Table Reference-Based Acceleration of a Lithography Hotspot Detection Method Based on Approximate String Search

Shuma Tamagawa, Masato Inagi, Shinobu Nagayama, Shin'ichi Wakabayashi
email: shuma@lcs.info.hiroshima-cu.ac.jp, {inagi, s_naga, wakaba}@hiroshima-cu.ac.jp
Graduate School of Information Sciences, Hiroshima City University
3-4-1 Ozuka-higashi, Asaminami-ku, Hiroshima, 731-3194 Japan

***Abstract*—In nanoscale large-scale integration (LSI) manufacturing, there exist hotspots on mask patterns, which cause failures of pattern transfer. Such hotspots are detected by optical simulation to remove them. However, it requires a long time. Thus, development of efficient hotspot detection methods is required. As one of the methods, an existing one based on approximate string search has been proposed. Although this method is expected to find hotspots more flexibly than commonly-used template matching, computation of edit distance matrices used for approximate string search still requires a long time. Thus, in this study, we accelerate the computation by using table-reference of precomputed values and simultaneous computation of multiple elements. Our experiments showed that our improved method achieved about 1/11 computation time compared to the original one.**

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

I. INTRODUCTION

In nanoscale large-scale integration (LSI) manufacturing, lithography process is one of the most important processes, in which mask patterns printed on photomasks are transferred to the wafer using exposure equipment. In the process, some patterns tend to be failed to be transferred because of optical diffraction. Such patterns are called *hotspots* [1].

Since the cost of manufacturing photomasks is quite high, it is better to remove hotspots from the mask patterns in advance. Thus, lithography engineers apply optical simulation to the mask patterns received from mask designers. If hotspots are found by the optical simulation, it is informed to the designers, and the patterns are revised. This is repeated until all the hotspots are removed. However, optical simulation is time-consuming. To reduce the number of times of optical simulation, mask designers need to detect and remove hotspots in advance. Therefore, some studies on efficient hotspot detection have been conducted [2]–[6].

[2] proposed a template-matching-based method, which directly matches mask patterns and hotspot patterns. A mask pattern and a hotspot pattern are shown in Figure 1, as examples. A hotspot pattern is a pattern which should be detected from a mask pattern. While this method has a high ability to detect known patterns, its ability to detect unknown patterns is low. [3] discussed some hotspot detection methods using some machine-learning methods, such as artificial neural network (ANN) and support vector machine (SVM). According to the nature of machine-learning-based methods, they can detect unknown patterns as hotspots. However, they cause a large number of false-positive detections. [4] proposed a hybrid method based on template matching and machine-learning. Though the hybrid approach improved the accuracy of detection, the number of false-positive detections is still large. In addition, it takes 10 to 100 times longer for detection. [5] adopted a fuzzy-matching model instead of ANN or SVM. They improved both of execution time of detection and the accuracy of detection. Although there exist aforementioned hotspot detection methods, some mask designers use template-matching to detect hotspot patterns since their execution time and accuracy do not meet the level required by the designers.

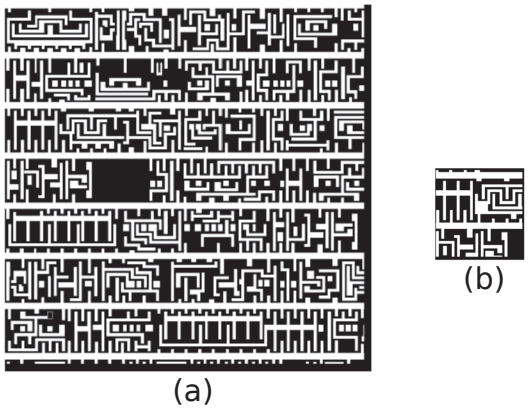

Figure 1. Circuit patterns: (a) mask pattern, (b) hotspot pattern

In [6], we proposed an approximate string matching-based hotspot detection method for flexible hotspot detection. Comparing to machine-learning-based methods, this method can detect hotspot candidates in a short time. Comparing to template-matching-based methods, this method can detect hotspot candidates more flexibly (*i.e.*, unknown patterns can be detected.) However, to calculate the value of each element in the edit distance matrix for approximate string matching, we need to refer to three elements in the matrix. Thus, comparing to template-matching-based methods, in which only one value is referred for the corresponding calculation, it takes a longer time for calculation (although it is just a constant coefficient factor).

Thus, in this paper, we improve the approximate string matching-based method [6] by using table-reference of precomputed values and simultaneous calculation of multiple elements in the edit distance matrix. Each region of simultaneously handled elements is a $k \times k$ partial matrix of the edit distance matrix, where k is the user-defined constant which decides parallelism. In the proposed method, the calculation of a region for every possible input set is performed in advance, and the result is memorized in a reference-table. Then, the

 ISBN: 978-1-61208-585-2

edit distance matrix is calculated by using the table. For efficient calculation, the values in a region is encoded to one word of memory to calculate the values of multiple elements with only one memory access. Experimental result showed the high effectiveness of the proposed method in execution time compared to the existing method [6].

The rest of this paper is organized as follows. First, in Section II, we explain about lithography, and we provide the definitions of the approximate string matching problem, the edit distance, the approximate string search problem, which is one of the variations of the approximate string matching problem, and on which our proposing method and [6] are based. Next, in Section III, the definition of the hotspot detection problem and the existing hotspot detection method [6] are shown. Then, in Section IV, we propose an improved method which uses table-reference. Section VI shows some experimental results, and finally conclusions are given in Section VII.

II. PRELIMINARIES

A. Lithography

Lithography is one of the processes of LSI manufacturing. In the process, a circuit pattern drawn on a photomask is transferred to the wafer using exposure equipment. A photomask is one of the masters to make a circuit on the wafer.

Figure 2 illustrates lithography process. In lithography process, a mask pattern drawn on a photomask is transferred onto the wafer via lenses, shedding light from above the photomask. While 193nm laser is commonly used in advanced lithography processes [7], the minimum pitch between wires is decreasing, and has reached 14nm. Therefore, some sub-patterns cannot be transferred correctly because of diffraction of light. Such a sub-pattern is called *a hotspot*. An example of a mask pattern is shown in Figure 3(a). The transferred image (by optical simulation) of Figure 3(a) is shown in Figure 3(b). In Figure 3(b), wires are connected at an unintended position, and some wires are too thin. Therefore, the pattern shown in Figure 3(a) is a hotspot pattern.

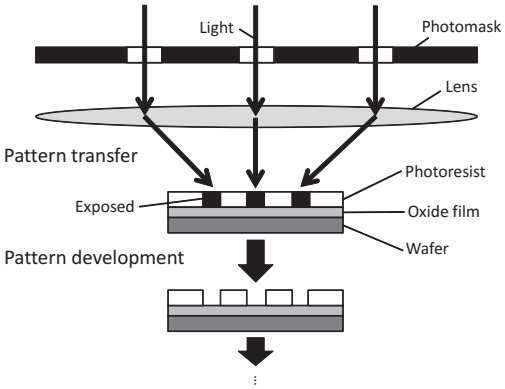

Figure 2. Principle of photolithography process

B. Approximate String Matching Problem

Approximate string matching problem [8] 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.

(a) (b)

Figure 3. Hotspot: (a) mask pattern, (b) its transferred image

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

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

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

Suppose strings A and B on alphabets Σ are given. A sequence of edit operations to transform A into B is denoted as $O = o_1, o_2, \ldots, o_m$. The cost of O is defined as

$$\gamma(O) = \sum_{i=1}^{m} \gamma(o_i).$$

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* [8].

D. 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 t), in the input sequence S.

We here explain a dynamic programming-based (DP-based) algorithm for approximate string search [8], [9]. Prepare an $(n + 1) \times (m + 1)$ two-dimensional array D, where D has $n+1$ rows and $m+1$ columns, n is the length of the pattern $P = a_1 a_2 \cdots a_n$, and m is the length of the input sequence $S = b_1 b_2 \cdots b_m$. An element $D(i, j)$ (at $(i+1)$-th row, $(j+1)$-th column) of D is defined by the following equations:

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

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

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

where the functions ins, del, and sub denote the insertion, deletion, and substitution costs. Figure 4 illustrates the DP-based calculation of an edit distance matrix, and Figure 5 shows the resultant edit distance matrix. D is called *edit distance matrix*, and $D(n,j)(1 \leq j \leq m)$ gives the edit distance between the pattern P and a substring (whose terminal character is b_j) in the input sequence S. If the value is the minimum among all the $D(n,j)(j = 1,2,\ldots,m)$ (or less than the user-defined threshold t), we consider that the substring is similar to the pattern. The initial character of such a substring is found by tracing back the DP-based calculation on D. Figure 6 illustrates how to identify similar substrings. The details of the identification of similar substrings in our proposed method are explained in Section III.

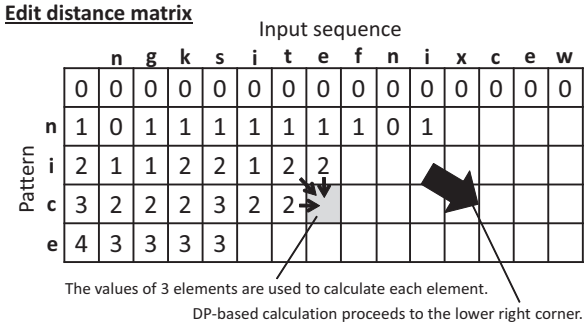

Figure 4. Calculation of edit distance matrix

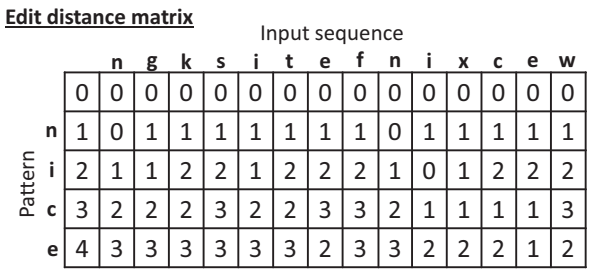

Figure 5. Edit distance matrix

III. Hotspot Detection Based on Approximate String Search

In this section, the existing hotspot detection method based on approximate string search [6], which we improve in this study, is explained. 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.

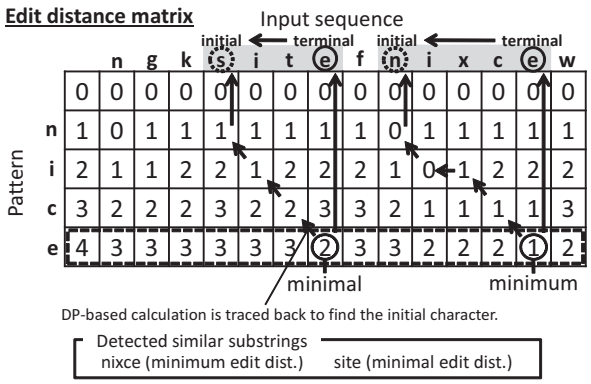

Figure 6. Identification of similar substrings

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 7. In the left image in it, the white areas represent wires (or other objects), and the black area represents an empty space.

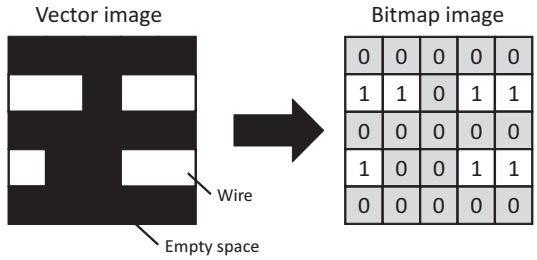

Figure 7. Image data and its corresponding array

We transform the two-dimensional arrays into one-dimensional data. First, 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 are connected. And, the tail of the second row is connected to the head of the third row. Likewise, all the rows are connected and the two-dimensional mask pattern is transformed into one-dimensional data (Figure 8).

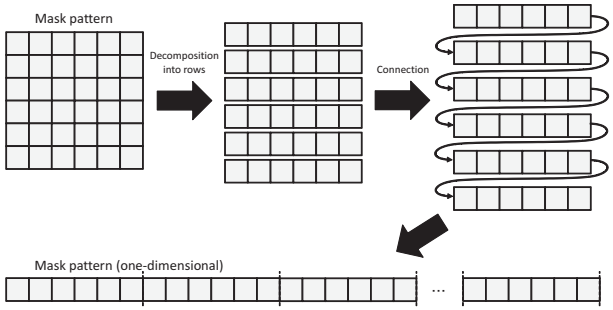

Figure 8. Transformation of mask pattern data

Next, the array of the hotspot pattern is divided into rows, like the transformation of the mask pattern. Then, for each

row, don't-care characters are added so that the number of characters of the row becomes equal to that of a row of the mask pattern (Figure 9). A don't-care character is the special character which matches any character. By adding don't-care characters, mismatch of the positions of the head characters of the rows of the hotspot pattern is corrected. Note that such consecutive don't-care characters can be substituted by a special character, called a large-don't-care, to efficiently calculate the edit distance matrix [6]. Hereinafter, the hotspot pattern is processed just like the mask pattern.

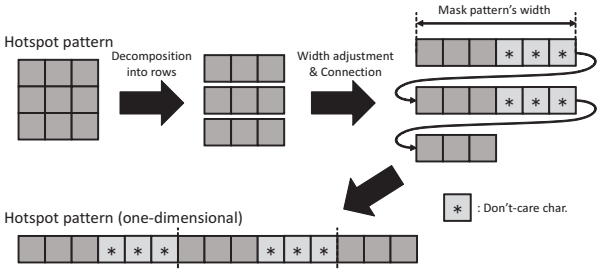

Figure 9. Transformation of hotspot pattern data

B. Dynamic Programming

In both of our proposed method and our previous one [6], 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.

C. Detection of Hotspot Candidates

After calculating array D, substrings similar to the hotspot pattern are detected as hotspot candidates. To detect hotspot candidates, we focus on the elements with the minimal values (less than a user-defined threshold) in the bottom row of D. Each of these elements is considered to correspond to 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 the same size. Figure 10 illustrates an example of hotspot candidate detection of our methods. (In the example, patterns are described in regular strings for simplicity.)

IV. Proposed method

In this section, we propose an improved hotspot detection method based on table-reference. Our proposed method is an extension of the existing method [6]. First, we explain the basis of table-reference-based edit distance calculation, some problems for implementation, and our solutions of the problems. Then, we explain our proposed method, by describing mask pattern encoding, hotspot pattern encoding, calculation of the encoded edit distance matrix, and detection of hotspot candidates.

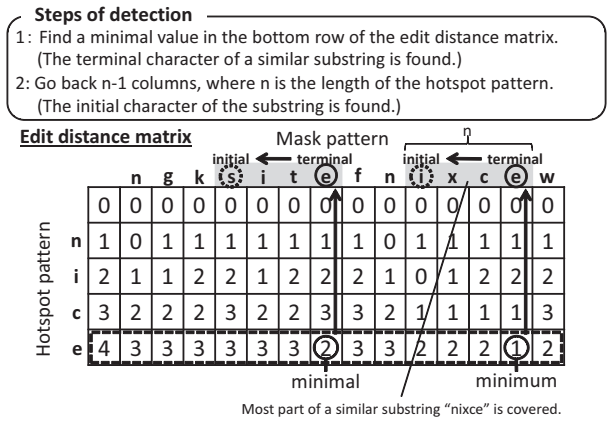

Figure 10. Detection of hotspot candidate

A. Table-reference

In our proposed method, calculation of the edit distance matrix is accelerated by using table-reference of precomputed values and simultaneous calculation of multiple elements in the matrix. Each region of simultaneously handled elements is a $k \times k$ partial matrix of the edit distance matrix, where k is the user-defined constant which decides parallelism. Figure 11 shows an example of 3×3 region. The number of inputs and outputs necessary for calculating each region is decided by k, as shown in Figure 12 (a) and (b). In addition, the values of the outputs are uniquely determined by the values of the inputs. Thus, a reference table for calculating a region can be developed. Hereinafter, a set of inputs for a table refers to an address (or index) of the table.

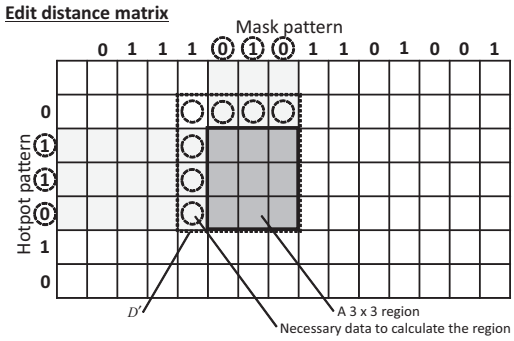

Figure 11. 3 x 3 region in edit distance matrix

A set of inputs for the table consists of the necessary values for DP-based calculation of the elements in a region. That is, it consists of the corresponding characters of the mask and hotspot patterns ($2k$ characters), the k elements to the left of the region, the k elements to the upper of the region, and the element to the upper-left of the region ($2k + 1$ elements in total). A set of outputs consists of the values of the elements in the region. Given a region, then we call the partial matrix of D which includes the region and the elements corresponding to the inputs of the region, D' (Figure 11). D' is equivalent to the region expanded one column and one row to the upper-left direction.

However, it is not practical to directly make a table of

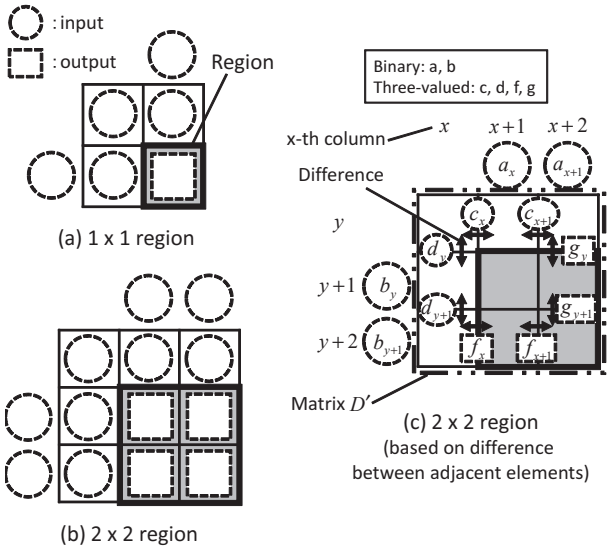

Figure 12. Input-output relation on the calculation of a region : (a) case of 1×1, (b) case of 2×2, (c) case of 2×2 using the difference between adjacent elements in the edit distance (with input-output names)

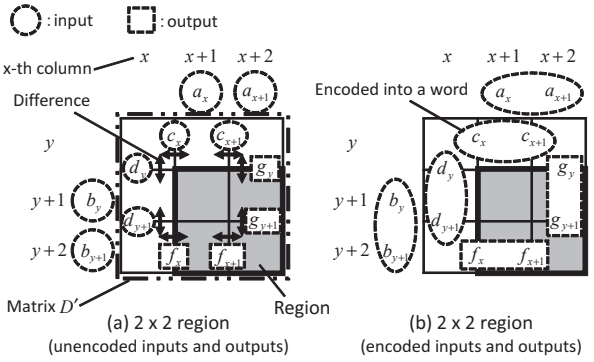

Figure 13. Encoding of inputs/outputs of region: (a) unencoded inputs/outputs of 2×2 region, (b) encoded inputs/outputs

region calculation, because it requires a huge memory space. This is because the range of the value of each element is large, and thus the number of combinations of the input values of the table explodes. Thus, we focus on the fact that, by deciding the reference element (*e.g.*, the upper-left one) of D', the value of an element can be represented by the difference between the element and the reference one, and the value can be calculated by using only the differences between elements. In addition, we found the fact that the difference between the adjacent two elements is -1, 0 or 1 (when an edit cost is defined as 0 or 1). (The proof is omitted due to the limitation of space.) By adopting difference calculation considering the facts, we mitigate the virtual range of the value of each element, and thus the required memory space. Furthermore, only the values of the elements in the bottom row and the right-most column in D' are necessary for calculating the edit distance matrix D (the values of the other elements are not necessary), because only the bottom row of D is necessary for hotspot detection. Thus, the computational complexity of each region can be reduced to $O(k)$ from $O(k^2)$ by table-reference. Examples of sets of inputs/outputs of a table (the necessary values for calculating regions) are shown in Figure 12(c).

Moreover, a set of inputs/outputs can be encoded into one word when k is small. Thus, in that case, the edit distance matrix can be calculated in the encoded values, so that the computational complexity of each region is reduced to $O(1)$ from $O(k)$. Figure 13 illustrates encoded inputs and outputs.

Let (x, y) be the coordinate of the upper-left element of D' in the edit distance matrix D. Then, the inputs are

- $a_x, a_{x+1}, ..., a_{x+(k-1)}$: mask pattern characters corresponding to the region (input1)
- $b_y, b_{y+1}, ..., b_{y+(k-1)}$: hotspot pattern characters corresponding to the region (input2)
- $c_x, c_{x+1}, ..., c_{x+(k-1)}$: the differences between adjacent two elements in the upper-most row of D' (input3)
- $d_y, d_{y+1}, ..., d_{y+(k-1)}$: the differences between adjacent two elements in the left-most column of D' (input4),

and the outputs are

- $f_x, f_{x+1}, ..., f_{x+(k-1)}$: the differences between adjacent two elements in the bottom row of D' (output1)
- $g_y, g_{y+1}, ..., g_{y+(k-1)}$: the differences between adjacent two elements in the right-most column of D' (output2).

The inputs/outputs are encoded by the encoding functions

$$P(p_x, p_{x+1}, ..., p_{x+i}) = \sum_{i=0}^{k-1} p_{x+i} \times 2^{(k-1)-i} \quad (1)$$

$$Q(q_x, q_{x+1}, ..., q_{x+i}) = \sum_{i=0}^{k-1} q_{x+i} \times 3^{(k-1)-i} \quad (2)$$

to develop a table.

Hereinafter, we call the four inputs, input1, input2, input3, and input4, and the two outputs, output1 and output2. The ones composed of binary variables are encoded by using (1), and the ones composed of three-valued $(-1, 0, 1)$ variables are encoded by using (2). Therefore, the outputs can be memorized in two words of memory if $\log_2 3^k$ is less than or equal to the bit width of a word. In that case, $k \times k$-element calculation can be done accessing the memory only two times, so that the computational complexity of each region is $O(1)$. Note that the number of possible combinations of the inputs is $2^{2k} \times 3^{2k}$ and is an exponential function of k. In our experimental environment, we developed tables up to $k = 5$ (0.23GB).

B. Hotspot detection using table-reference

In this subsection, we explain our table-reference-based hotspot detection method. As mentioned in the previous subsection, a table is developed by calculating the output values of a region for each possible combination of input values before starting hotspot detection. Then, the table is used for efficient hotspot detection.

1) Pattern encoding: The mask and hotspot patterns in two-dimensional arrays are transformed into one-dimensional strings in the same way as [6]. Next, the mask and hotspot patterns are divided into k-character substrings and then each substring is encoded to use the reference table for calculating the edit distance matrix. Note that, in the hotspot pattern, since large don't-care characters cannot be calculated in the encoded

form, large don't-care characters are not encoded, and thus they are not included in a k-character substrings. Each k-character substring is encoded by using (1). The terminal substring (of the mask pattern) whose length are less than k is left as an unencoded string. Likewise, in the hotspot pattern, large don't-care characters and the substrings whose length are less than k (including those resulting from the inserted large don't-care characters) are left as unencoded strings.

2) Calculation of edit distance matrix D: Using the encoded patterns, the calculation of the edit distance matrix D is performed by referencing the table. Let us explain the calculation using Figure 14. First, the values of the element of the first row and column are set according to the definition of the edit distance matrix. Next, the region whose upper-left element corresponding to the element $D(1,1)$ (region (1)) is calculated by table-reference. After calculating region (1), the region adjacent to the right of region (1) (region (2)) is calculated. Let region (5) be an example. The input3 of region (5) is from the output1 of region (3), and the input4 of region (5) is from the output2 of region (4). In this way, the encoded outputs of regions can be directly used as inputs to calculate other regions. After table-reference-based calculation, the values of the elements in the bottom row of the edit distance matrix D can be restored by using the value of the left-most element and the differences between adjacent two elements obtained by decoding the output1 of each bottom region.

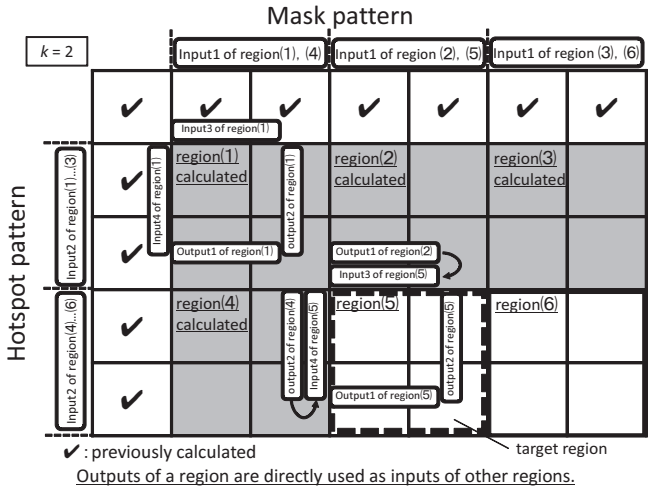

Figure 14. Calculation of edit distance matrix by table reference

We have explained the edit distance calculation ignoring unencoded substrings. Here, let us explain how to handle unencoded mask pattern substrings using Figure 15(a). First, the output2 of the region just before the unencoded mask pattern character is decoded to the differences between adjacent elements in the corresponding column. Next, the values of the elements are restored using the value of the upper-most element and the differences between elements. Then, the values of the elements corresponding to the unencoded substrings are calculated according to the DP-based definition in the same way as [6].

Next, let us explain how to handle unencoded hotspot pattern substrings using Figure 15(b). First, the output1s of the regions just above the unencoded hotspot pattern character are decoded to the differences between adjacent elements in the corresponding row. Next, the values of the elements are restored using the value of the left-most element and the differences between elements. Then, the values of the elements corresponding to the unencoded substrings are calculated according to the DP-based definition in the same way as [6]. Also the substrings before the large don't-care characters are handled in the same way. Since each row of hotspot pattern (in a two-dimensional hotspot pattern) means one large don't-care character, a hotspot pattern string contains multiple large don't-care characters. This calculation is performed for each large don't-care character.

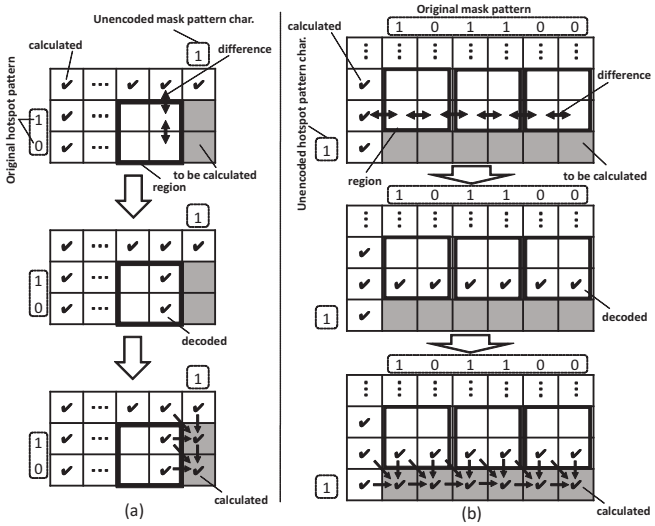

Figure 15. Calculation for unencoded character: (a) mask pattern character, (b) hotspot pattern character

Finally, if the hotspot pattern character corresponding to the bottom row is encoded, the values of the elements in the bottom row of the edit distance matrix D are restored to find hotspot candidates. If the hotspot pattern character is not encoded, the values of the elements in the bottom row of D are calculated according to the definition as mentioned in the previous paragraph.

3) Detection of similar patterns: After calculating the edit distance matrix D, patterns similar to the hotspot pattern are detected from the values of the bottom row of D. The elements with minimal values (less than the user-defined threshold) are identified in the same way as [6]. Each of them corresponds to the terminal character of a hotspot candidate. The initial character can be identified by the terminal character because the length of a hotspot candidate is the same as the given hotspot pattern.

V. EXPERIMENTAL RESULTS

We performed experiments to evaluate our method. In the experiments, we evaluated the execution time of template-matching, the existing method [6], and our proposed method for $k = 1, \ldots 5$, for the same mask pattern (1020×1020 pixels) and the same hotspot pattern (250×250 pixels), on a CentOS (release 6.3 (Final)) PC equipped with Intel Core i7-3770 CPU @ 3.4GHz and 7.6GB memory using gcc 4.4.7.

The experimental results are shown in Figure 16. Our proposed method achieved the better result compared to [6]

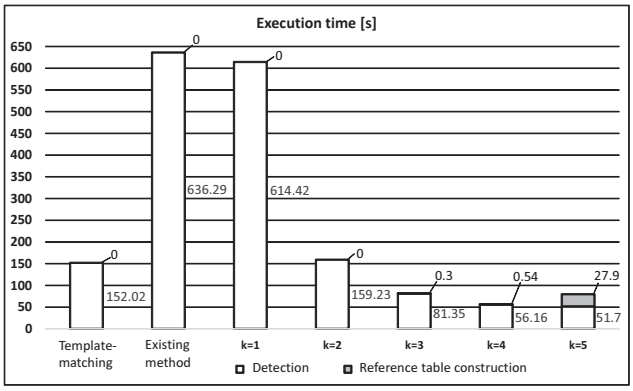

Figure 16. Calculation time

for each setting of k. When $k \geq 3$, our method outperformed template-matching. We confirmed that the calculation time of the edit distance matrix D was drastically reduced.

The calculation time of D is inversely proportional to k^2. Thus, it is expected that the decreasing rate of the calculation time decreases with increasing k. In addition, the time to make the reference table is proportional to the number of combinations of input values (*i.e.*, $2^{2k} \times 3^{2k}$). That is, the time is proportional to an exponential function of k. Therefore, from Figure 16, the sum of the time to make the reference table and the time to calculate D increases when $k \geq 5$. Thus, we conclude that $k = 4$ is best under the experimental environment. The execution time when $k = 4$ was about 1/11 compared to the existing method [6]. Note that once a reference table is made, it can be reused. In such a case, $k \geq 5$ are potentially effective.

VI. Conclusions

In this paper, we proposed and evaluated an efficient hotspot detection method. Experimental results showed that our proposed method found hotspot candidates much faster than the existing one [6] on which our method is based. Our future work includes further improvement of the execution time and improvement of the accuracy of hotspot detection.

References

[1] T. Higashi and Y. Onishi, "Trends in semiconductor lithography technologies and Toshiba's approach," TOSHIBA review, vol. 67, no. 4, pp. 2-6, 2012. (in Japanese)

[2] H. Yao et al., "Efficient range pattern matching algorithm for process-hotspot detection," IET Circuits Devices Syst., vol. 2, issue 1, pp. 2–15, 2008.

[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. the 16th Asia and South Pacific Design Autom. Conf. (ASP-DAC), pp. 775–780, 2011.

[4] J. Wuu, Fedor G. Pikus, and M. Marek-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, pp. 1–8, April 04, 2011.

[5] W. Wen, J. Li, S. Lin, J. Chen, and S. Chang, "A fuzzy-matching model with grid reduction for lithography hotspot detection," IEEE Trans. CAD, vol. 33, no. 11, pp. 1671–1679, Nov. 2014.

[6] S. Tamagawa, R. Fujimoto, M. Inagi, S. Nagayama, and S. Wakabayashi, "A hotspot detection method based on approximate string search," in Proc. the 9th International Conference on Advances in Circuits, Electronics and Micro-electronics, pp. 6–12, 2016.

[7] H. Yaegashi, "Pattern fidelity control in multi-patterning towards 7nm node," in Proc. the IEEE 16th Int. Conf. on Nanotechnology (IEEE-NANO), pp. 452–455, Aug. 2016.

[8] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," Journal of the ACM., vol. 21, issue 1, pp. 168–173, Jan. 1974.

[9] Y. Utan, S. Wakabayashi, and S. Nagayama, "An FPGA-based text search engine for approximate regular expression matching," in Proc. the 2010 International Conference on Field-Programmable Technology, pp. 184–191, Dec. 2010.

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

 ISBN: 978-1-61208-585-2