A Hotspot Detection Method Based on Approximate String Search

Shuma Tamagawa*, Ryo Fujimoto†, Masato Inagi‡, Shinobu Nagayama*, Shin'ichi Wakabayashi*
*‡Graduate School of Information Sciences, Hiroshima City University
†Faculty of Information Sciences, Hiroshima City University
3-4-1 Ozuka-higashi, Asaminami-ku, Hiroshima, 731-3194 Japan
Email: ‡inagi@hiroshima-cu.ac.jp

***Abstract*—In this paper, we propose an approximate string search-based method for detecting hotspots on mask patterns used in very-large-scale integration (VLSI). In mask patterns for manufacturing VLSI chips, there are some local patterns which induce open/short circuits and thus failures. Such patterns are called hotspots. They are detected by optical simulation before manufacturing. Since it, however, requires very long time, it is desirable to detect hotspot candidates in order to limit the target regions of optical simulation. For the detection, methods which check matching between each pattern from a pre-defined hotspot library and the mask pattern are attracting attention. At first, our proposed method transforms the mask pattern and the pre-defined hotspot patterns, which are two-dimensional, into one-dimensional strings. Then, it finds hotspot candidates by using approximate string search to detect patterns similar to hotspot patterns. The transformation is performed in a particular way to efficiently realize quasi-two-dimensional search by using string search. In addition, we focus on the distance between wires as a metric to find hotspot candidates, and give a priority to patterns which have wires with a shorter distance. To evaluate the effectiveness of our method, we conducted some experiments.**

Keywords–lithography; hotspot; optical simulation; approximate string matching.

I. INTRODUCTION

In recent decades, the density of semiconductor chips has greatly been increased with advances in very-large-scale integration (VLSI) technology. In lithography process for manufacturing VLSI chips [1], the circuit pattern (mask pattern) drawn on a photomask is transferred to a silicon wafer using photolithography machine. While 193nm wave length laser is currently used for the lithography process, the minimum pitch between wires is becoming smaller and has reached 14nm. Thus, because of diffraction during exposure, transfer of the mask pattern drawn on the photomask to the wafer sometimes fails. Hotspot is a place on the mask pattern where such a failure of transfer is likely to occur.

For manufacturing VLSI chips, a photomask is necessary. However, the manufacturing cost of the photomask is very high. To avoid remanufacturing photomasks, it is desirable to remove hotspots before manufacturing the photomask. Thus, optical simulation is conducted to detect hotspots. However, it takes very long time to conduct the optical simulation to the entire mask pattern. It is possible to shorten the simulation time by applying the optical simulation only to the hotspot candidates on the mask pattern. Thus, some methods for detecting hotspot candidates have been studied [2]-[6].

The method proposed in [2] is based on pattern matching [7]. The others [3]-[6] build a decision model, such as artificial neuron network, by learning hotspot/non-hotspot patterns, and detect hotspot candidates based on the model. [8] summarized related work including most of the methods.

In this study, we consider a method which conducts two-dimensional pattern matching by using string matching. In string matching, there are a lot of variation problems. String search problems are its variations to find the substrings same to a given pattern in a given string. An algorithm proposed by Knuth *et al.* [7] is well-known as an algorithm for the basic problem of string search. Hardware algorithms for the problems also have been proposed [9]-[11]. Our proposed method is based on approximate string search [12][13], which is a variation of string search and in which substrings similar to a given pattern are searched in a given string. Our method conducts approximate string search by transforming the two-dimensional mask pattern and the two-dimensional hotspot patterns into one-dimensional strings. When transforming these patterns, it is necessary to pad a lot of don't care characters to the hotspot patterns in order to match the widths of the mask and hotspot patterns. For efficient string search, we introduce a quantitative don't care character which represents a number of consecutive don't care characters. In addition, we confirmed that a pattern (in the mask pattern) similar to a hotspot pattern is more likely to be a hotspot if there are two wires in the pattern whose distance is smaller than that between the corresponding wires in the hotspot pattern. Thus, we propose a hotspot detection considering the distance between wires. Finally, we confirm the effectiveness of our proposed method comparing with a method based on two-dimensional simple matching.

The rest of this paper is organized as follows. First, in Section II, lithography, hotspot detection problem and related work, approximate string search problem, and an algorithm for the string matching problem are explained. Section III presents our hotspot detection method based on approximate string search. In Section VI, experimental results are shown. Finally, conclusions are described in Section V.

II. PRELIMINARIES

A. Lithography

Lithography (photolithography) is one of the processes for VLSI manufacturing, and a technology to transfer a circuit pattern drawn on a photomask, which is the master to replicate the circuit pattern, to a silicon wafer using a photolithography machine (exposure device).

The basis of lithography is shown in Figure 1. In lithography process, light is shed on the photomask and the mask

 ISBN: 978-1-61208-496-1

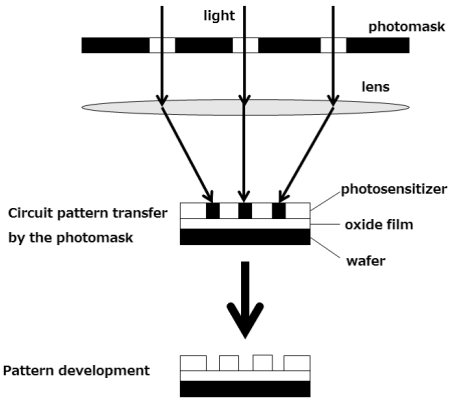

Figure 1. Lithography

(1) (2)

Figure 2. Hotspot: (1) a hotspot pattern, (2) transferred image

pattern drawn on the photomask is transferred to the wafer via lenses. While 193nm wave length laser is currently used for the lithography process, the minimum pitch between wires is becoming smaller and has reached 14nm. Thus, diffraction occurs during exposure. For this reason, there are cases in which the mask pattern drawn on the photomask cannot be correctly transferred to the wafer. Hotspot is a place on the mask pattern where such a failure of transfer is likely to occur. Let us consider a pattern (1) in Figure 2 as an example. Figure 2 (2) is a transferred image of the pattern (1). In the image (2), wires are fused at an unintended position. Thus, the pattern (1) is a pattern which induces a failure and considered as a hotspot.

B. Hotspot Detection Problem

There exist two problems relating to hotspots: hotspot decision and hotspot detection (hotspot search). Hotspot decision problem is to determine whether a given small pattern is a hotspot (candidate) or not, while hotspot detection problem is to detect hotspots (hotspot candidates) from a given large mask pattern. In hotspot detection problems, a mask pattern and a known hotspot pattern (or a set of known hotspot patterns) are given, and patterns similar to the known hotspot pattern (or one of the known hotspot patterns) are searched in the mask pattern.

C. Related Work

Most existing methods [2]-[6] solve a hotspot detection problem by scanning the mask pattern and solving hotspot decision problems.

The existing methods can be classified into [2], [3][4], and [5][6]. The main difference among the three groups is the information used for detection. In [2], a binary image of a pattern is encoded to a vector and used for detection. In [3][4], information such as the length and width of each wire is extracted from the pattern, and encoded to a vector. In [5][6], a pattern is encoded as a density-based data, and used for detection.

From another viewpoint, the existing methods can briefly be explained by their base algorithms: [2] is based on pattern matching, [3] is based on machine learning, [4] is a hybrid method combining machine learning and pattern matching, and [5][6] are based on fuzzy matching.

Here, we explain the overview of each group of the existing methods. In this paragraph, we describe the flow of a pattern matching-based method [2]. First, overlay a coarse grid on the mask pattern. Then, grid points on wire segments are colored. The coloring of the grid is transformed into a matrix, and then it is again transformed into a vector. Each hotspot pattern is transformed into vector in the same way. Then, the mask pattern vector and a hotspot pattern vector are matched by using the Knuth-Morris-Pratt (KMP) string matching algorithm [2][7]. If a hotspot pattern vector matches a part of the mask pattern vector, range pattern matching (RPM) is performed to them using a finer grid. In RPM, users can specify the ranges of the length and width of a segment of a wire in the hotspot pattern, etc., and thus flexible matching can be realized. This method realizes high accuracy by the hierarchical matching, in addition to low memory usage by the coarse global grid.

Second, we describe the flow of a machine learning-based method [3]. First, extract feature quantities for each known hotspot pattern considering the length, width, etc. of wire segments in the pattern, and construct a compact vector from the feature quantities of each pattern. Then, hierarchically build both artificial neural network (ANN) models and a support vector machine (SVM) models training by the vectors for accurate and robust decision-making. Then, a target pattern is applied hierarchical machine learning-based matching using the hierarchical model. This method realizes high accuracy, high robustness and low false-alarm ratio by the hierarchical model, in addition to short runtime by their compact feature vectors.

Next, we describe the flow of a hybrid method [4]. First, perform pattern matching for a target pattern and the hotspot patterns. If it matches one of the hotspot patterns, they conclude that it is a hotspot. Otherwise, matching based on machine learning is performed to the pattern. A pattern which is determined as a non-hotspot pattern in this stage is concluded to be a non-hotspot. A pattern which is determined as a hotspot pattern is applied machine learning-based matching again, using a different machine learning model to accurately examine if it is likely to be a hotspot or not. Only if it is determined as a hotspot by the second machine learning-based matching, they conclude that it is a hotspot. This method realizes high accuracy and short runtime by its hybrid strategy.

Then, we explain the common flow of fuzzy matching-based methods [5][6]. First, divide the mask and pre-defined hotspot patterns into tiles and calculate the density of wires in each tile as a feature quantity of the tile. Next, build a fuzzy matching model from the vectors of the feature quantities of the hotspot patterns. To build the model, calculate a distance, called city block distance (CBD), between every two hotspot patterns. If the CBD between two hotspot patterns is smaller

 ISBN: 978-1-61208-496-1

than a threshold, they are placed in the same hotspot group. Next, a fuzzy region is extracted from each hotspot group in the fuzzy space. Then, fuzzy regions are expanded as far as they are legal as fuzzy regions. This is necessary to divide the fuzzy space into hotspot regions and non-hotspot regions in a fuzzy manner. Finally, determine if a given pattern is a hotspot or not, using the fuzzy matching model. If an inequality is satisfied on the model with a given pattern, it is determined as a hotspot. When building and using the model, a heavy weight is imposed to the center area of each pattern, because failures occur in the center area of hotspot patterns with a higher possibility than outside the center area. For hotspot detection in [5][6], candidate patterns, which include suspicious polygons (wires), are extracted and applied fuzzy matching. This method realizes high accuracy and low false-alarm ratio when using fine tiles.

Our proposed method is based on string matching. In the method, every pixel image of the mask and known hotspot patterns is sliced and converted into a string, and then approximate string search is performed. We expect that approximate string search realizes more flexible hotspot search since it can find not only the substrings exactly same as the given pattern but also similar substrings. Before approximate string search, each known hotspot pattern is analyzed, and congested areas, where wires are close to each other, are given heavy weights to find severe patterns in priority. We consider a hierarchical approach, which most existing methods employ, is promising. Therefore, this study is the first step of an attempt to develop a better method to replace a stage of hierarchical methods.

D. Approximate String Matching Problem

Approximate string matching problem [12][13] is one of the string matching problems, and is a problem to determine if two given strings are similar or not. In this study, the similarity between strings is measured by the edit distance explained in the next subsection. If the edit distance is less than or equal to a given threshold, we consider they are similar each other.

E. Edit Distance

Let us consider a pair of characters $(a, b)(\neq (\epsilon, \epsilon))$, where ϵ is an *empty character*, which represents nonexistence of any character. The operation transforming character a in a string into b is called an *edit operation*, and is denoted by $a \rightarrow b$. For example, let us consider a string $A = gzh$. If an edit operation $g \rightarrow f$ is applied to the first character of A, we get $A^{'} = fzh$ as the resultant string of the operation. If an edit operation $z \rightarrow \epsilon$ is applied to the second character of A, we get $A^{'} = gh$. If an edit operation $\epsilon \rightarrow j$ is applied to the empty character between the second and third characters of A, we get $A^{'} = gzjh$. Hereinafter, we call an operation $a \rightarrow b$ a *substitution* if $a \neq \epsilon$ and $b \neq \epsilon$. Likewise, we call an operation $a \rightarrow \epsilon$ a *deletion*, and call an operation $\epsilon \rightarrow b$ an *insertion*. Any string can be transformed into an arbitrary string by applying the edit operations. An edit operation has its cost denoted by $\gamma(a \rightarrow b)$. We assume the costs of edit operations satisfy the equation below.

$$\gamma(a \rightarrow a) = 0$$

$$\gamma(a \rightarrow b) + \gamma(b \rightarrow c) \geq \gamma(a \rightarrow c)$$

Suppose strings A and B on alphabets Σ are given. A sequence of edit operations to transform A into B is denoted as $S = s_1, s_2, \ldots, s_m$. The cost of S is defined as

$$\gamma(S) = \sum_{i=1}^{m} \gamma(s_i).$$

The minimum value among the costs of all the sequences each of which transforms A into B is defined as *the edit distance between A and B* [12].

F. Approximate String Search Problem

Approximate string search is to find substrings similar to a given pattern in a long input sequence. More precisely, approximate string search is to find all the substrings whose edit distance to the pattern P are the minimum among all the substrings (or less then the given threshold k), in the input sequence S.

We here explain a dynamic programming-based algorithm for approximate string search [12][13]. Prepare an $(n + 1) \times (m + 1)$ two-dimensional array D, where n is the length of the pattern $P = a_1 a_2 \cdots a_n$, and m is the length of the input sequence $S = b_1 b_2 \cdots b_m$. The element $D(i, j)$ of D is defined by the equation below. Then, $D(n, j)(1 \leq j \leq m)$ give the edit distances of substrings from the pattern. If the value is the minimum among all the $D(n, j)(1 \leq j \leq m)$ (or less than the user-defined threshold k), substrings whose terminal is b_j are considered as those similar to the pattern. The initials of the substrings can be found by backtracing on D. The details of the identification of similar substrings of our proposed method are explained later.

$$D(0, 0) = 0, \quad D(0, j) = 0,$$

$$D(i, 0) = D(i - 1, 0) + del(a_i),$$

$$\begin{aligned} D(i, j) = \min\{&D(i - 1, j) + del(a_i), \\ &D(i, j - 1) + ins(b_j), \\ &D(i - 1, j - 1) + sub(a_i, b_j)\} \end{aligned}$$

$$sub(a_i, b_j) = \gamma(a_i \rightarrow b_j),$$

where the function ins, del, and sub denote the insertion, deletion, and substitution costs.

Let us consider an example of searching substrings of $S = tttytzmy$ similar to $P = tyzm$. Here, we set the every cost of the insertion, deletion and substitution operations to 1. Then, we obtain an array D shown in Figure 3 from the recursive definition of $D(i, j)$. Next, we search the minimum element of the bottom row of D. In this example, $D(4, 7) = 1$ has the minimum value. Next, we trace the edit operations back from the element. If a deletion operation was applied there, *i.e.,* the minimum function in the recursive definition chose a deletion operation, we move to the upper element. Likewise, if an insertion (substitution) operation was applied there, we move to the left (upper-left) element. Finally, we obtain $S < 3, 7 >= tytzm$ as a substring similar to the pattern P, where $S < i, k >= b_i b_2 \cdots b_k$. As shown in the example, in approximate string search, substrings similar to the pattern are found by calculating the edit distances from the middle of the input sequence. The time-complexity of this algorithm is $O(nm)$.

 ISBN: 978-1-61208-496-1

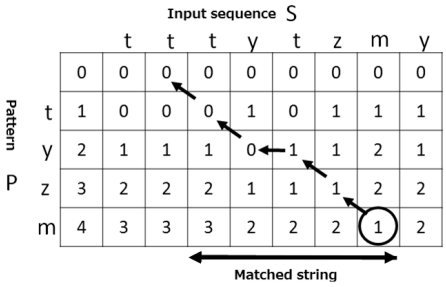

Figure 3. Edit distance array D

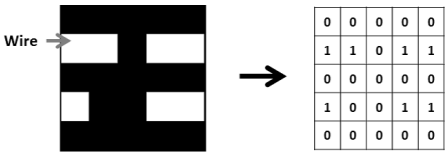

Figure 4. Image data and its corresponding array

III. Hotspot Detection Based on Approximate String Search

In this section, we present our hotspot detection method based on approximate string search. In this method, the mask pattern and a hotspot pattern, which are both two-dimensional data, are transformed into one-dimensional strings to apply approximate string search calculating array D by dynamic programming. In addition, we propose an extension of the method to give priority to be picked up to patterns more likely to be hotspots.

A. Transformation into One-dimensional Data

Mask patterns and hotspot patterns are image data. We transform them into two-dimensional array of characters, in which wire area is represented by 1 and empty area is represented by 0.

An example is shown in Figure 4. In the left image in it, white areas show wires, and black areas show empty areas.

We transforms the two-dimensional arrays into one-dimensional data. In the transformation of the mask pattern, the two-dimensional array of the mask pattern is divided into rows. Then, the tail of the first row and the head of the second row is connected. And, the tail of the second row is connected to the head of the third row. Connecting all the rows according to the procedure, the two-dimensional mask pattern data is transformed into one-dimensional data. Next, we explain how to transform the two-dimensional hotspot pattern into one-dimensional data. The procedure of the transformation is shown in Figure 5. First, the array of the hotspot pattern is divided into rows, like the transformation of the mask pattern. Next, don't care characters are inserted to each row to fit the width of the hotspot pattern to that of the mask pattern. Then, the rows are connected into one-dimensional data, like the procedure of the mask pattern.

The insertion of don't care characters virtually realizes two-dimensional matching by one-dimensional matching. Figure 6 illustrates hotspot search with don't care characters. A don't care character can be represented by setting its substitution

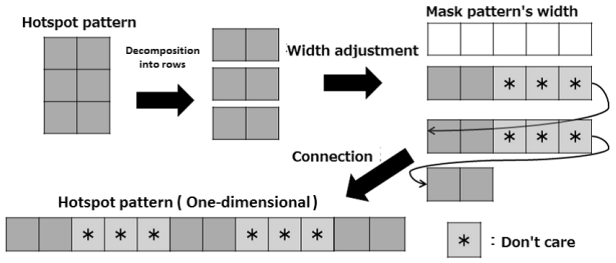

Figure 5. Transformation of hotspot data

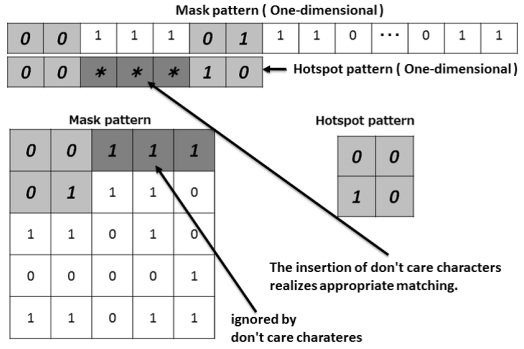

Figure 6. Matching after inserting don't care characters

cost to 0. To make the explanation simple, let us consider the case of matching the two-dimensional data (*i.e.*, not searching but matching). By padding don't care characters to the one-dimensional data of the hotspot pattern in order to fit the width of the hotspot pattern to that of the mask pattern, the values of the elements of the hotspot pattern and the values of the elements in the corresponding area of the two-dimensional mask pattern are matched (*i.e.*, compared). In the figure, the values from the third element to the fifth element in the first row of the mask pattern are matched with the inserted don't care characters in the hotspot pattern. (That is, the values from the third element to the fifth element of the mask pattern are ignored.) Therefore, the values from the first element to the second element in the second row of the two-dimensional mask pattern are appropriately matched with the values from the first element to the second element in the second row of the two-dimensional hotspot pattern by one-dimensional matching. This way, the hotspot pattern is matched with the corresponding area of the two-dimensional mask pattern.

B. Dynamic Programming

In our method, since hotspot candidates are searched by using approximate string search, array D is calculated by using the dynamic programming shown in the previous section. Except the first row and column, the value of each element of the array D is calculated by using the value of its upper, left and upper-left elements. These calculations are done line by line from the top to the bottom.

Let us focus on the calculation of the element where the character of the hotspot pattern is don't care. In the case, consecutive x don't care characters require x rows of the calculations. Figure 7 illustrates the calculation of array

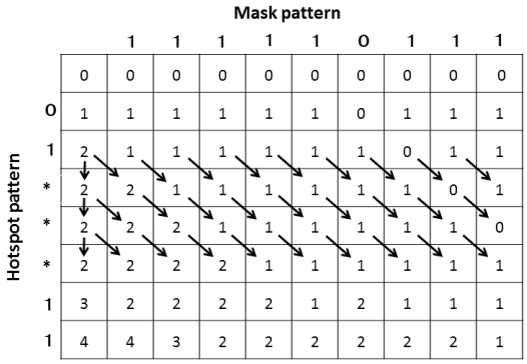

Figure 7. Calculation of array D with consecutive don't care characters

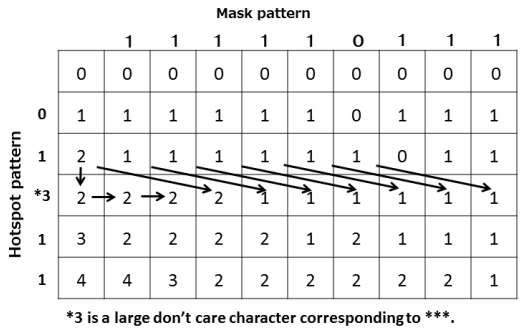

Figure 8. Calculation of array D with a large don't care character

D with consecutive don't care characters. In our method, such consecutive x don't care characters are merged to one character, called a *large don't care*. If the character of the hotspot pattern is a don't care character, each element of the corresponding row is equal to its upper-left element. (Each element of the leftmost column is equal to its upper element.) On the other hand, if the character of the hotspot pattern is with a large don't care character, each element of the corresponding row is equal to the element at the x-th left column at the upper row. In this way, x lines of calculations of don't care characters are realized by a line of calculations of a large don't care character. Figure 8 illustrates the calculation of array D with a large don't care character.

C. Detection of Hotspot Candidates

After calculating array D, substrings similar to the hotspot pattern are detected as hotspot candidates. To detect hotspot candidates, first, we focus on the elements with the minimum value in the bottom row of D. Each of these elements is considered as the terminal character of a hotspot candidate. Since we assume the hotspot candidate has the length same as the hotspot pattern, the initial character can be identified from the terminal character. The assumption is based on the fact that a hotspot pattern and candidates similar to the pattern are originally two-dimensional images, and have the same size or almost same sizes. Next, we focus on the elements with the minimal values in the bottom row of D. The elements whose values are less than or equal to a use-defined value (threshold) k are chosen from the elements with the minimal values, and processed in the same way.

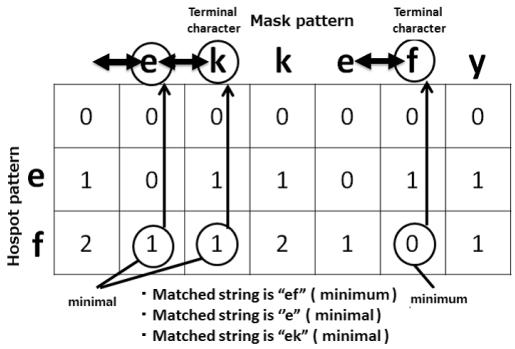

Figure 9. Detection of hotspot candidate

Figure 9 illustrates an example of hotspot candidate detection. First, the minimum values in the bottom row of array D are found. In Figure 9, the value 0 of the 6th column is the minimum. The column with the minimum value corresponds to the character f of the mask pattern. Thus, the substring ef which has the terminal character f and the length same as the pattern ef is detected as a hotspot candidate. Next, the minimal values no more than the threshold k in the bottom row of array D are found. In this example, we set $k = 1$. The second and third columns in the bottom row satisfy the condition. Thus, the substring $e(= \epsilon e)$ whose terminal is in the second column is found as a hotspot candidate. Likewise, the substring ek whose terminal is in the 3rd column is found as a hotspot candidate.

It seems that array D requires a large amount of memory area. However, since we need only the bottom raw of D to detect the hotspot candidates, the memory area for each raw (except the bottom raw) can be released after the calculation of the next raw. That is, we need to memorize only the current and previous rows at a time.

D. Detection of Severe Patterns Considering Distance between Wires

Hotspot patterns are those which induce short or open circuits. If there exist patterns in the mask pattern similar to a hotspot pattern, they are likely to be hotspots. We focus on the distance between wires as a measure to determine the criticality of hotspot candidates. We performed simulations for two patterns by using an optical simulator [14]. The results of the simulations for the patterns (1) and (2) in Figure 10 are shown in Figure 11 (1) and (2), respectively. In Figure 10, pattern (1) and (2) are variations of the original hotspot pattern. In pattern (1) ((2)), the horizontal distance between the left and right wires is shorter (longer) than the original one (the horizontal segment of the right wire is longer (shorter) then that of the original one). As shown in Figure 11, pattern (1) clearly caused a short circuit, and thus it is a hotspot. Although pattern (2) might be a hotspot, too, it is not clear from the simulation. These results indicate that among patterns which have the same similarity to the original hotspot pattern, patterns with shorter distances between wires are more critical. Therefore, we propose an extension of our hotspot detection method to detect more critical patterns in priority among patterns with the same similarity, considering the distances between wires.

Let us consider two hotspot candidates similar to a hotspot

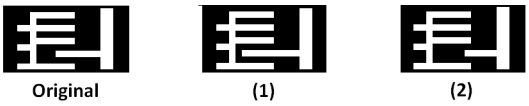

Figure 10. Patterns

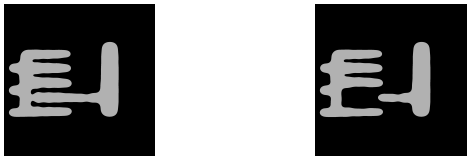

Figure 11. Transferred images

pattern. There exists the (horizontally) shortest distance between wires in the hotspot pattern. One of the hotspot candidates (shown in Figure 12(1)) has the shorter distance between wires at the corresponding area than the hotspot pattern, and the other (shown in Figure 12(2)) has the longer distance between wires at the corresponding area with the same difference. As mentioned above, the candidate with shorter distance is more likely to be a hotspot. However, these candidates have the same similarity (*i.e.*, edit distance) to the hotspot pattern. In our extension, more critical hotspot candidates are found by considering the distance between wires and increasing the costs of the edit operations around the corresponding area (between wire segments with the minimum distance). Although there are vertical and horizontal distances, it is difficult to consider both the distances at the same time because our method is base on one-dimensional matching. Thus, we here discuss only horizontal distance between wires. Vertical distance can be handled in the same way.

The procedure is as follows. First, find the place where the distance between wires is the minimum in the hotspot pattern, by scanning the hotspot pattern. Then, calculate the threshold by multiplying the minimum distance and a user-defined coefficient. The threshold decides if the distance is short or not. Next, find the places where the distance between wires is less than or equal to the threshold in the hotspot pattern, by scanning the hotspot pattern again. We assume the places are likely to cause short circuits. Next, decrease the substitution costs of the places (between wires). If the places of the hotspot pattern are applied substitution operations, the non-wire pixels (= 0s) are substituted by wire pixels (= 1s). Thus, patterns more likely to be hotspots with shorter distance between wires can be found. The procedure is depicted in Figure 13.

IV. Experimental Evaluation

We performed experiments to evaluate the effectiveness of our method. We evaluated the runtime of a two-dimensional template matching-based method and our one, and evaluated the hotspot candidates detected by ours. The two-dimensional template matching-based method is a method to detect hotspot candidates by matching the hotspot pattern and the mask pattern moving the hotspot pattern from the top-left corner to the bottom-left corner on the mask pattern pixel by pixel.

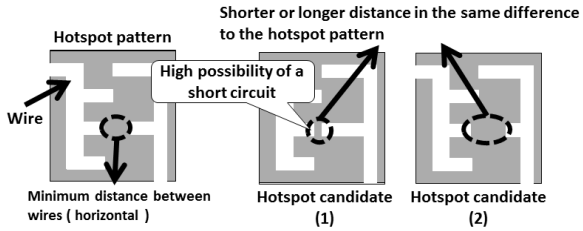

Figure 12. Hotspot pattern and its candidates

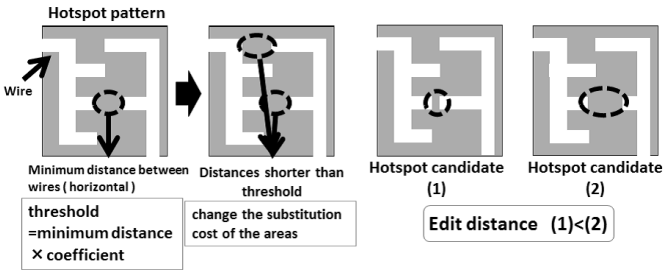

Figure 13. Detection of severe patterns

TABLE I. RUNTIME

	Runtime (sec)
2D matching	152.02
Non-LD method	906.54
Proposed method	512.41

In addition, we also evaluated the runtime of non-LD method which is based on our method but without large don't care characters, to evaluate the effectiveness of large don't care characters. The experiments are conducted on a Linux PC (CentOS release 6.3) equipped with Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz CPU.

A. Comparison of Runtime

To evaluate runtime, we performed detection of hotspot candidates with a 1020x1020-pixel mask pattern and a 250x250-pixel hotspot pattern.

The runtime of each method is shown in Table I. The runtime of our method was longer than that of two-dimensional matching. However, we confirmed that large don't care characters improved the runtime.

Two-dimensional template matching starts its matching at the first column at the first row of the mask pattern, and moves to the next column until the right edge of the hotspot pattern reaches the right edge of the mask pattern. When the right edge of the hotspot pattern reaches the right edge of the mask pattern, it moves to the first column at the next row. This is repeated until the bottom-left corner of the hotspot pattern reaches that of the mask pattern. Thus, the time-complexity of two-dimensional matching is ((the width of the mask pattern) - (the width of the hotspot pattern)) $\times$ ((the height of the mask pattern) - (the height of the hotspot pattern)) $\times$ (the number of characters of the hotspot pattern)). On the other hand, the time-complexity of our proposed method is (the number of characters of the hotspot pattern) $\times$ (the number of characters

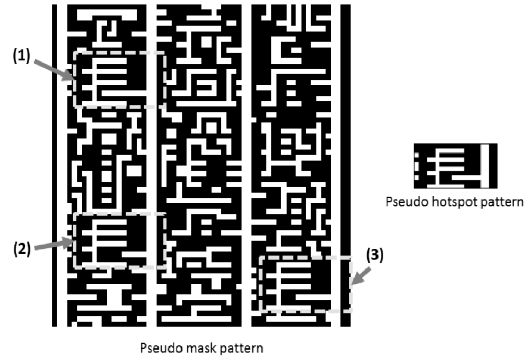

Figure 14. Pseudo patterns

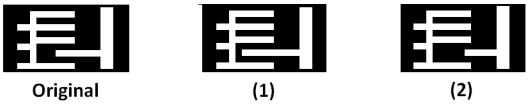

Figure 15. Detected candidates

of the mask pattern). In addition, in the dynamic programming in our proposed method, three elements are referenced in the calculation of each element, and thus it takes time to calculate the value of an element compared to the case of template matching. They are the reasons why the runtime of our proposed method was longer than that of two-dimensional matching.

B. Comparison of Detected patterns

Next, we confirmed that if the detection of severe hotspot candidates considering the distances between wires is realized by our extension or not, using a pseudo mask pattern (60x60) and a pseudo hotspot pattern (10x20).

The standard cost of edit operations was set to 10, and the cost of each substitution operation at the places with distance between wires which is no more than the threshold was set to 5. The mask and hotspot patterns are shown in Figure 14, and the detected hotspot candidates are shown in Figure 15. The pattern (1) shown in Figure 15 is the same as the hotspot pattern, and thus their edit distance was 0. The edit distance of the pattern (2) is less than that of the pattern (3). As a result, the pattern (2) was detected in priority.

V. CONCLUSIONS

In this paper, we proposed a hotspot detection method based on approximate string search, and an extension which detects severe hotspot candidates in priority. Our future work includes more in-depth experiments using open benchmark data, comparison with other existing methods, analysis of patterns more likely to be hotspots, and extensions of our method based on the analysis.

REFERENCES

[1] Tatsuhiko Higashiki and Yasunobu Onishi, “Trends in semiconductor lithography technologies and Toshiba’s approach,” TOSHIBA review, Vol.67, No.4, 2012, pp.2–6.

[2] H. Yao, S. Sinha, J. Xu, C. Chiang, and X. Hong, “Efficient range pattern matching algorithm for process-hotspot detection,” in Proc. IET Circuits Devices Syst., 2008, pp. 2–15.

[3] D. Ding, A. J. Torres, F. G. Pikus, and D. Z. Pan, “High performance lithographic hotspot detection using hierarchically refined machine learning,” in Proc. 16th Asia South-Pacific Design Autom. Conf. (ASP-DAC), Yokohama, Japan, 2011, pp. 775–780.

[4] Jen-Yi Wuu, Fedor G. Pikus, and M. M-Sadowska, “Efficient approach to early detection of lithographic hotspots using machine learning systems and pattern matching,” in Proc. SPIE 7974, Design for Manufacturability through Design-Process Integration V, 79740U, April 04, 2011.

[5] S.-Y. Lin, J.-Y. Chen, J.-C. Li, W.-Y. Wen, and S.-C. Chang, “A novel fuzzy matching model for lithography hotspot detection,” in Proc. Design Autom. Conf. (DAC), Austin, TX, USA, 2013, pp. 1–6.

[6] W. Wen, J. Li, S. Lin, J. Chen, and S. Chang, “A fuzzy-matching model with grid reduction for lithography hotspot detection,” IEEE Trans. on CAD, Vol. 33, No. 11, Nov. 2014, pp.1671–1679.

[7] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, “Fast pattern matching in strings,” SIAM J. Comput., vol.6, no.2, 1977, pp.323–350.

[8] Jhih-Rong Gao, Bei Yu, Duo Ding, and David Z. Pan, “Lithography Hotspot Detection and Mitigation in Nanometer VLSI,” in Proc. IEEE International Conference on ASIC (ASICON), 2013, pp.1–4.

[9] J. T. L. Ho and G. G. F. Lemieux, “PERG: A scalable FPGA-based pattern-matching engine with consolidated bloomier filters,” in Proc. 2008 IEEE International Conference on Field Programmable Technology, Dec. 2008, pp.73–80.

[10] Y. Sugawara, M. Inaba, and K. Hiraki, “Over 10Gbps string matching mechanism for multi-stream packet scanning system,” in Proc. International Conference on Field Programmable Logic and Applications, Aug. 2004, pp.484–493.

[11] B. C. Brogie, R. K. Cytron, and D. E. Taylor, “A scalable architecture for high throughput regularexpression pattern matching,” in Proc. 33rd International Symposium on Computer Architecture, 2006, pp.191–202.

[12] Yuichiro Utan, Shin’ichi Wakabayashi, and Shinobu Nagayama, “An FPGA-based text search engine for approximate regular expression matching,” in Proc. International Conference on Field-Programmable Technology, Dec. 2010, pp.184–191.

[13] Yuichiro Utan, Shin’ichi Wakabayashi, and Shinobu Nagayama, “A systolic algorithm for approximate regular expression matching and its FPGA implementation”, IEICE Journal D, Vol. J94-D, No.6, June 2011, pp.935–944. (in Japanese)

[14] Zhuo Li, et al., “ICCAD 2013 Contest,” http://cad-contest.cs.nctu.edu.tw/CAD-contest-at-ICCAD2013/problem_c/ [retrieved: June 2016].