Efficient Process-Hotspot Detection Using Range Pattern Matching

H. Yao
Tsinghua University

S. Sinha, C. Chiang
ATG, Synopsys

X. Hong, Y. Cai
Tsinghua University

Abstract

In current manufacturing processes, certain layout configurations are likely to have reduced yield and/or reliability due to increased susceptibility to stress effects or poor tolerance to certain processes like lithography. These problematic layout configurations need to be efficiently detected and eliminated from a design layout to enable better yield. In this paper, such layout configurations are called *process-hotspots* and an efficient and scalable algorithm is proposed to detect such process-hotspots in a given layout.

The concept of a range pattern is introduced and used to accurately and compactly represent these process-hotspots. This representation is flexible and can incorporate information about the deficiencies of available modeling and/or subsequent correction (for instance, mask synthesis) techniques. Each range pattern can also be associated with a scoring mechanism to score the problem regions according to yield impact. A library of range patterns is being developed in collaboration with a fab. The proposed process-hotspot detection system assumes that process-hotspots are specified as a library of range patterns and determines all occurrences of any of these range patterns in a layout. It is fast and accurate and can be applied to large industrial layouts. Unlike previous work, the proposed scheme can identify problems that cannot be efficiently modeled or corrected by subsequent mask synthesis techniques and can thereby complement existing work in that area. Experimental results are quite promising and show that all locations that match a range pattern in a given layout can be found in a matter of minutes.

1. Introduction

Manufacturability-aware physical design (considering both yield and reliability) is becoming a necessity to bridge the gap between design and manufacturing for nanometer processes. Many of these yield and reliability issues can be attributed to the presence of certain layout configurations that are susceptible to stress, lithographic process fluctuations, etc. To improve yield, it is necessary to remove these configurations and replace them with more yield-friendly configurations.

A key technique needed to facilitate this is to accurately identify layout configurations in a given routed layout that are most susceptible to process issues, thereby potentially affecting functional and parametric yield. In the rest of this paper, these regions will be referred to as **process-hotspots**. Process-hotspots have to be represented accurately and succinctly in the router for later use. Typically, fabs use design rules (like recommended rules) to represent process-hotspots. The representation has been found to be inadequate due to the following reasons. Some of these effects are not necessarily very local. For instance, lithography effects involve interactions over longer distances (∼1um) than typical minimum spacing rules. Thus complicated relationships between non-neighboring objects need to be captured. While design rules have gotten more complex with the incorporation of multiple width and length dependent rules, it is still difficult to represent relationships between a large group of non-neighboring objects using a small set of rules. Relying entirely on design rules to represent all process-hotspots would result in an explosion of the number of design rules, which can significantly slow down the router. In some cases, certain recommended rules are ignored by the router for runtime efficiency. The inadequacy of design rules is further supported by the fact that DRC tools are being supplemented with more accurate process simulators (for instance, for lithography) during physical verification steps.

To address the limitations imposed by design rules, there has been a push to incorporate process models to analyze and/or drive corrections during routing. For instance, new work in the area of lithography-aware routing [1, 2] has proposed embedding an aerial image simulator in the router to identify process-hotspots. However, these approaches also have certain limitations.

- *Lack of knowledge of downstream steps*: It is impossible to accurately model certain downstream processing steps due to IP issues such as lack of knowledge of OPC recipes. On the other hand, simple aerial image based lithography simulations often tag regions that can be easily corrected using mask synthesis techniques as process-hotspots. This overestimation of hotspots is wasteful and can produce an unnecessary burden on the router.

- *Huge computational expense*: Certain process models are computationally expensive and hard to incorporate during physical design. For example, metal stress computations to determine layout configurations susceptible to stress can significantly slow down the router.

Thus it is not realistic to rely entirely on design rules and process models during physical design. Instead, we believe that a mechanism is necessary to directly represent the low-yielding layout configurations efficiently and accurately. Since effects like lithography and stress are non-local and often involve interaction distances that are a few microns long, a good representation of a process-hotspot would be a 2D layout of rectangles, i.e. a pattern. These patterns can be built off-line using test-structures or by running more accurate simulation tools. Thus, they can capture layout configurations that are prone to yield issues due to fundamental limitations of the mask synthesis techniques, incomplete or expensive modeling or undesirable interactions between processing steps. This ensures that the process-hotspot detection during routing would complement current advances in yield enabling steps like mask synthesis. While a pattern representation is accurate[1], it is usually too expensive to represent each problem region as a separate pattern. This is because a lot of these layouts/patterns are quite similar with minor variations in spacings, lengths and/or widths. Figure 1 shows an example where two similar layouts could become process-hotspots due to different RET constraints (the one on the left is "un-OPC-able" and the one on the right is "un-SRAF-able"). In the pattern on the left, a serif cannot be added in some technologies without violating mask design

[1] In some recent work [5], pattern matching has been used to speed up some exact simulation tools.

ICCAD 2006, Nov 5–9 2006, San Jose, USA

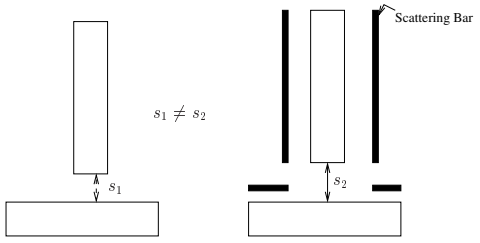


Figure 1. Similar layouts that are process-hotspots.

rules. A scattering bar cannot be inserted in between the two rectangles for the pattern on the right without violating design rules. The two patterns are very similar, except for different spacings between the rectangles. In reality, each of these patterns is representative of a set of patterns as the same problem could occur for a range of s_1 and s_2 values. Thus, multiple similar patterns will have to be represented if exact patterns are used.

To address the above mentioned problems, we propose to succinctly represent a group of "similar" layouts with allowable variations in length, width and/or spacing using a new representation called the **range pattern**. *The use of patterns ensures that non-local effects are captured and the ranges in the pattern enable compact representation.* The range patterns can be built either in collaboration with a fab or with in-house accurate simulation and mask synthesis flows to identify layout configurations that could have yield and/or reliability issues. An user-modifiable scoring mechanism provided with each range pattern can be used to score the patterns in the set based on yield impact. This information can be used by the router to give higher priority during correction to the problem regions with greater yield impact. Thus, in the proposed scheme, process-hotspots are represented as a library of range patterns. **Process-hotspot detection** is the task of finding all the locations where the layout is identical to one of the patterns contained in a range pattern in the library. The key contributions of this work are summarized below:

- *Introduction of the concept of a range pattern and the associated range pattern matching problem and a novel and efficient algorithm for the same.*
- *Representation of process-hotspots as a library of range patterns. Since the library is built off-line or provided by the fab, they can capture information about the true yield killers more accurately.*
- *A fast, accurate and scalable algorithm for process-hotspot detection during physical design using multiple calls to an efficient range pattern matching algorithm.*

The paper is organized as follows. In Section 2, the concept of a range pattern and the associated range pattern matching problem is introduced. In addition, efficient representations of the routed layout and range patterns are discussed. In Section 3, a detailed description of the proposed process-hotspot detection system is provided. Experimental results for multiple range patterns on real industrial layouts are presented in Section 4. Finally, conclusions and directions for future work are provided in Section 5.

2. Range Patterns and Matching

In this section, we formally introduce the concept of a range pattern and the associated matching problem. Suitable representations of the layout and range pattern for use during range pattern matching are also discussed.

2.1 Range Pattern Definitions

DEFINITION 1. *A* **range pattern** *is a DRC-correct two-dimensional layout of rectangles with additional specifications:*

1. *Widths and lengths of the rectangles can vary within certain user-specified bounds.*
2. *Spacing between pairs of rectangles can vary within certain user-specified bounds.*
3. *Optimal widths and lengths of the rectangles and optimal spacings between pairs of rectangles can be specified.*
4. *Constraints can be specified over linear combinations of the widths, lengths and spacings of the rectangles.*

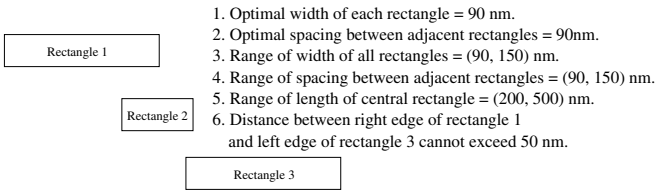


Figure 2. Range pattern `Staircase`.

Figure 2 shows an example of a range pattern called `Staircase` with the following specifications:

1. *Optimal width of each rectangle = 90 nm.*
2. *Optimal spacing between adjacent rectangles = 90 nm.*
3. *Range of width of all rectangles = (90, 150) nm.*
4. *Range of spacing between adjacent rectangles = (90, 150) nm.*
5. *Range of length of central rectangle = (200, 500) nm.*
6. *Distance between the right edge of rectangle 1 and the left edge of rectangle 3 cannot exceed 50 nm.*

It is clear that the above range pattern contains a multitude of exact patterns. Thus, a range pattern is a compact representation of a set of "similar" patterns. A scoring mechanism can also be introduced for the range pattern. The scoring mechanism typically differentiates between the various patterns contained in a range pattern based on the differences in lengths, widths and/or spacings. For instance, a pattern where the line widths are 90 nm is given a higher score than a pattern where the line widths are 110 nm as the former is likely to have greater printability problems and hence worse yield. The scoring model for each range pattern can be developed using some representative patterns such that the differences in yield of the patterns contained in a range pattern can be captured using polynomial functions of the range pattern parameters (like width, spacing, length, etc. of the rectangles).

The **Range Pattern Matching (RPM)** problem can be stated as follows: Given a layout and a range pattern, determine all occurrences of the range pattern in the layout and score these occurrences using the scoring mechanism for the range pattern.

Once these match locations are found, the router can work to correct them. The scores of the various occurrences of the range pattern can be used to guide the router to correct the most yield-critical ones first.

2.2 Layout Representation

During RPM, the layout is represented by a two-dimensional layout matrix $L_{N_1 \times N_2}$ where $L[i,j] = 0$ or $1 (0 \leq i < N_1$ and $0 \leq j < N_2)$. The conversion is as follows: If a rectangle overlaps a grid location, the value at that location is set to 1. Otherwise, the value of the grid is set to 0. Figure 3 illustrates with an example the representation of a layout as a layout matrix.

2.3 Range Pattern Representation

If the range pattern specification is such that it only represents a small set of exact patterns, matrix representations for each individual pattern can be used. In that case, existing pattern matching algorithms (for example, the algorithm in [3]) would suffice to find all occurrences of the range pattern. However, it would be too computationally expensive to explicitly represent all patterns contained in a general range pattern. To address this issue, we propose a new representation called the

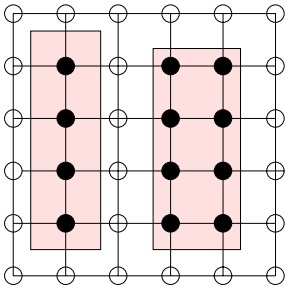


Figure 3. Representation of layout as layout matrix.

cutting-slice to efficiently represent all the flexibility inherent in a range pattern during RPM. The theory and the specific algorithm for automatically translating the range pattern into its cutting-slice representation is given in the Appendix. Below, we define a cutting-slice representation and provide an intuitive explanation of how a range pattern can be completely represented with one or more cutting-slices. Typically, the number of cutting-slices is much smaller than the number of exact patterns.

DEFINITION 2. *A* **horizontal(vertical) slice** *is a 2D matrix in which all the rows(columns) are equal. A* **fragment** *of a horizontal or vertical slice is a sub-matrix in which all the matrix elements are equal.*

DEFINITION 3. *A* **cutting-slice** *is a set of horizontal or vertical slices* $\{S_0, \cdots, S_{n-1}\}$ *with the following specifications:*

1. *Adjacent slices are not equal, i.e.* $S_i \neq S_{i+1}, 0 \leq i \leq (n-2)$.
2. *Each slice* S_i *is decomposed into fragments* $\{F_{i,0}, \cdots, F_{i,m-1}\}$, *where* $F_{i,j} \neq F_{i,j+1}, 0 \leq j \leq (m-2)$.
3. *If applicable, optimal values are specified for the fragments in each slice and for the slices themselves.*
4. *If applicable, ranges are specified for each slice and/or fragments within the slice.*
5. *If applicable, constraints between different fragments and/or slices are specified as linear functions.*

The optimal values, ranges and constraints can be given on an absolute scale (i.e. in microns or nanometers) or in terms of the number of grids. Typically, the same grid will be used to translate the layout into a layout matrix and to generate the cutting-slices of a range pattern.

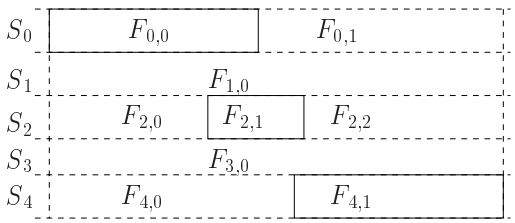
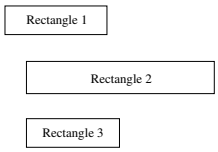


Figure 4. Cutting-slice of range pattern `Staircase`.

For example, the cutting-slice of range pattern `Staircase` (Figure 2) is shown in Figure 4. The range pattern can be cut into 5 slices denoted as $S_0, S_1, \cdots, S_4$. Note that all the fragments in the i^{th} slice S_i have the same width as S_i (i.e. slice width) but their lengths can vary. It is possible to specify the variation range for the length of each fragment and the width of each slice. In the sequel, S_i can denote either the slice or the slice width; $F_{i,j}$ can denote either the fragment or the fragment length. For example, based on the specification of the range pattern, the optimal width of S_1 is 90 nm and the allowable variation range is (90, 150) nm. In addition, it is also possible to specify constraints between different fragments. In this particular example, the absolute distance between the right edge of rectangle 1 and the left edge of rectangle 3 cannot exceed 50 nm. This translates to the following linear inequality: $|F_{0,0} - F_{4,0}| \leq 50$.



1. Optimal width of each rectangle = 90 nm.
2. Optimal spacing between the rectangles = 90 nm.
3. Range of width = (90, 150) nm.
4. Range of spacing = (90, 150) nm.
5. Rectangle 1 and rectangle 3 can be of unequal length.
6. The right end of Rectangle 2 is (50, 250) nm away from the right ends of rectangle 1 and rectangle 3 on one side.

Figure 5. Range pattern `Rocket`.

It should be noted that the number of cutting-slices required to fully capture all the patterns contained in a range pattern depends on the slicing direction, i.e. the direction used to generate the slices. The number of cutting-slices for a given slicing direction depends on the number of overlaps of different fragments caused by the ranges on their dimensions. Only ranges on dimensions that are orthogonal to the slicing direction need to be considered. Overlaps between two fragments can be classified into two categories:

1. **Uni-directional overlap**: The two fragments extend in the same direction and their ranges may cause overlap. Rectangles *1* and *3* of `Rocket` in Figure 5 are an example of uni-directional overlap for a vertical slicing direction.
2. **Bi-directional overlap**: The two fragments extend in the opposite direction and their ranges may cause overlap. Rectangles *1* and *3* of `Staircase` in Figure 2 present an example of bi-directional overlap for a vertical slicing direction.

For both uni-directional overlap and bi-directional overlap between two fragments, three cutting-slices are needed to completely represent all possible scenarios for these two fragments. As an example, for range pattern `Rocket` in Figure 5, three cutting-slices are required in the vertical slicing direction (Figure 6(a)-Figure 6(c)), whereas only one cutting-slice is required in the horizontal slicing direction (Figure 6(d)) to capture all the patterns contained in the range pattern. This is because the allowable length variations of Rectangles 1 and 3 (Item 5 of the specification) allow three cases: (a) Rectangle 1 is shorter than Rectangle 3; (b) Rectangle 1 is equal to Rectangle 3; and (c) Rectangle 1 is longer than Rectangle 3. If a vertical slicing direction is used, three different cutting-slices will be needed to fully capture the flexibility. Figure 6(a-c) illustrates the scenario. On the other hand, a single cutting-slice will suffice when a horizontal slicing direction is used (Figure 6(d)), since neither uni-directional nor bi-directional overlap occurs in this case.

The total number of the cutting-slices needed for a given range pattern is calculated by enumerating all the overlapping cases, the details of which are given in the Appendix. Typically, the slicing direction that results in the least number of cutting-slices is chosen.

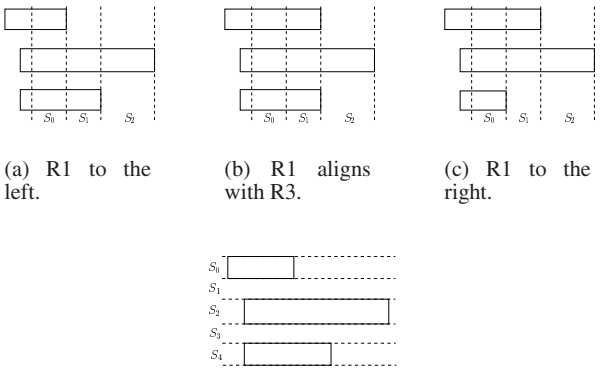
3. Process-Hotspot Detection System

In this section, we present the details of the process-hotspot detection system. The input for the system is the routed layout and a library of range patterns that describes process-hotspots. The routed layout is processed layer by layer.

3.1 Overview

The flowchart in Figure 7 describes the process-hotspot detection system with a single range pattern for one layer. This process can be easily generalized to multiple range patterns.

The algorithm uses a hierarchical dual-grid scheme with matching done on two grid sizes (one coarse and the other much finer). The grid sizes are used to generate the layout matrices and the cutting-slices of



(a) R1 to the left.

(b) R1 aligns with R3.

(c) R1 to the right.

(d) Horizontal slicing direction.

Figure 6. Horizontal and vertical cutting-slices of range pattern `Rocket`.

a range pattern for each stage. Matching with the coarse grid identifies locations that are potential matches for the range pattern. In the next stage, the layout matrix of the layout at each of the match locations and the cutting-slices of the range pattern are generated using a finer grid and the matching process is repeated. The match locations identified at this stage are the locations where a true match to the range pattern exists and hence are true process-hotspots. Typically, the fine grid size is equal to the manufacturing grid size. The algorithm is executed on both the original and a 90°-rotated layout.

3.2 Range Pattern Matching Sub-Problem

In this section, we discuss the solution for the RPM problem for a given range pattern and a given window of the layout, which is represented as a layout matrix. The matching problem is invoked both for the original range pattern and its 180°-rotated version. For ease of presentation, we discuss the solution for the original range pattern and also assume that a single cutting-slice in the vertical slicing direction can completely represent it.

The matching algorithm is performed block by block. Here a block refers to a sub-matrix of the layout matrix. The number of columns in the block is equal to the width of the layout matrix. The height of the block is equal to h, where $min \leq h \leq max$. Here, min and max denote the minimum and maximum possible number of rows of the range pattern, respectively (the height of the range pattern is not unique as the widths, lengths and spacings can vary). The first block starts from the bottom row of the layout matrix.

A naive approach would look for potential matches at each location of a block. This would make the task computationally infeasible. Instead, a fast filtering operation is first performed at each block to efficiently filter out locations that can never be matches to the range pattern. The details of the filtering algorithm are provided in the next section. It is proved that this operation never filters out locations that are true matches of the range pattern and hence does not result in the loss of any true matches of the range pattern in the window. All the locations that are not filtered out are examined more closely to determine if they are true matches. To achieve this, the layout matrix near the match location is decomposed into slices and a thorough comparison is done between the slices of the layout matrix and the slices in the cutting-slice of the range pattern. This includes a check of the constraints on the fragments of each slice or between slices as well as constraints on the slices themselves. If the location passes the verification phase during the coarse-grid stage, it is a potential match and is re-examined during the fine-grid stage. If the location passes the verification during the fine-grid stage, a true match is recorded. In addition, a matching score is computed based on the cost function provided with

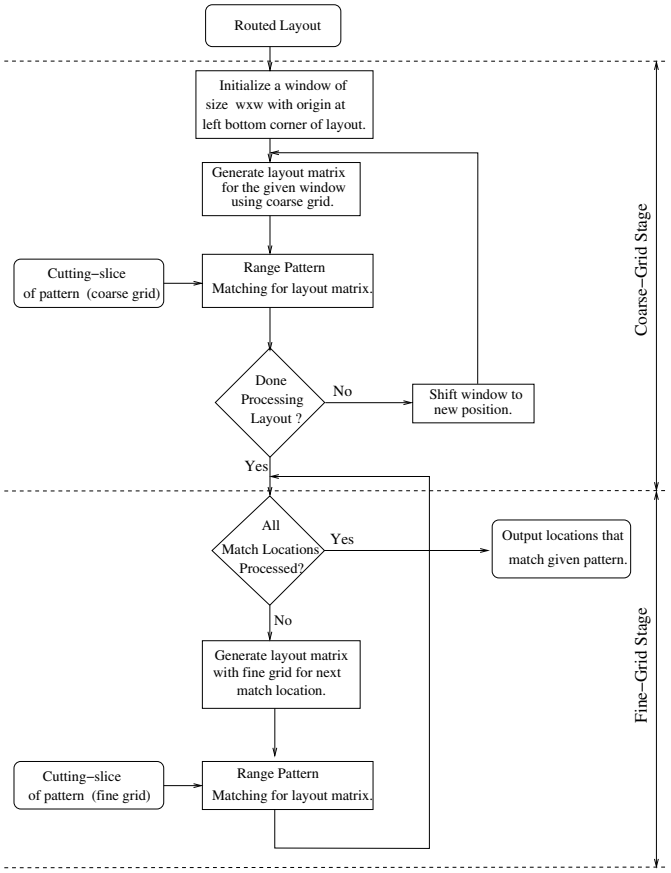


Figure 7. Flowchart of the hotspot detection system.

the range pattern. The cost function is a function of the following variables in the given range pattern: the optimal values and the lower and upper bounds of widths, lengths and spacings. The calculated cost is typically translated to the matching score between 1 and 100.

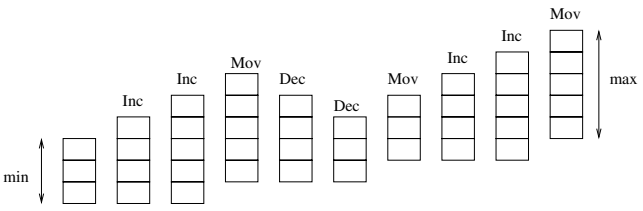


Figure 8. Worm-like movement of the layout block.

It is necessary to enumerate all the blocks whose heights are between min and max starting from each row of the layout matrix to find all the occurrences of the range pattern without loss of matches. However, these blocks share a lot of common information. In order to reuse work done in encoding the previous block, the blocks are processed in a worm-like fashion such that only the top and the bottom rows are changed each time (Figure 8). The first block is of height min starting from the bottom row. Each time a new block is generated by either adding a row at the top, removing a row from the top or moving the whole block up by one row when the height of the block reaches min or max. It ensures that all heights between min and max are enumerated

for all blocks that start at a row in the layout matrix. This worm-like enumeration enables incremental encoding of the block, thereby greatly improving runtime.

3.2.1 KMP-based Filter

The first step of the filtering operation is to encode both the block $\mathcal{B}$ and the cutting-slice $\mathcal{C}$ of the range pattern as 1D strings. Let the string representations of $\mathcal{B}$ and $\mathcal{C}$ be $\mathcal{B}_E$ and $\mathcal{C}_E$, respectively. Given $\mathcal{B}_E$ and $\mathcal{C}_E$, a KMP matching [4] is done to find all potential matches of $\mathcal{C}_E$ in $\mathcal{B}_E$. All locations that are not matches are filtered out. The locations that match are mapped back to locations in the block and are examined more closely using a secondary check (based on slice-by-slice and fragment-by-fragment matching) to determine if they are true matches.

The block and cutting-slice encoding are done as follows:

DEFINITION 4. *The* **run-length compression** *of a column $C[j][N]$ is equal to $\{b_0, b_1, \cdots, b_{n-1}\}$, where: (1) $b_i \neq b_{i+1}$ $(0 \leq i < n-1)$; (2) $C[j][N]$ can be represented as a concatenation of n segments, i.e. b_0 repeated σ_0 times, b_1 repeated σ_1 times, and so on; (3) $\sum_{i=0}^{n-1} \sigma_i = N$.*

For the range pattern, the run-length compression of each vertical slice (a vertical slice is uniquely represented by a single column) is generated. This is a string of alternating 0's and 1's. A "1" is appended at the top of each string generated after run-length compression to distinguish between strings "01" and "1". Each string is encoded into an integer value using binary encoding[2] . Encoding each slice converts the cutting-slice of the range pattern into a string of numbers, where the length of the string is equal to the number of slices in the cutting-slice.

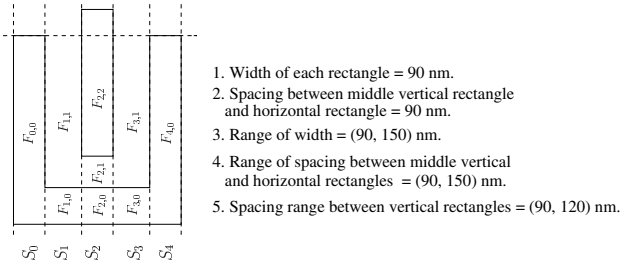


Figure 9. Cutting-slice of range pattern Mountain.

As an example, the slices in the cutting-slice of the range pattern `Mountain` (Figure 9) are encoded as follows: S0: "11" = 3; S1: "101" = 5; S2: "1101" = 13; S3: "101" = 5; S4: "11" = 3; The 1D string representation of the range pattern is $\{3, 5, 13, 5, 3\}$.

The block is encoded in a similar fashion. First, the slices in a block are identified in one sweep starting from the left end of the block. The slices are created such that no two adjacent ones are equal. Then, the run-length compression of each slice is generated and each slice is encoded using the same method used for the slices in the cutting-slice of the range pattern. Using this method, the 1D string representation of the layout in Figure 10 is $\{2, 10, 10, 2, 3, 5, 13, 5, 3, 2, 10, 10, 2\}$. It is easy to see that there is an exact match of the encoded range pattern $\{3, 5, 13, 5, 3\}$ in the encoded block. Hence, columns 5-14 of the block in Figure 10 are examined more closely to see if it is a true match. The remaining locations can never be true matches and are filtered out.

THEOREM 1. *The filtering algorithm satisfies the following conditions: Let C denote the cutting-slice of a range pattern P_R. For every occurrence of the original range pattern P_R in the original block B, there is an occurrence of the encoded cutting-slice $\mathcal{C}_E$ in the encoded block $\mathcal{B}_E$.*

[2]It is possible to use any encoding scheme that can represent each string of alternating 0's and 1's with a unique number.

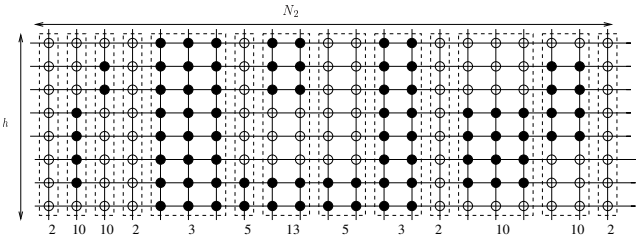


Figure 10. Layout block encoding.

Proof Sketch: We prove the theorem for the case when the range pattern P_R has a single cutting-slice C and for the case when the occurrences of P_R in B have the same orientation.

The block encoding has two steps: (1) block slicing, (2) run-length compression of each slice. The block slicing process moves horizontally from one end to the other identifying vertical slices such that no two adjacent slices are equal. By definition, a vertical slice is a 2D matrix where all the columns are equal. By definition of a cutting-slice, the slices in C also have the property that no two adjacent slices are equal. Since the range pattern P_R has a single cutting-slice C, the block slicing ensures that all occurrences of P_R in B will be replaced by slices that are identical to the slices in the cutting-slice C. Furthermore, since the same run-length compression is used for each slice of the block, both occurrences of C in the sliced block will have the same encoding and will be represented by the same string in $\mathcal{B}_E$. In addition, this string is equal to $\mathcal{C}_E$ since the same run-length compression is also used on the slices in C to generate $\mathcal{C}_E$. □

The same can be shown to hold if a range pattern P_R has multiple cutting-slices. Thus, no true matches are lost during the filtering operation. However, it is not necessarily true that each occurrence of the encoded range pattern in the encoded block implies a true match.

3.2.2 Complexity Analysis of the RPM Algorithm

Given a layout block $B_{h \times N_2}$ where $min \leq h \leq max$, the different slices are first identified by comparing $B[i, j]$ with $B[i, j+1](0 \leq i < h$ and $0 \leq j < N_2 - 1)$, which takes $O(h \times (N_2 - 1))$ time. Let the number of different identified slices in the layout block be s $(1 \leq s \leq N_2)$. Then the run-length compression and the binary encoding for the layout block runs in time $O(h \times s)$. The KMP string matching algorithm takes $O(s)$ to find the potential matches and the verification process for each potential match is of $O(1)$ time. Therefore, the time complexity of the RPM algorithm for one layout block is $O(h \times N_2)$. According to the block enumeration strategy, the total number of different blocks in layout matrix $L_{N_1 \times N_2}$ is less than $(N_1 - min + 1) \times (max - min + 1)$. So the total time complexity of the RPM algorithm in the whole layout matrix without the worm-like enumeration technique is $O(max \times N_2 \times (N_1 - min + 1) \times (max - min + 1))$.

In the worm-like enumeration technique, only the top row and bottom row of the new block need to be checked to incrementally update the identified slices and the run-length compression result. The time complexity of slice identification is reduced to $O(N_2)$ and that of run-length compression is $O(s \times U)$, where U is the average time for updating the run-length compression of each slice. U is not a constant as it depends on the number of fragments of each slice. The incremental binary encoding runs in $O(s)$, which is typically smaller than $O(N_2)$. Thus, the total runtime complexity using the worm-like enumeration strategy is $O(max(N_2, s \times U) \times (N_1 - min + 1) \times (max - min + 1))$.

From the above analysis, it can be observed that the dimension of the layout matrix (N_1 and N_2) and the variation range in the height of the cutting-slice (min and max) are the key factors which affect the runtime of the RPM algorithm.

3.3 Scalability and Runtime Optimization

In most practical cases, a direct translation of the entire layout into a layout matrix is impossible. Hence, the hotspot detection system

works window-by-window in an incremental fashion to cover the entire layout and an RPM algorithm is executed for each window. It is necessary to ensure that consecutive windows overlap to avoid loss of matches. The amount of overlap between adjacent windows depends on the maximum possible size of the range pattern. If the maximum possible width and height of the range pattern are max_w and max_h, respectively, the amount of horizontal overlap between two consecutive windows should be $(max_w + 1)$ and the amount of vertical overlap should be $(max_h + 1)$, respectively. In the implementation, the size of the window is much larger than that the maximum dimension of the range pattern on either side. However, it should not be too large to avoid excessive memory consumption.

The grid size used for generating the layout matrix and the cutting-slices of a range pattern can greatly impact the runtime.

THEOREM 2. *Let* single-grid hotspot detection *consist of overlapping window generation to cover the entire layout along with RPM within each window for a given range pattern. If the layout matrix and the cutting-slices of a range pattern are generated using the manufacturing grid, then* single-grid hotspot detection *can determine all occurrences of the range pattern in the layout.*

Proof Sketch: Assume for the sake of simplicity that the range pattern has a maximum dimension of max on both sides. We begin by proving that the single-grid hotspot detection can find all matches of a range pattern contained in a block B_i. Since the grid is generated using the manufacturing grid, hence all the layout rectangles and the rectangles in the range pattern snap to the grid. Hence, no rounding error is introduced in the generation of the layout matrix or the cutting slices. This combined with Theorem 1 ensures that all matches contained within a block are located.

Next, we need to prove that all occurrences of the range pattern in a given window W_i are identified by the single-grid scheme. Let b_i and l_i denote the bottom and left coordinates of W_i, respectively. Let $|W|$ denote the size of W_i in either direction. There are two cases:

1. Matches that start between (l_i, b_i) and $(l_i + |W| - (max + 1), b_i + |W| - (max + 1))$: The worm-like movement ensures that starting from each row of the layout matrix LM_i (which is essentially a digitization of the rectangles in W_i) all blocks of height between min and max (min and max denote the minimum and maximum height of the range pattern, respectively) are enumerated. The size of the W_i is much larger than max in either direction. Since all range pattern occurrences within a block are identified and all possible blocks are enumerated, this ensures that all range pattern occurrences in a window that start between (l_i, b_i) and $(l_i + |W| - (max + 1), b_i + |W| - (max + 1))$ will be completely contained in W_i and found during the RPM call in W_i.
2. Matches that start from $(l_i + |W| - max, b_i + |W| - max)$: By construction, W_i overlaps with two windows W_h (W_h is to the right of W_i) and W_v (W_v is above W_i) such that W_i and W_h overlap in the horizontal direction by $(max + 1)$ and W_i and W_v overlap in the vertical direction by $(max + 1)$. Hence a range pattern that starts within max grids of the right edge of W_i will be captured during RPM in W_h. Similarly, a range pattern that starts within max grids of the top boundary of W_i will be completely contained in W_v and hence will be found when RPM is called on W_v. Thus, all range pattern occurrences in W_i will be identified. □

It should be noted that the single-grid hotspot detection system can find redundant matches, i.e. multiple matches are reported for the same occurrence. A simple redundancy checking algorithm based on sorting the coordinates of the bottom left corners of the reported locations can eliminate such duplicate matches. The runtime for directly finding matches on the whole layout using a fine grid size is typically slow. Hence, a hierarchical matching strategy is adopted to speedup the program. To begin with, matching is done on the coarse grid. This means the layout matrix and the cutting-slices are generated using a coarse grid. During this stage, potential match locations can be quickly identified. However, this stage could introduce rounding errors since it is not necessary for every rectangle in the layout and the range pattern to align with the grid. To counter this, the constraints in the range pattern are typically relaxed (i.e. a wider range is allowed) during this stage. Then, a second round of matching is carried out on the match locations found in the coarse-grid stage. This check is typically done using a much finer grid (usually equal to the manufacturing grid) to eliminate errors due to rounding issues or over-relaxation of constraints. It should be noted that the coarse grid size has to be less than the smaller of the two: the minimum width or the minimum spacing of the layer. Otherwise, neighboring features might merge resulting in an incorrect translation of the layout into a layout matrix. However, a very large coarse grid size can result in a large number of potential matches and drive up the cost of the fine grid validation step. In practice, a grid size that is equal to half the allowable coarse grid size is found to provide the right balance in terms of runtime.

Table 1. Range Pattern Characteristics

Range Pattern	# Rects.	Overlap?	Multiple Patterns?
Bird	5	Yes	Yes
Bridge	6	Yes	Yes
Weave	4	No	Yes
Zigzag	3	Yes	Yes

4. Experimental Results

We have tested the proposed process-hotspot detection system on a Linux 2.4 system with two 2.2 GHz CPUs and 2 GB RAM (only a single CPU was used for the experiments). Five layouts were used in the experiments: $D1$, $D2$ and $D3$ are three metal layers of a $0.6 \times 0.6mm^2$ design; $D4$ and $D5$ are two metal layers of a $1.8 \times 1.8mm^2$ design. Both designs are $65nm$ designs. The process-hotspot library has the range patterns we discussed in the paper as well as some additional range patterns that were provided by a fab. Since we are unable to divulge the details of the range patterns obtained from the fab, we characterize them using three criteria to give a sense of their complexity: number of rectangles, whether the rectangles overlap and whether they contain multiple exact patterns (typically each range pattern specified by the fab has ranges on multiple dimensions resulting in a large number of exact patterns). The key characteristics of the range patterns obtained from the fab are summarized in Table 1.

The library was tried on each of the designs. For the sake of brevity, only the designs where matches were found for a particular range pattern are presented. For the hierarchical matching strategy, the coarse grid size is set to be 50 nm and the fine grid size is equal to the manufacturing grid of the design.

The results for the library are shown in Table 2. Columns *Range Pattern* and *Design Name* identify the range pattern and design, respectively. The number of locations in the layout that match each range pattern and the runtime for finding all matches are shown in Column *# of Matches* and Column *Runtime(s)*. The data for both the hierarchical scheme and the single-grid scheme (where the grid is equal to the manufacturing grid) is presented. Column *Score Range* shows the score range of the matches in the layout with the single-grid detection scheme.

4.1 Discussion

The range in scores strengthens our claim that many similar patterns can exist in a given layout. Conventional exact pattern matching solutions would require multiple invocations to find all these patterns. By Theorem 2, the single-grid scheme where the grid size is equal to the manufacturing grid should find all the matches of a given range pattern. Comparing the number of matches of the hierarchical scheme with the single-grid scheme shows that the hierarchical scheme can identify all the matches at a fraction of the runtime. This indicates that the proposed hierarchical scheme is quite accurate and has a very low likelihood of missing any potential match locations in the layout. The runtimes for the hierarchical matching strategy range between a few seconds and 6 minutes for each range pattern, whereas the runtimes for single-grid detection are much higher. The maximum memory used for matching all the range patterns is $\sim 21MB$.

Table 2. Process-Hotspot Detection Results

Range Pattern	Design Name	# of Matches		Runtime (s)		Score Range
		Hier.	Single-Grid	Hier.	Single-Grid	
Bird	D1	212	212	156.43	3189.94	[87.81,97.29]
Bird	D2	52	52	14.86	2166.80	[85.99,96.84]
Bird	D3	5	5	15.75	2862.74	[93.22, 96.84]
Bird	D4	5480	5480	264.07	15933.31	[80.11, 96.84]
Bird	D5	36	36	73.24	13397.48	[80.56, 92.01]
Bridge	D1	2062	2062	137.98	11517.72	[98.32,98.74]
Weave	D4	14	14	358.59	17694.92	[93.40, 95.88]
Weave	D5	2	2	83.46	16036.56	[93.40, 93.40]
Zigzag	D2	2474	2474	19.47	2142.57	[93.23,98.73]
Zigzag	D3	1642	1642	13.31	3130.89	[97.46,97.46]
Zigzag	D4	12939	12939	358.59	14888.53	[93.23, 98.73]
Zigzag	D5	3038	3038	95.08	12878.50	[97.46, 97.46]
Mountain	D4	10	10	157.99	16598.91	[91.00, 92.50]
Staircase	D4	349	349	188.11	22865.83	[99.15, 99.43]

The above results show that the proposed scheme is a promising approach and can be embedded in the router to efficiently detect process-hotspots during routing. Once the problem locations are identified, local wire-spreading and/or widening can be attempted to break the particular occurrence of the range pattern. If local solutions do not suffice, the connections can be removed and new re-routings attempted. To avoid returning to the same solution, new DRC rules based on the constraints of the range pattern can be added when new routings are being explored in the area.

5. Conclusion

In this paper, we propose the concept of representing process-hotspots as range patterns, which are compact representations of a set of similar layouts. The use of patterns ensures that non-local lithography or stress effects are captured and the ranges enable compact representation. [6]

The range pattern matching problem is also introduced and an efficient algorithm is proposed for the same. A process-hotspot detection system, based on range pattern matching, to efficiently and accurately find and score process-hotspots in a given layout is also presented. The system guarantees no false alarms since only patterns that match one of the range patterns in the library are located. The proposed algorithm is scalable and can work on large layouts, thereby making it a practical scheme for efficiently detecting process-hotspots during routing and/or physical verification. Experimental results indicate that the runtimes are quite small, which makes it possible to embed it in a router for post-routing optimizations. The algorithm is being extended to handle range patterns with "don't care" regions, i.e. regions in the range pattern that do not require user specification and allow any combination of rectangles to exist.

We are currently investigating algorithmic correction schemes to eliminate and/or reduce the occurrences of the detected process-hotspots at the routing stage. Another noteworthy direction for future work would be to investigate if already existing recommended rules can be combined and compactly represented using fewer range patterns, thereby reducing the runtime burden on routers and/or physical verification tools. In addition, more thorough comparisons with model-based approaches are necessary and are being done as a part of future work.

6. References

[1] L.-D. Huang and M. D. F. Wong. Optical proximity correction (opc)-friendly maze routing. In *DAC*, pages 186–191, June 2004.

[2] J. Mitra, P. Yu, and D. Z. Pan. RADAR: Ret-aware detailed routing using fast lithography simulations. In *DAC*, pages 369–372, June 2005.

[3] R. Zhu and T. Takaoka. A technique for two-dimensional pattern matching. In *Comm. ACM, 32(9)*, pages 1110–1120, 1989.

[4] D.E. Knuth, J.H. Morris and V.R. Pratt. Fast Pattern Matching in Strings. *Siam Journal on Computing 6(2)*, pages 323–350, 1977.

[5] http://www.commandcad.com.

APPENDIX

Here, we describe how the range pattern has to be specified by the user and an algorithmic procedure to convert the specification into one or more sliced patterns. A typical range pattern specification includes: (1) Edges at the left and bottom boundary of the range pattern. (2) Ranges (with or without optimal values) or absolute values between rectangle edges. These ranges in (2) could either specify ranges in the length, width or spacing as well as additional constraints.

DEFINITION 5. *A* **Range Graph** G *is a quadruple* (V, E, ψ, ω) *where* V *and* E *are finite sets,* $\psi : E \rightarrow \{(v, w) \in V \times V; v \neq w\}$ *and* $\omega : E \rightarrow \{(m, n) \in R \times R; m \leq n\}$. *The elements of* V *are vertices, the elements of* E *are edges and the elements of* R *are real numbers.* G *satisfies the condition that whenever there is an edge* $e = (v, w) \in E(G)$ *with* $\omega(e) = (m, n)$, *there is also an edge* $\tilde{e} \in E(G)$ *where* $\tilde{e} = (w, v)$ *and* $\omega(\tilde{e}) = (-n, -m)$. *m(n) is denoted as min(e)(max(e)) and is called the lower (upper) bound of edge* e. *The* **range** *of* e *is* $|\omega(e)| = (n - m)$. *A range graph* $G = (V, E, \psi, \omega)$ *is called* **stable** *iff the following condition is satisfied: for each edge* $e \in E(G)$, *range of* e *is finite and minimized.*

The specification for a range pattern RP is first converted to two range graphs: one for the horizontal edges of the rectangles henceforth denoted as HRG and another for the vertical edges of the rectangles henceforth denoted as VRG. Thus, each rectangle edge r_i becomes a vertex v_i in the corresponding range graph and an edge e_{ij} exists in the range graph between any two vertices v_i and v_j, where $i \neq j$. If the user has specified a range $= (d_1, d_2)$ and/or an absolute distance d between rectangle edges r_i and r_j, then $\omega(e_{ij})$ is set to (d_1, d_2) (or (d, d)). Otherwise, $\omega(e_{ij})$ is set to $(-\infty, \infty)$. Then, the All-Pair Min-Range Path (APMRP) algorithm (as outlined below) is invoked. The APMRP algorithm works as follows:

1. For k = 0 to ($|V| - 1$), For i = 0 to ($|V| - 1$), For j = 0 to ($|V| - 1$),
   (a) If $(i \neq j)$ and $(j \neq k)$ and $(i \neq k)$
      i. If $(max(e_{ik}) < \infty)$ and $(max(e_{kj}) < \infty)$ and $\{max(e_{ik}) + max(e_{kj}) < max(e_{ij})\}$
         A. $max(e_{ij}) = max(e_{ik}) + max(e_{kj})$.
      ii. If $(min(e_{ik}) > -\infty)$ and $(min(e_{kj}) > -\infty)$ and $\{min(e_{ik}) + min(e_{kj}) > min(e_{ij})\}$
         A. $min(e_{ij}) = min(e_{ik}) + min(e_{kj})$.

The lower bounds of some edges can increase and the upper bounds of some edges can decrease after the application of the APMRP algorithm

to a range graph G. If the range of an edge $e \in E(G)$ becomes negative or unbounded after the application of the APMRP algorithm, it can be concluded that the specification for RP is invalid and needs to be revised. It can be proved that in all other cases (i.e. when the range pattern specification is valid) the APMRP algorithm finds a stable range graph G' for the given input range graph G.

DEFINITION 6. *Edge $e \in E(G)$ where $\omega(e) = (m, n)$ is called* **indefinite** *iff the following conditions are satisfied: $m \neq n$; $m \leq 0$ and $n \geq 0$. Otherwise, e is called* **definite**.

An indefinite edge $e \in E(G)$ with $\omega(e) = (m, n)$ **contains** a set of definite ranges. If $m < 0$ and $n > 0$, then there are three definite ranges: $\{(m, -1), (0, 0), (1, n)\}$. If $m < 0$ and $n = 0$, then there are two definite ranges: $\{(m, -1), (0, 0)\}$. If $m = 0$ and $n > 0$, there are two definite ranges: $\{(0, 0), (1, n)\}$.

DEFINITION 7. *A stable range graph is* **definite** *iff each edge is definite.*

It is not necessary that a definite stable graph is obtained after the application of the APMRP algorithm. Given an indefinite stable range graph G^s, it is necessary to convert the indefinite edges of G^s into definite ones to determine the unique topological orders of the rectangle edges. Each topological order corresponds to a cutting-slice. The algorithm, **Enum_DRG**, described below takes a indefinite stable range graph G^s corresponding to RP and outputs all definite range graphs contained in it.

1. Invoke APMRP(G^s) to update the ranges of each edge in G^s.
2. Set flag allEdgeDefinite = TRUE.
3. For each edge $e = (v, w) \in E(G^s)$, do:
   (a) If e is indefinite, then
      i. For each definite range $r = (min, max)$ of e, do:
         A. Set $\omega(e(v, w)) = (min, max)$ and $\omega(\tilde{e}) = (-max, -min)$, where $\tilde{e} = (w, v)$ denotes the edge from w to v.
         B. Invoke Enum_DRG on the modified G^s.
      ii. Set allEdgeDefinite = FALSE.
   (b) If allEdgeDefinite == FALSE, then break.
4. If allEdgeDefinite == TRUE, then Output definite range graph G^d.

The cutting-slices needed to represent a range pattern can be derived from the definite stable range graphs obtained using **Enum_DRG**. Next, we provide some theory to explain why this is the case.

DEFINITION 8. *Given a definite edge $e = (v, w) \in E(G)$, where $\omega(e) = (m, n)$, vertex v is said to* **precede** *vertex w iff $m > 0$. Two vertices are* **equal** *iff $m = n = 0$. If vertex v precedes vertex w, then vertex v and vertex w are said to satisfy the* **precedence** *relation R_p denoted vR_pw. If vertex v equals vertex w, then vertex v and w are said to satisfy the equivalence relation denoted as vR_ew.*

Note that the topological order of the vertices can be derived according to the precedence and equivalence relations between the vertices.

LEMMA 1. *In a definite range graph $G^d = (V, E, \psi, \omega)$, any pair of vertices v and $w \in E(G)$ ($v \neq w$) satisfies one of the following three conditions: (1) v Rp w; (2) w Rp v; (3) v Re w and w Re v.*

DEFINITION 9. *Two topological orders $TP_1(V) = \{v_1, v_2, \cdots, v_n : |V| = n, v_i \neq v_{i+1}, 1 \leq i < n\}$ and $TP_2(V) = \{w_1, w_2, \cdots, w_n : |V| = n, w_i \neq w_{i+1}, 1 \leq i < n\}$ are said to be* **equivalent** *to each other iff the following conditions are satisfied: For any i ($1 \leq i \leq n$), if $v_i \neq w_i$, then $v_iR_ew_i$.*

When equivalent topological orders are counted as one, it can be proved that a definite range graph $G^d = (V, E, \psi, \omega)$ has a unique topological order $TP(V)$.

THEOREM 3. *Given a slicing direction, the number of cutting-slices needed is equal to the number of different topological orders of the rectangle edges along the same direction regardless of the topological order of the rectangle edges along the other direction.*

The above theorem states that it is necessary to convert only one of the range graphs (either the horizontal or the vertical) into definite range graphs. Suppose the HRG has been converted into a set of definite range graphs denoted as HD and the VRG has been converted into a stable range graph VD^s. The algorithm **Enum_Cutting_Slice** described below enumerates a cuttings-slice for each definite graph $G^d \in HD$ and VD^s.

1. Sort vertices of G^d into topological order $TP1$.
2. Consecutive vertices $(v, w) \in TP1$ identifies a slice s. The range of the edge $(v, w) \in E(G^d)$ is the range of s's width.
3. For each slice s, do:
   (a) Identify vertices in VD^s that correspond to vertical rectangle edges that are contained in s.
   (b) Sort these vertices to derive topological order $TP2$[3].
   (c) Consecutive vertices $(m, n) \in TP2$ identifies a fragment f. The range of the edge $(m, n) \in E(VD^s)$ is the range of f's length.
4. For each edge e in G^d or VD^s, translate the range of e into additional constraints on the slices or fragments.

[3]It can be proved that a unique topological order exists for these vertices.