathcal{S}^+$ is the tuple $([1,1,2],[1,0,1,0])$.

It is easy to see that $\mathcal{S}^+$ will not change as we apply a continuous scaling over the whole configuration, since equal rows or columns are represented by a single slice. In addition, $\mathcal{S}^+$ will be the same for patterns that are similar and of the same size but differ in the distance between and width of rectangles (see Section IV). Also, by the definition of the signature, the number of changes along a slice is equivalent to the number of fragments in the slice minus 1.

III. Design layout representation and signature extraction

This section describes the representation of a design layout and the extraction of $\mathcal{S}^+$ by slicing a window over the layout. We assume that the signature extraction from layouts and index construction is an off-line process applied previously to the hotspot-detection process.

Similarly to [5], a layout is represented by a $N \times M$ regular grid L, denoted by $L_{N \times M}$, where N and M depend on the granularity of the layout representation, which is typically equal to the manufacturing grid size. Each element $L[i,j]$, with $0 \leq i \leq N-1$ and $0 \leq j \leq M-1$, is associated with a spatial location in the manufacturing grid, such that $L[i,j] = 1$ if there exists a rectangle that overlaps this location; otherwise, $L[i,j] = 0$. Figure 2 shows an example of a layout representation.

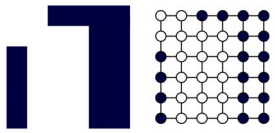

Fig. 2. Example of a layout representation

The $N \times M$ regular grid L representing a layout is preprocessed by windows, which are then mapped to irregular grids to extract $\mathcal{S}^+$. A window corresponds to a rectangular sub-portion of the layout and is defined as a $N' \times M'$ subgrid W of L, denoted by $W_{N' \times M'}$, with $1 \leq N' \leq N$ and $1 \leq M' \leq M$, and where rows (and columns) in W are consecutive in L. The dimension of the window is related to the dimension of the search patterns.

The signature extraction process starts with a window located at the top-left corner of the layout grid. Then, the window slides one-by-one position along the x-axis and then along the y-axis. Thus, given a layout of size $N \times M$ and a window size of $N' \times M'$, the number of possible windows in the layout is $(N - N' + 1) \times (M - M' + 1)$.

Windows are identified by using a correlative number following the window sliding. Given a layout $L_{N \times M}$, windows of size $N' \times M'$, and a number i representing the correlative window incremented by sliding the window horizontally and then vertically, coordinates for the top-left corner of the window are $(i/(M - M' + 1), i \bmod (M - M' + 1))$.

For each layout window, the horizontal and vertical slices are determined and $\mathcal{S}^+$ is extracted from the irregular-grid representation of the window.

Example 3: Let consider the simple layout represented with a 8×8 regular grid shown in Figure 3(a). Assume now a window size of 6×6, then the number of possible windows within the layout is 9. Figures 3(a)-(c) show windows created along the three possible horizontal rows. $\mathcal{S}^+$ for window 0 is $([0, 1, 2, 1], [2, 1, 1])$ and for window 1 is $([0, 2, 0], [1, 1, 1])$.

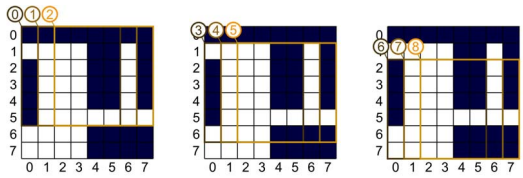

Fig. 3. A layout and the window for the signature extraction

Notice that a large layout can be partitioned to make a distributed extraction of $\mathcal{S}^+$. This is a domain decomposition processing of a layout. An important consideration is that consecutive portions of the layout sent to different nodes should overlap to be able to detect patterns located at their intersection.

IV. Pattern representation and signature extraction

Patterns are 2D configurations composed of non-overlapping rectangles. As such, they can also be described as irregular grids. Patterns of the same size can have small differences such that one could say that there exists a representative pattern with several similar realizations that vary in the distance between rectangles' boundaries. To avoid to represent each particular occurrence of a representative pattern, it is possible to compact the geometric information of patterns by using the specification of *range patterns* as in [5].

***Definition** 4:* A *range pattern* is a configuration of 2D non-overlapping rectangles with additional specifications about the horizontal or vertical distances between rectangles's boundaries.

The following example illustrates the case of different realizations of a general pattern that is then codified as a range pattern.

Example 4: Figure 4 illustrates three different realizations of the same range pattern with three rectangles.

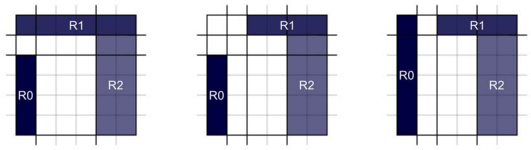

Fig. 4. Three different *realizations* of the same representative patterns with three rectangles

We can compress the specification of these three patterns by a range pattern as follows.

Horizontal constraints	Vertical constraints
1. $R_0.R - R_0.L$ is 1	1. $R_0.T - R_0.B = (4, 6)$
2. $R_1.R - R_1.L = (4, 6)$	2. $R_1.T - R_1.B$ is 1
3. $R_2.R - R_2.L$ is 2	3. $R_2.T - R_2.B$ is 5
4. $R_2.L - R_0.R$ is 3	4. $R_1.B - R_2.T$ is 0
5. $R_2.R - R_1.R$ is 0	5. $R_0.B - R_2.B$ is 0

Fig. 5. A range pattern specification ($R.L$: left boundary of R, $R.R$: right boundary, $R.T$: top boundary, $R.B$: bottom boundary)

Since a range pattern may have different realizations, it is not possible to extract $\mathcal{S}^+$ from a range pattern directly. $\mathcal{S}^+$ uses the horizontal and vertical slices of a realization of the pattern. Consequently, we derive possible realizations from range patterns that have the same irregular grid representation, i.e., the same horizontal and vertical slices. This is done by first mapping the range pattern specification to horizontal (HRG) and vertical (VRG) range graphs, independently, which are then combined to derive possible irregular grid representations. A range graph was introduced in [5], where it is described how to obtain a number of definite vertical or horizontal graphs from a range pattern. Definite horizontal or vertical graphs are instances of horizontal and vertical slices of irregular grids that satisfy the specification of a range pattern. For consideration of space, we refer to [5] for the algorithms to map range patterns to range graphs and illustrate here their application with the following example.

Example 5: (Cont. of Example 4). Figure 6 shows the horizontal and vertical range graphs from the specification of the range patten in Figure 5. In this figure, edges with bold labels denote edges that are indefinite and that lead to multiple possible definite graphs.

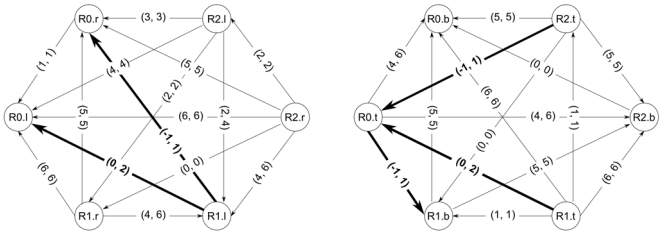

Fig. 6. Range graphs of the range pattern in Figure 5 : (a) horizontal range graph and (b) vertical range graph

A definite range-graph defines a topological order in which limits of slices can be defined. In these graphs, each label on an edge defines a possible range of distance between boundaries, where the range is of the form (i, j) or $(0, 0)$, with i and j being both positive or negative. By using algorithms in [5], we extract definite vertical and horizontal range graphs. But this is not enough since combinations of graphs may produce inconsistent configuration, that is, configurations where rectangles overlap. The following example illustrates this case.

Example 6: (Cont. of Example 5). Range graphs in Figure 6 derive 3 definitive horizontal and 3 definitive vertical graphs. The combinations of theses graphs produces initially 9 combinations, where one of them is inconsistent. Figure 7 shows the horizontal and vertical topological orders derived from range graphs that derive the corresponding inconsistent configuration where rectangles R_0 and R_1 overlap. Topological orders are specified by a sequence of sublists sorted by the coordinates along an axis. Each sublist is the list of boundaries that share the same coordinate.

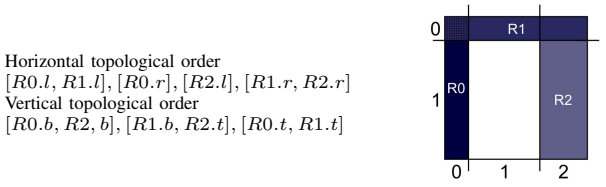

Fig. 7. An incompatible range graphs: (a) horizontal and vertical topological orders and (b) inconsistent resulting configuration

To overcome this situation, we designed and implemented an algorithm that combines both graphs and checks if the combination is consistent. This is included as an appendix of this paper.

V. Signature-based searching

Having patterns and layouts with their $\mathcal{S}^+$, it is possible to use $\mathcal{S}^+$ for selecting candidate windows of the layout that can contain hotspots. To do that, we use $\mathcal{S}^+$ as a search key in an inverted index. This inverted index is composed of a dictionary of possible values of $\mathcal{S}^+$ found in the layout. For each entry of the dictionary, there is a list of ids of windows whose signatures match the entry in the dictionary. Note that this structure can be created for a single layout or for several layouts if we consider to store not only the window's id but also a layout's id.

The following example illustrates the construction of an inverted index and its use for selecting candidates for hotspot matching.

Example 7: (Cont. of Example 3) Assume the range pattern of Figure 5, which results in 8 possible irregular grids, two of them with the same signature (see Figure 8). The difference between these two patterns with the same signature is the owner of top-left corner (i.e., R_0 or R_1).

Figure 9 shows the inverted index for the layout in Figure 3. This layout contains 9 different windows, but 7 different $\mathcal{S}^+$. For the range pattern of Figure 5, there are 7 different signatures (see Figure 9), and for each of them, there are 8 possible

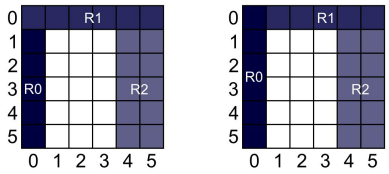

Fig. 8. Two different patterns from the same range pattern that share the same signature

rotations $(0°, 90°, 180°, 270°$, and their mirror realizations). In total, searching for these 7 different signatures requires to search for 56 possible rotations, which requires the efficient use of indexing structures.

$\mathcal{S}^+$	**List of windows**
$([0, 1, 2, 1], [2, 1, 1])$	$[0]$
$([0, 2, 0], [1, 1, 1])$	$[1]$
$([0, 3, 0], [1, 1, 1, 1])$	$[2]$
$([1, 2, 1, 1], [2, 0, 2])$	$[3]$
$([2, 1, 1], [1, 0, 2])$	$[6]$
$([2, 0, 1], [0, 2, 1])$	$[4, 7]$
$([3, 0, 1], [0, 2, 1, 2])$	$[5, 8]$

Fig. 9. Inverted index for layout in Figure 3

Searching for candidate windows takes the time needed for finding the signature in the dictionary and then, traversing the list of windows with the same signature. For efficiency, we can organize the dictionary with different types of structures such as a binary search tree, hash table, and another more sophisticated structure that makes possible searching from approximate and exact queries. For this work, we use a binary search tree with a cost of $O(logn)$, with n the number of entries in the dictionary, that is, the number of different signatures found in the layout. We chose the use of binary search trees over hash tables because they are able to optimize the search for range queries, that is, searching for groups of similar signatures.

VI. Experimental evaluation

Our algorithms were implemented in the C++ programming language on a Linux Platform of a server with 2 processors Intel Xeon QUAD core E5620 (2,40 GHZ / 12 MB cache L3) and 64 GB RAM. Due to non-disclosure agreement, we did not count with real layouts. Then, we created layouts with the following criteria: (i) rectangles were created and placed randomly on the layout and (ii) rectangles in a layout had a predominant vertical or horizontal orientation. Also, because our work is not about pattern matching, but about preprocessing layouts, we do not have previous work to compare with. Instead, we provide values of runtime for the preprocessing step and evaluate the quality of the signature to filter candidate portions of a layout that may contain process-hotspots.

Two design layouts were used in the following experimental evaluation: D_1 is a layer of $0.88 \times 0.88mm^2$ represented by a $13,540 \times 13,540$ regular grid and D_2 is a layer of $0.91 \times 0.91mm^2$ represented by a $14,000 \times 14,000$ regular grid. For

each of these layouts, we extracted $\mathcal{S}^+$ using 10×10 windows. Table I summarizes the number of signatures extracted from the layouts.

	D_1	D_2
Total windows	183.087.962	195.748.081
Number of $\mathcal{S}^+$	87.691.425	93.784.199
Min. windows per $\mathcal{S}^+$	1	1
Max. windows per $\mathcal{S}^+$	64.752	64.408
Average windows per $\mathcal{S}^+$	2,087866	2,087218
Standard deviation	33,078149	30,373846

TABLE I
SUMMARY OF SIGNATURES EXTRACTED FROM LAYOUTS

We inserted different realizations of 7 range patterns in D_1 and D_2 using a 10×10 regular-grid representation. These patterns were taken from related work [6]. Figure 10 shows a realization of each of the seven different patterns and Table II summarizes the range-patterns complexities.

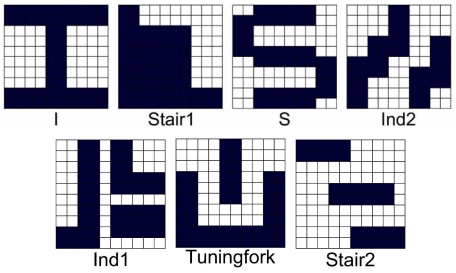

Fig. 10. A realization of each of the 7 range patterns used in the experimental evaluation

Range Pattern	Size	# rectangles	# irregular grids	# $\mathcal{S}^+$
I	10×10	3	1	1
Ind1	10×10	5	9	9
Ind2	10×10	5	15	12
S	10×10	5	16	16
Stair1	10×10	3	1	1
Stair2	10×10	3	6	6
Tuningfork	10×10	4	12	10

TABLE II
RANGE-PATTERN COMPLEXITIES

We checked the quality of $\mathcal{S}^+$ as the search key to retrieve candidate windows. We used classical measures of recall and precision of information retrieval. Recall determines the number of relevant elements that are retrieved versus the total number of relevant elements, and precision determines the number of relevant elements that are retrieved versus the total number of retrieved elements. We distinguished, however, two alternatives to calculate these measures: window-based and area-based measures. The window-based measure determines the total number of retrieved windows, the number of retrieved windows that actually contain patterns, and the total number of windows that indeed contain patterns. The area-based measure determines the retrieved area, the retrieved area that contains patterns, and the area of the layout that indeed contains patterns. For precision and recall, good values are close to 1.0.

Results indicate that the recall is always 100% using window-based or area-based measures, that is, $\mathcal{S}^+$ does not miss relevant elements. Figure 11 shows the precision of the candidate windows retrieved from layouts D_1 and D_2. In all cases, we filtered out over 97% of the number of windows or area of each layout by using the signature, which depends on the number of hotspots in the layout and the precision of the search process. However, area-based precision differs significatively from window-based precision. This is due to that consecutive windows share large part of their areas, which is counted just once for the area-based measure. Thus, if two consecutive retrieved windows are then seen not being relevant, taking the union of their areas counts less than counting the two windows separately. As you can see for patterns $Ind1$ and $Ind2$, they have important differences as a window slices along the horizontal axis, which results in different signatures. For patterns such as I and $Stair2$, in contrast, slicing a window along the horizontal axis does not represent great changes on the signature; however, windows do not contain the pattern with the desired dimension. Precision also depends on the type of pattern such that simple patterns as the $\mathrm{Stair1}$ are less differentiated by $\mathcal{S}^+$ than other types of patterns.

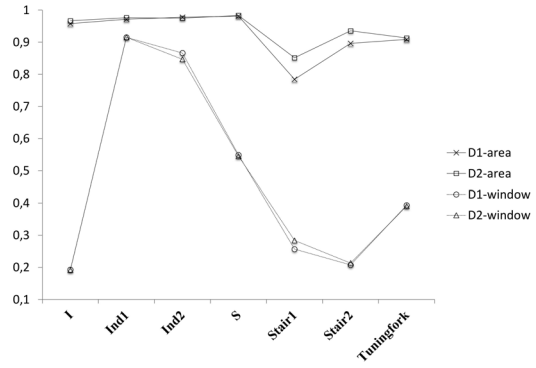

Fig. 11. Precision of the retrieved windows

Figure 12 shows how good the signature is to discriminate different patterns. This figure represents on the x-axis the number of different patterns with a single signature and the y-axis indicates the number of windows that were retrieved by using $\mathcal{S}^+$. For this evaluation, we analyzed four different small layouts. For example, in this figure, $L1_{5000\times5000}$ is a 5000×5000 where there are 2,147,885 (i.e., 5.67 base-10 logarithm) windows that match a signature representing a single pattern.

To explore the sensitivity of $\mathcal{S}^+$ to the window's size with respect to the pattern's size, we analyzed the use of $\mathcal{S}^+$ with windows whose sizes were larger than the pattern's size. To do so, we extracted signatures from the layout's windows using a regular instead of irregular representation of the layout and patterns. The idea is that a pattern may occur within a

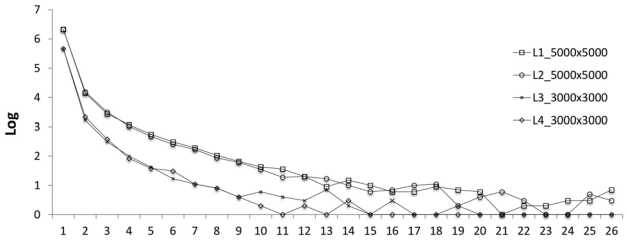

Fig. 12. Number of different patterns per signature

window if the number of changes along slices of the window are larger or equal to the number of changes along slices of the pattern. By applying this strategy, the recall keeps being 100% when the window's size increases, but precision decreases drastically. For pattern S, for example, the area-based precision decreases from 0.98 to 0.2 when using a window's size 10% larger than the pattern's size. Notice, however, that this evaluation searches for the exact match of the pattern S within the window and does not allow different sizes of the pattern S.

VII. Conclusion

We have proposed the use of a signature that can filter out portions of design layouts from the hotspot-detection process. We have shown how to extract the signature from layouts and also from range patterns, which are compact specifications of similar patterns. When the signature is extracted from layouts using a window's size equivalent to the pattern's size, it is able to find candidate windows for hotspot detection with good precision and without missing any possible candidate window.

Our experiments concentrate on the evaluation of the quality of the signature to discriminate regions with potential hotspots. We are now making improvements on the algorithms to make a more efficient process of preprocessing. Although the main motivation of this work was to decrease the time cost of the hotspot-detection process, we also visualize the use of the signature for different purposes. Signature similarity can also be used for comparing layouts. Due to the good property for distinguishing 2D configurations of non-overlapping rectangles, we can obtain histograms of signatures per layouts and derive distribution of different patterns within a layout. This has been left for future work. In addition, we want to explore new signatures that can be rotation independent and less sensitive to the window's size, and implement structures and algorithms that optimize the search for multiple patterns.

References

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

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

[3] D. Ding, A. J. Torres, F. G. Pikus, and D. Z. Pan, "High performance lithographic hotspot detection using hierarchically refined machine learning," in *Proceedings of the 16th Asia South Pacific Design Automation Conference, ASP-DAC*. IEEE, 2011, pp. 775–780.

[4] D. Ding, J. A. Torres, and D. Z. Pan, "High performance lithography hotspot detection with successively refined pattern identifications and machine learning," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1621–1634, 2011.

[5] H. Yao, S. Sinha, C. Chiang, X. Hong, and Y. Cai, "Efficient process-hotspot detection using range pattern matching," in *ICCAD*, 2006, pp. 625–632.

[6] J. Xu, S. Sinha, and C. Chiang, "Accurate detection for process-hotspots with vias and incomplete specification," in *ICCAD*, 2007, pp. 839–846.

[7] F. E. Gennari, Y.-C. Lai, M. W. Moskewicz, M. C. Lam, and G. R. McIntyre, "Fast pattern matching," *US Patent*, vol. US 7,818,707 B1, 2010.

[8] J.-M. Lin and Y.-W. Chang, "Tcg: A transitive closure graph-based representation for non-slicing floorplans," in *38th Design Automation Conference (DAC'01)*. ACM, 2001, pp. 764–769.

[9] A. A. Y. Mustafa, "Fuzzy shape matching with boundary signatures," *Pattern Recognition Letters*, vol. 23, no. 12, pp. 1473–1482, 2002.

[10] J. Chanussot, I. Nyström, and N. Sladoje, "Shape signatures of fuzzy star-shaped sets based on distance from the centroid," *Pattern Recognition Letters*, vol. 26, no. 6, pp. 735–746, 2005.

Appendix

The following algorithm checks whether or not the combination of an horizontal (HTO) and a vertical (VTO) topological orders, derived from a horizontal and vertical definite range graphs, is consistent. The input of the algorithm are two arrays of sub arrays representing the horizontal topological order from left to right and the vertical topological order from top to bottom. Each sub array, called *limit*, groups edges (borders of rectangles) that fulfill the *equivalence* relationship, which establishes that the edges of rectangles are the same when projected onto the corresponding axis. For each edge, variables $rectangle$ and $type$ indicate the rectangle to which the edge belongs and the type of the edge (i.e., left, right, top, or bottom). Finally, rectangles are numerated by consecutive integers.

Algorithm 1 Check consistency of topological orders

```
Input 1_topo_order: Topological order array, 2_topo_order: Topological order array
Output True or False
status← vector of integers; open_rec← 0
for all rectangle i in 1_topo_order do
    status[i] ← 0
for all limit L in 1_topo_order do
    for all edge e in L do
        if (e.type = left) ∨ (e.type = bottom) then
            status[e.rec]← 1; open_rec← open_rec+1
        else if (e.type = right) ∨ (e.type = top) then
            status[e.rec]← 0; open_rec← open_rec-1
    if open_rec> 1 then
        open_ortho_rect← 0
        for all limit L' in 2_topo_order do
            for all edge e' in L' do
                if ((e'.type = bottom) ∨ (e'.type = left)) ∧ status[e'.rec] = 1 then
                    open_ortho_rec← open_ortho_rec+1
                else if ((e'.type = top) ∨ (e'.type = right)) ∧ status[e'.rec] = 1 then
                    open_ortho_rec← open_ortho_rec-1
            if (open_ortho_rect> 1) then
                return False
return True
```