JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.17, NO.1, FEBRUARY, 2017
https://doi.org/10.5573/JSTS.2017.17.1.086

ISSN(Print) 1598-1657
ISSN(Online) 2233-4866

Automatic Defect Detection from SEM Images of Wafers using Component Tree

Sunghyon Kim[1] and Il-seok Oh[1,2]

Abstract—**In this paper, we propose a novel defect detection method using component tree representations of scanning electron microscopy (SEM) images. The component tree contains rich information about the topological structure of images such as the stiffness of intensity changes, area, and volume of the lobes. This information can be used effectively in detecting suspicious defect areas. A quasi-linear algorithm is available for constructing the component tree and computing these attributes. In this paper, we modify the original component tree algorithm to be suitable for our defect detection application. First, we exclude pixels that are near the ground level during the initial stage of component tree construction. Next, we detect significant lobes based on multiple attributes and edge information. Our experiments performed with actual SEM wafer images show promising results. For a 1000 × 1000 image, the proposed algorithm performed the whole process in 1.36 seconds.**

Index Terms—**Inspection, defect detection, component tree, semiconductor, SEM images**

I. INTRODUCTION

Fast and reliable inspection technology is essential for improving yield and productivity in semiconductor device fabrication [1, 2]. For this purpose, various inspection technologies for silicon wafers have been developed and used. The inspection of silicon wafers to identify defects is usually performed by analyzing scanning electron microscopy (SEM) images of the wafers, but this image processing is made difficult by the diversity and irregularity of the defects, the many noise sources in SEM image generation, and the unpredictable variations in the device fabrication.

Many approaches have been proposed and are being used in actual production processes. Yum, Koo and Kim, in their review of existing analyses of defective patterns [3], classified methods as either automatic detection only or automatic detection and classification, and then described the unique features of each method. Huang and Pan presented a taxonomy of visual inspection algorithms which classified algorithms into four method categories—projection, filter-based, learning-based and hybrid methods [4]. Their taxonomy scheme also categorized the algorithms with respect to three kinds of semiconductor products to which the algorithms have been applied—wafers, thin-film-transistor liquid-crystal displays (TFT-LCDs) and light-emitting diodes. In his review, Xie classified approaches as being either statistical, structural, filter-based, model-based, or color-based [5] and discussed important issues such as the need for standard databases and standard experimental protocols.

The top-hat transformation of the morphological algorithm is a general inspection algorithm in the manufacturing industry. It is simple and effective, but its performance is greatly dependent on the definition of the structure element. Since it uses a peeling operation, a defect region might be divided into several regions and

Manuscript received Oct. 20, 2016; accepted Dec. 27, 2016
[1] Dept. of Nano Science Technology Graduate School, Chonbuk National University.
[2] Dept. of Computer Science and Engineering Graduate School, Chonbuk National University
E-mail : isoh@jbnu.ac.kr

failure is highly probable for elongated defect regions such as scratch defects. Recently, the quad-tree algorithm has been applied to identify defects in SEM images [6]. Recognizing that defective areas tend to be inhomogeneous, this method divides each image block into four quadrants recursively if the inhomogeneity of the block is greater than a certain threshold. This algorithm is effective in finding the approximate locations of defects, but pre-processing steps, such as edge relaxation, is essential to achieve good performance. In addition, an additional process is necessary to obtain more information about the detected defect areas. The algorithm is likely to split a defect region into different quadrants, which can result in a false positive or false negative. Additionally, neither the top-hat or quad-tree algorithm can properly deal with images containing uneven background.

In this paper, we propose a novel defect detection algorithm that uses the component tree as a data structure, whereby SEM images can be viewed as a topographic map. By analyzing this map, we can extract rich information that is useful for detecting suspicious defect regions. The component tree is an adequate framework for representing topographic structures and provides an efficient algorithm for computing the rich information. Here we present a modification of the component tree algorithm that is suitable for extracting several attributes such as area, height, volume, and stiffness. These accurate attribute values can be obtained even from images with uneven background brightness. The attributes can then be used to determine highly probable defect regions with complex shapes such as elongation. We note that the conventional methods, such as the morphological and quad-tree algorithms, cannot process images with uneven brightness and complex shapes. We performed experiments with actual SEM images, printed electronic images, and generated promising results. Another merit of the proposed algorithm is the high speed of its processing. For a 1000 × 1000 image, the proposed algorithm performed the whole process in 1.36 seconds.

II. METHODS

The input of our algorithm is the difference image obtained by aligning two segments of an SEM image or two different SEM images.

In this paper, we modify the original component tree algorithm to be able to perform defect detection in various ways. We describe these modifications in the following sections.

1. Building the Component Tree

We use the algorithm proposed by Najman to build a component tree in quasi-linear time [8]. A component is the set of connected pixels that survives after thresholding. When we increase the threshold level, the number of components increases and an inclusion relation appears between the previous and current components. In this way, the components are organized into a tree structure. Fig. 1 shows an example of a component tree.

This algorithm, which is based on Tarjan's union-find procedure, first sorts the pixels into non-decreasing order and then processes the pixels in that order by inserting them into the tree one at a time.

2. Building Component Tree Excluding Dark Pixels

Regions of suspicious defects are usually brighter than normal regions and they occupy very small parts of the entire image. Areas consisting of sufficiently low gray-level pixels are unlikely to be defects. Therefore, in the first stage of bucket sorting, we may exclude pixels whose intensity is lower than the preset ground level. This simple idea results in a great speed up with no degradation in the defect identification.

Since this modification generates an incomplete component tree consisting of several subtrees, in the final stage of the algorithm we categorize these subtrees as children of a virtual root node. Fig. 1(e) shows the results of the modified algorithm when the ground level is set to 100.

3. Computing Attribute Values and Detecting Defects

The authors in [8] describe an efficient algorithm for computing the attribute values of the area and volume of each node in the component tree and identifying significant lobes by the volume attribute. To identify significant lobes, the original algorithm repeatedly

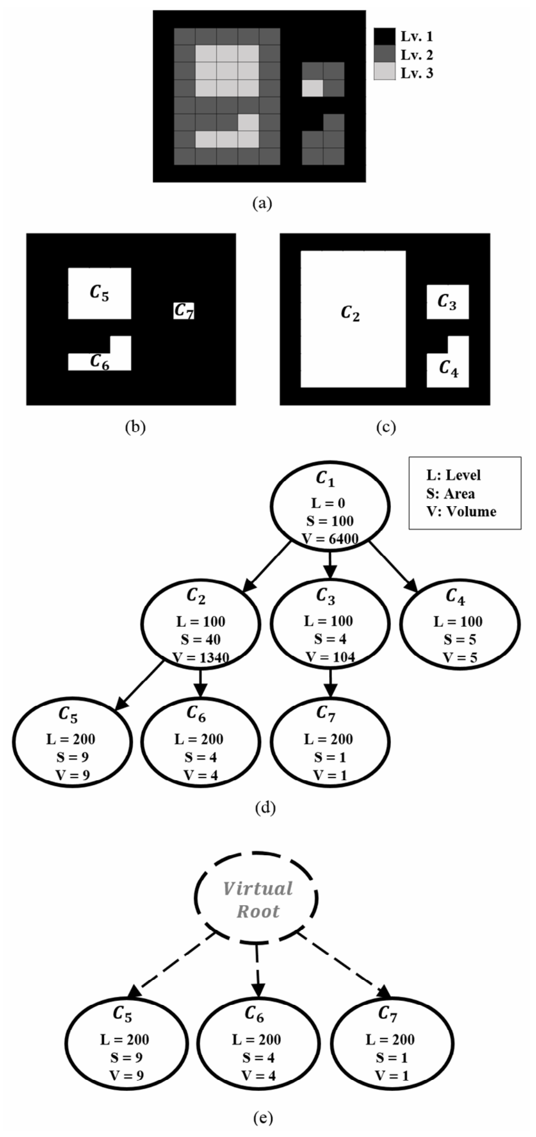
Lv. 1
Lv. 2
Lv. 3
(a)
C_5
C_7
C_6
C_2
C_3
C_4
(b)
(c)
C_1 L = 0 S = 100 V = 6400
L: Level
S: Area
V: Volume
C_2 L = 100 S = 40 V = 1340
C_3 L = 100 S = 4 V = 104
C_4 L = 100 S = 5 V = 5
C_5 L = 200 S = 9 V = 9
C_6 L = 200 S = 4 V = 4
C_7 L = 200 S = 1 V = 1
(d)
Virtual Root
C_5 L = 200 S = 9 V = 9
C_6 L = 200 S = 4 V = 4
C_7 L = 200 S = 1 V = 1
(e)

Fig. 1. (a) Example image with three gray levels, (b), (c) Binary images after thresholding at 200 and 100, respectively, (d) The component tree of (a), (e) The modified component tree of (a) composed of pixels whose gray levels are higher than 100.

deletes leaf nodes in decreasing order of volume. In this process, a parent that loses every child becomes a leaf. The algorithm repeats this process until *n* leaf nodes remain, with *n* being the desired number of lobes. The pseudo code of the algorithm can be sketched as follows. Here, we provide only a brief sketch, and a more detailed description of the algorithm may be found in [8].

```
Algorithm 1: Finding significant lobes (original algorithm in [8])
Input: n (number of significant lobes desired),
component tree T
Output: n leaf nodes representing the most significant lobes
1. while number of leaf nodes in T > n
2.     select a leaf node with the smallest volume;
3.     delete the selected leaf node from T;
4. output the leaf nodes;
```

We consider the remaining *n* leaf nodes to be significant lobes and, as such, defective areas. Since the original algorithm uses only the volume attribute to find significant lobes, it is seriously affected by the number of significant lobes *n*. If *n* is too small, the algorithm will delete most of the nodes and identify nodes near the root of the tree as significant lobes. In many cases, the result is not what we expect, as seen in Fig. 4(a)-(c). Usually the remaining lobes are those resulting from the merging of non-defective areas. In order to realize good performance, we must set *n* to be close to the number of actual defects in the difference image. However, it is very difficult to determine a priori the number of defects in an image.

To solve this problem, we use not only the volume attribute but also other attributes such as area and gradient magnitude. We describe these details in the following section.

4. Using Multiple Attributes and Edge Information

In this section, we describe two algorithms we use to solve the problems noted above. The first deals with an easy case of detecting defects of similar sizes and shapes, and the second deals with more a complicated case involving defects of various sizes and shapes.

Case 1. Defects of nearly fixed sizes and shapes

Fig. 2(a) and (b) show patches of an SEM wafer image provided by a semiconductor manufacturing company and Fig. 2(c) shows their difference image. We see that the defects have nearly uniform sizes and shapes. In this simple case, we could define an approximate defect size, which is related to the area attribute. The modified algorithm uses both the volume and area attributes, and repeats the operation of deleting leaf nodes in decreasing order of volume, as in Algorithm 1. The algorithm exempts leaf nodes from deletion whose parent area is

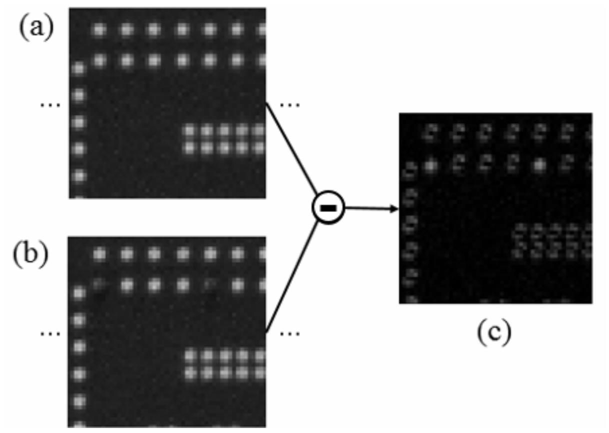

Fig. 2. (a), (b) Two aligned wafer SEM image segments, (c) Difference image.

bigger than a, where a is a predefined area threshold. The user sets a to the minimum area of a defect. When no more leaf nodes are found, the process is complete. Then, the n nodes with the largest volume are output as suspicious defects. **Algorithm 2** shows the pseudo code of the above process.

Algorithm 2: Finding significant lobes by area attribute
Input: n (number of significant lobes desired), area threshold a, component tree T
Output: n leaf nodes representing the most significant lobes

```
1. while unprocessed leaf node is in $T$
2.     select a leaf node with the smallest volume;
3.     if (area of parent of the leaf node $< a$)
           delete the leaf node from $T$;
4.     else push the leaf node into list $L$;
5. output $n$ nodes in $L$ with the largest volume;
```

Case 2. Defects of various sizes and shapes

Fig. 3 shows an example test image of a printed pattern provided by a visual inspection company. The original image is very large, so we cut out the defect areas and pasted them into one image, Fig. 3, for testing. Note that the defects have different sizes and shapes as well as uneven brightness. Careful inspection reveals that the background also has uneven brightness. If we apply Algorithm 2 to this case, performance would be very poor because it is almost impossible to capture all the defects of various sizes.

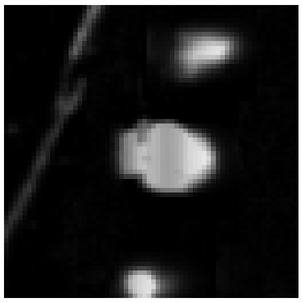

Fig. 3. Example patch from a printed pattern image.

Thus, we consider edge information since the edges can indicate the contours of the defects. We may choose to detect edges and extract regions by identifying closed contours. However, this approach fails due to the phenomenon of fragmented edge segments. Instead, we decided to embed the edge information as an attribute in the component tree. Two options are available—we can use a binary edge map or a gradient magnitude map. Due to the larger computational demand and increased noise associated with a binary edge map, we opted for the gradient magnitude map. Generally, the gradient magnitude reaches a peak near the object contour. So, the proposed algorithm traces the path from a leaf node to the root and attempts to identify the node with the maximum gradient attribute.

During tree construction, a component, i.e., a node, also computes the gradient attribute by averaging the gradient magnitude map obtained by the use of a Sobel mask.

The algorithm visits leaf nodes in decreasing order of volume and traces upward to the root node. It identifies nodes with the maximum gradient attribute value and saves these nodes to a list. Then the n nodes with the largest volume are output as suspicious defects. Algorithm 3 shows the pseudo code of the above process.

Algorithm 3: Finding significant lobes by edge-gradient attribute
Input: n (number of significant lobes desired), component tree T
Output: n nodes representing the most significant lobes

```
1. while unprocessed leaf node in $T$
2.     select an unvisited leaf node with the smallest volume;
3.     trace upward to the root and search for the node
```

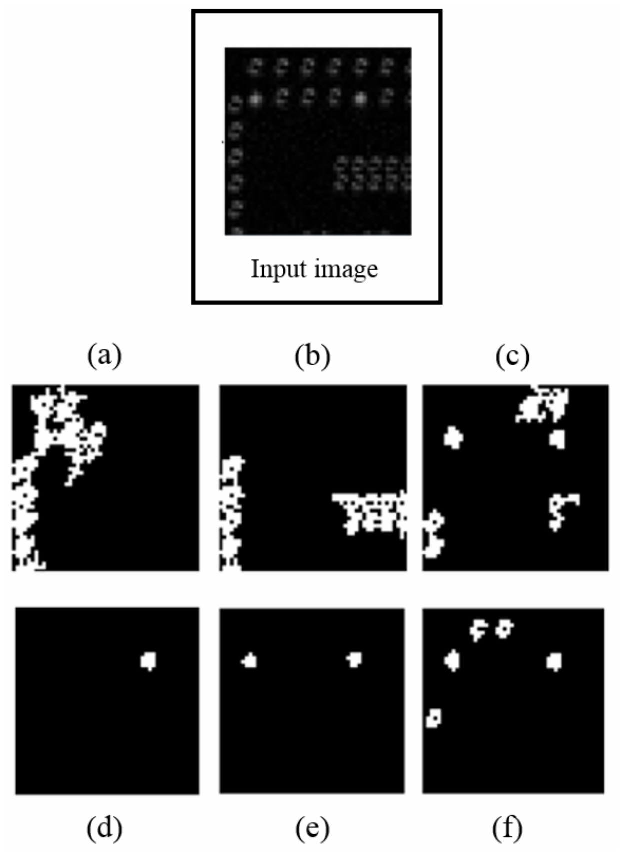

Fig. 4. (a), (b), and (c) One, two, and five most significant lobes of the input image using the original algorithm, (d), (e), and (f) One, two, and five most significant lobes of the input image using the modified algorithm, Algorithm 2.

with the largest gradient attribute;

4. push the node into list L;

5. output n nodes in L with the largest volume;

III. Experiments and Discussions

1. Experimental Results

We implemented the algorithm in C++ language and tested it on a 3.30-GHz PC. Fig. 4 shows the suspicious defects of the input image identified by the original algorithm (Algorithm 1) and Algorithm 2 described above. With the original algorithm, neighboring insignificant lobes are merged into one lobe, with the resultant lobes having a large volume, which were then detected as defects. In contrast, the proposed algorithm correctly detected the candidate defects.

Fig. 5 compares the computation time of the original algorithm with those of Algorithm 2, which achieved about a tenfold speed up. For a 1000 × 1000 image, the proposed algorithm ran the whole process in 1.36 seconds. Via parallelism to build the component tree, further speed up is easily achieved by the use of multi-cores or GPUs.

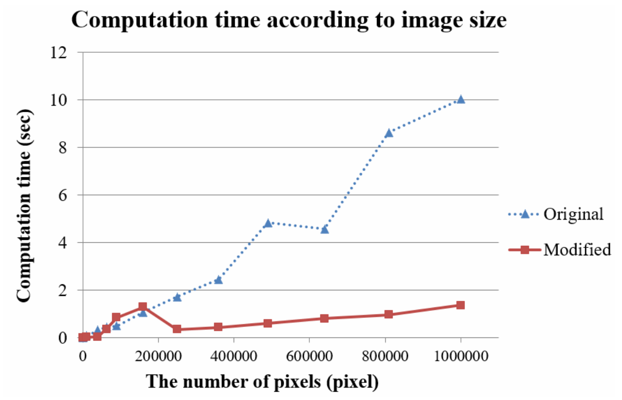

Fig. 5. Computation time of whole process according to image size.

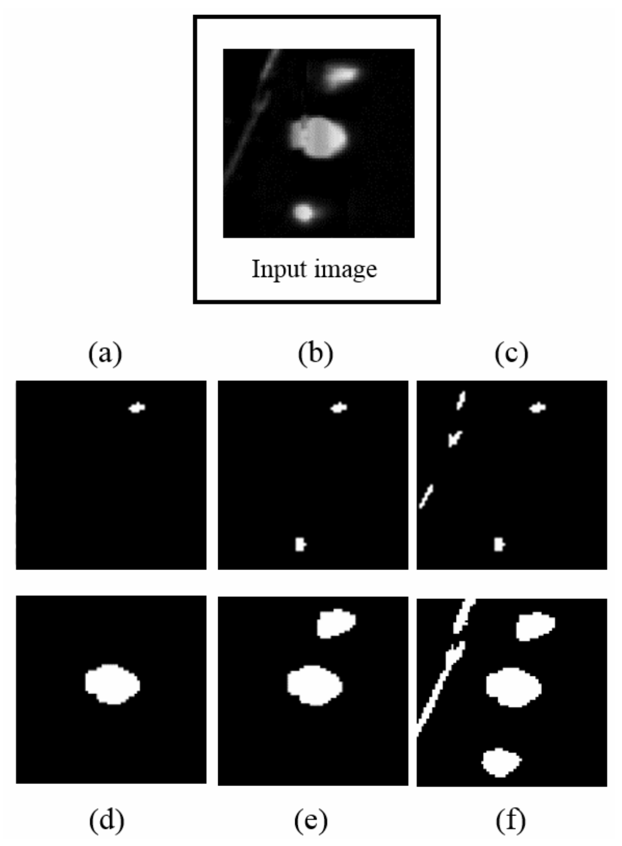

Fig. 6. (a), (b), and (c) One, two, and five most significant lobes of the input image using Algorithm 2, (d), (e), and (f) One, two, and five most significant lobes of the input image using Algorithm 3.

Fig. 6 shows the suspicious defects of the input image produced by Algorithms 2 and 3 described in section 2.4. The output from Algorithm 2 is not satisfactory. In contrast, Algorithm 3 correctly detects the candidate defects. Note that the detection results are satisfactory for a variety of shapes such as elongation due to a scratch and for uneven brightness.

2. Comparision of Three Algorithms

We noted above the applicable images of the three algorithms. Here, we supplement those discussions. Algorithm 1 was successful for cases in which the defect area and background have a clear separation of high contrast and the designated number of defects N is properly set to be similar to the actual number of defects. However, in our experiments with real images, noises present during image acquisition and small blobs due to slight misalignments caused failure. Setting N to be too small caused the algorithm to trace upward near the root, thus resulting in a high computation time.

The experiments with various images showed that Algorithm 2 works stably in situations in which the defect shapes and sizes are homogeneous. Since Algorithm 2 works without edge information and is faster than Algorithm 3, we highly recommend the use of Algorithm 2 in this situation. As expected, Algorithm 3 works well when heterogeneous defects exist in an image, but the extra edge extraction processing slows its performance. A node in the tree contains rich information including the attributes of area, volume, roundness, elongation, and slope stiffness. As such, Algorithm 3 can be readily extended to particular defect type classifications.

3. Comparision with Quad-tree Algorithm

We implemented the quad-tree algorithm in [6] and compared it with the proposed algorithms. Quad-tree algorithm has two essential parameters, adaptive decomposition threshold(ADT) and defect block size Since the default parameter setting produced very poor results, we tried to find out the optimal values. Fig. 7 shows the results obtained by applying the quad-tree algorithm to the input image in Fig. 6. It illustrates effects of varying the parameter value.

In Fig. 7, we can observe some missing and merged defects. The quad-tree algorithm is susceptible to parameter setting. The rightmost figure shows the optimal setting.

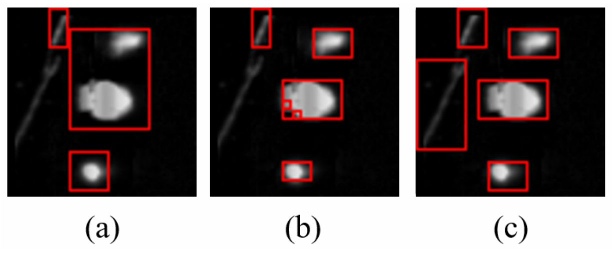

Fig. 7. Detection result of the input image I in Fig. 6 using quad-tree algorithm with various parameter values (a) $ADT = mean(I)+3\times std(I)$, (b) $ADT = mean(I)+3\times std(I)$, (c) $ADT = mean(I)+std(I)$.

Table 1. Comparison between component tree algorithms and quad-tree algorithm

	Component tree		**Quad-tree**
	Algorithm 2	Algorithm 3	
Image size	Arbitrary		$2^n \times 2^n$
Prior knowledge	Blob size	None	None
Sensitivity to noise	Small		Large
Output	Blobs		Overlapping boxes
Features	Rich (area, volume, brightness, stiffness, etc.)		None

Table 1 compares the proposed algorithms and the quad-tree algorithm in various aspects. Due to the recursive partition operation into four quadrants, the quad-tree algorithm requires the images with $2^n \times 2^n$ size. For the images violating the size requirement, rows and column should be appended. The proposed algorithms have the advantage of being less sensitive to noises than the quad-tree algorithm. To reduce the noise effect, the quad-tree algorithm performs a preprocessing. The output of the proposed algorithms is the blobs cutting out the potential defects. In addition, valuable features are automatically produced. The features can be used as input to a classifier such as multi-layer perceptron. On the contrary, the quad-tree algorithm indicates the potential defects by overlapping boxes. So additional post-processing should be developed to cut out the defect area and extract features. The quad-tree algorithm is faster than the proposed algorithms. However the speed comparison is not shown since their outputs are different. The quad-tree algorithm should further perform a post-processing stage to get blobs and extract features.

IV. Conclusions

Defects occurring on uneven background and having various sizes and shapes are difficult to detect by conventional methods. To overcome these difficulties and process large images quickly, in this paper, we propose a novel method based on the component tree data structure. This method considers the image as a topographic map and builds a component tree in quasi-linear time with respect to various attributes. These attributes are used to define and detect suspicious defect regions. Recognizing that the significant lobes detected by the original algorithm do not provide satisfactory results, we modified the algorithm to better detect defects. We proposed two algorithms, one for the easy case of detecting defects of somewhat uniform size and shape and the other for the more difficult case of defects with various sizes and shape. We carried out experiments with actual images, which showed promising results for both cases. The proposed algorithm is also very fast due to the quasi-linearity of the component tree.

A limitations of our proposed method is that, as yet, the performance evaluation has been only qualitative. To perform a quantitative assessment, a large dataset divided into training and test sets must be prepared and every example image properly tagged according to defect type by a highly qualified field expert. While Lee and Lee used their own dataset to quantitatively evaluate their quad-tree algorithm [6], their dataset is not available to the public. Finally, there exists no standard dataset or standard evaluation protocol. We may ascribe this situation to the fact that the objects to be inspected are so diverse—ranging from microscopic materials such as the semiconductor to large surfaces such as steel plates. Another reason seems to be the secrecy of companies who are highly competitive. However, in order to adopt state-of-the-art technologies, such as machine learning, large datasets and standard performance protocols are essential. We are supported in our argument by Xie's remark in [5] that “There is also a clear need of some standard datasets and well-defined experimental protocols in order to carry out fair comparative analysis.”

References

[1] R. T. Chin and C. A. Harlow, “Automated Visual Inspection: A survey,” *Pattern Analysis and Machine Intelligence*, *IEEE Transactions on,* Vol.4, No.6, pp.557-573, Nov., 1982.

[2] R. T. Chin, “Automated visual inspection: 1981 to 1987,” *Computer Vision, Graphics and Image Processing*, Vol.41, Issue 3, pp.346-381, Mar., 1988.

[3] B.-J Yum, J. H. Koo and S.-J Kim, “Analysis of Defective Patterns on Wafers in Semiconductor Manufacturing: A Bibliographical Review,” *Automation Science and Engineering, 8th IEEE International Conference on*, pp.86-90, Aug., 2012.

[4] S.-H. Huang and Y.-C. Pan, “Automated visual inspection in the semiconductor industry: A survey,” *Computers in Industry*, Vol.66, pp.1-10, 2015.

[5] X. Xie, “A Review of Recent Advances in Surface Defect Detection using Texture analysis Techniques,” *Electronic Letters on Computer Vision and Image Analysis*, Vol.7, No.3, pp.1-22, 2008.

[6] Y. Lee and J. Lee, “Accurate Automatic Defect Detection Method Using Quadtree Decomposition on SEM Images,” *Semiconductor Manufacturing, IEEE Transactions on*, Vol. 27, No.2, pp.223-231, May, 2014.

[7] W.-C. Li and D.-M. Tsai, “Defect Inspection in Low-Contrast LCD Images Using Hough Transform-Based Nonstationary Line Detection,” *Industrial Informatics, IEEE Transactions on*, Vol.7, No.1, pp.136-147, Feb., 2011.

[8] L. Najman and M. Couprie, “Building the Component Tree in Quasi-Linear Time,” *Image Processing, IEEE Transactions on*, Vol.15, No.11, pp.3531-3539, Nov., 2006.

Sung-hyon Kim received a B.S. degree from the Department of Electronic Engineering at Chonbuk National University, Jeonju, Korea, in 2011 and is currently pursuing her Ph.D. in the Department of Nano Science Technology. She is currently a research student of KRISS (Korea Research Institute of Standards and Science). Her interests include inspection and particle segmentation & analysis using image processing and computer vision techniques.

Il-seok Oh received a B.S. degree in computer engineering from Seoul National University, Korea, and his PhD in computer science from KAIST (Korea Advanced Institute of Science and Technology), Korea, in 1984 and 1992, respectively. He was a visiting professor at Concordia University, Canada (1996) and the University of California, Irvine (2012). His research interests include machine learning, pattern recognition, and computer vision.