Evolving an Automatic Defect Classification Tool

Assaf Glazer[1,2] and Moshe Sipper[1]

[1] Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel
`www.moshesipper.com`
[2] Applied Materials, Inc., Rehovot, Israel[⋆]

Abstract. Automatic Defect Classification (ADC) is a well-developed technology for inspection and measurement of defects on patterned wafers in the semiconductors industry. The poor training data and its high dimensionality in the feature space render the defect-classification task hard to solve. In addition, the continuously changing environment—comprising both new and obsolescent defect types encountered during an imaging machine's lifetime—require constant human intervention, limiting the technology's effectiveness. In this paper we design an evolutionary classification tool, based on genetic algorithms (GAs), to replace the manual bottleneck and the limited human optimization capabilities. We show that our GA-based models attain significantly better classification performance, coupled with lower complexity, with respect to the human-based model and a heavy random search model.

1 Introduction

Traditional classification approaches suffer from a problem of poor generalization on image classification tasks. In this paper we focus on the image classification task for the semiconductors industry. During the production process within a fab, which is a customer's wafer fabrication facility, we would like to automatically find and characterize defects, and determine their sources. Classified defects may be fixed, thus increasing the yield of the wafer production process. The motivation for this paper is the industry's demand for better classification results with higher throughput (wafers produced per hour), and the desire for an automated process with minimal human intervention. Poor data, and a deceptive environment in the fab where the classification problem itself varies over time, renders the ADC task hard to solve. Our primary goal is to provide a solution for the above challenges by using genetic-algorithm techniques, providing an evolutionary classification tool that automatically optimizes itself across a machine's lifetime and adapts to the changing environment inside the fab.

[⋆] This research was supported in part by the IMG4 consortium (4th Generation Imaging Machines) of the Israeli Ministry of Industry, Trade, and Employment, and also by the Lynne and William Frankel Center for Computer Science.

Different models of Automatic Defect Classification (ADC) tools already exist in different fabs, using a wide range of classification models. Most of the ADC models are based on *machine learning* [3] techniques, which is a broad subfield of *artificial intelligence*. In our research we would like to optimize a given ADC classifier, which is based on a radial basis function neural network (RBFN) with Gaussian radial basis function kernels, by applying a genetic algorithm to select kernels that are used within the RBFN hidden units. The research has taken place in the SEM division, Applied Materials, Inc. (AMAT) and Ben-Gurion University. We use the SEMVision classification tool, which is based on a RBFN, as a benchmark for our research.

In Section 2 we introduce the defect classification problem. In Section 3 we provide background on *radial basis function neural networks* (RBFN). In Section 4, we present a reference model for the SEMVision ADC tool we intend to use later. In Section 5 we describe the basic evolutionary model. An additional model of heavy random search is defined for comparison with our model. In Section 6 we introduce the enhanced evolutionary model for the defect classification problem. Finally, Section 7 summarizes the results obtained in our research and suggests some directions for future research.

2 The Defect Classification Problem

The goal of the defect classification process is quite simple: given an image, classify the defect type found in the image. The need for an image classification tool arises in various fields; in our research we focus on the defect classification process in the semiconductors industry. Semiconductor wafer manufacturers invest much of their time in isolating the causes of yield-impacting defects during the lithographic printing and processing of integrated circuits on wafers. Automatic Defect Classification (ADC) automates the slow manual process of defect review and classification during optical microscopy and scanning electron microscopy (SEM). The ADC machine is a key step in the identification of the root cause of manufacturing problems. A fast, accurate, and reliable ADC tool is required. However, though the problem definition is simple, its solution involves many challenges. During the fab's lifetime new defects appear, and old defects become obsolete; the ADC model has to adapt itself to this changing environment. Additional challenges, such as poor data with inaccurate classified samples and high throughput demands from the customers, require a fast automated and reliable ADC tool.

The automatic defect classification (ADC) problem can be defined as a subset selection problem. Given a training set TR of pre-classified images and a pre-classified validation set VL, we would like to find an optimal subset $S \subseteq TR$, which maximizes the classification rate of the given ADC classifier. Each exemplar in the training set S is translated into a hidden unit in the classifier. For example, if we use a Gaussian kernel function in a *RBFN*, then $\forall t \in S$ we generate a hidden unit of a Gaussian function with a *mean* $\mu = t$ (in *RBFN* models we usually define a fixed *variance* σ for all hidden units). Eventually, the size of the given subset S is equal to the number of units inside the RBFN hidden layer. We briefly discuss the RBFN hidden units in Section 3. In addition, we keep an independent test set $TEST$ of pre-classified images to estimate the classification rate that can be achieved in a real-time application with the optimized classifier. The test set $TEST$ can also allow us to gauge the quality of the achieved solution on real field data.

We view each of our customers' fabrication factories as a 'greenhouse' for images of defects. Each greenhouse generates defect prototypes with some common characteristics at a specific time and place, and we have to fit and optimize the ADC tool to the current fab with the current process and the current time stamp. As a result, the objective function can change along the ADC machine's lifetime by modifying the input data sets TR, VL, and $TEST$ during the classification process. Therefore, our method should be based on an *anytime algorithm*, which generates the best solution within the scope of available data that have been explored up to the allowed time. Thus, we can change the data sets during the ADC's lifetime and the algorithm can fit itself to the new environment.

Usually, the classification rate is determined as the *accuracy* of the classified result, i.e., the percentage of the defects classified correctly, or as the *purity* of the classified result. *Purity* is the exactness of the classification, i.e., the fraction of defects classified correctly with respect to the number of total classified images, not including 'Unknown Defect' classification results. We can also use a hybrid definition involving both *accuracy* and *purity* results.

3 Radial Basis Function Neural Networks (RBFNs)

As shown in Figure 1, a RBFN is a three-layer, feed-forward network, with each layer fully connected to the next layer. The RBFN consists of an input layer L_0, a hidden layer L_1, and an output layer L_2. The k-dimensional input vector enters the RBFN through the k units of input layer L_0, and passes through weighted edges $w_{i,j}$ to the hidden units in hidden layer L_1. A kernel threshold function is activated in each of the hidden units, passing the activation result through additional weighted edges $w_{j,l}$ to the output layer L_2. The hidden layer consists of a set of radial basis functions. Associated with each hidden-layer node is a parameter vector c_i called a center. For each hidden unit calculate the Euclidean distance r between c_i and the input vector, and pass the result to the kernel function. Different kernel functions can be chosen as the network activation function, such as *Gaussian*, *Multiquadric*, and *Thin plate spline* functions [10].

Training an RBFN consists of three main stages [3]: the **first** stage is *network initialization*, which defines the selected features for the input layer and determines the number of centers and the radii c_i of the kernel functions in the hidden layer. The **second** stage is *obtaining the weights for the output layer*: once we have determined the network-initialization parameters we can seek the weights for the output layer using least-squares error function minimization, similar to single-layer networks, e.g., *QR decomposition.* The **third** stage is *iterative optimization*, where several methods can be used for RBFN optimization [3].

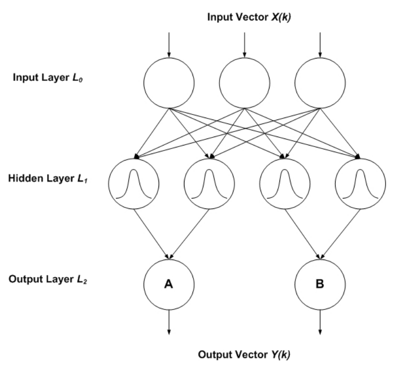

Fig. 1. Radial basis function neural network (RBFN)

Genetic algorithms can be used for different optimization tasks for RBFNs, including kernel-function optimization [1, 4, 7, 8, 9, 11], architecture optimization [5, 6, 11], and training-set optimization [6]. The evolutionary process is especially advantageous when the search space is large and the objective function is non-differentiable, e.g., subset selection for training-set optimization. We also find the GA solution to be advantageous when a deceptive objective function is involved, e.g., as caused due to a changing environment during the ADC machine's lifetime.

4 Reference Model for SEMVision ADC Tool

In order to understand the problem we introduce a Matlab-based model, using the *netlab toolbox*, defined as a reference for the SEMVision ADC model. The data set we used has 2613 individuals from 9 different defect classes, each one represented by a vector with 76 features. The data set is separated into a training set of size 605 and a validation set of size 2008. We ran a RBFN with *Gaussian kernel* functions. We used a principal components analysis (PCA) algorithm for the feature selection part. The *k-means* algorithm was used to select the centers c_i for each of the *Gaussian kernels* in the hidden units, and we used a fixed width for the Gaussian variance. After setting the centers of the kernels we optimized the network with an *EM (expectation maximization) algorithm* [2] using a *least-squares* error function.

The results showed no convergence. We tried a *cross-validation* technique in order to optimize the number of features and kernels with no success. A balancing method also proved unsuccessful in avoiding a trivial solution. If we use the whole training set, as the SEMVision ADC does, the classification accuracy reaches 85.7, a much higher performance than in the reference model. We can see that traditional *Machine Learning* algorithms for kernel selection, such as *cross-validation* for selecting the number of kernels, and *k-means* for clustering, attain low performance compared to the SEMVision ADC.

We conclude from the reference model that the data is too poor. The literature suggests that the data we used should be ten times more abundant than the network complexity, where the complexity of a network is measured as the total number of connected edges [3]. In our case they are almost equal, and we have no other choice than using the training data set, as is, for the centers in the hidden neurons.

5 Basic Evolutionary Model

When the data is poor and inaccurate, known methods for RBFN optimization do not converge to a reasonable solution, and we have no other choice than using the whole training set as kernels for the RBFN. When the data is too poor for using a clustering method, using real kernels in the classifier—which represent real defects—can reduce the risk of overfitting. However, though using the entire training set as kernels for the RBFN achieves reasonable results, this method suffers from a stability problem and demands frequent skilled Customer Engineer (CE) support, since every classified sample has an immediate impact on the classified product of an image with the same characteristics. Erroneous flyers in the training set can dramatically reduce the accuracy of the classifier, and the changing environment requires constant CE support. Moreover, the sampling of images for the training set is done manually by humans, who can hardly optimize the selected samples for the ADC tool with no computational support. Perhaps we can attain better accuracy by using a subset from the training data set.

In the model we use we would like to optimize the subset of kernels from the training set for our ADC tool such that we attain better results with less computational complexity: the fewer the kernels in the ADC, the less complex is the ADC model, and the higher the throughput and generalization of our model.

The problem of subset selection from a training set is hard to solve. Under the assumption that the data set is too poor for learning by using known optimization methods, this problem of subset selection is *NP-Hard* by a simple reduction from the *subset sum problem.*

The GA. We introduce a genetic algorithm for this optimization, with classifier accuracy being the fitness. The GA is defined as follows:

- *Initialization.* Each generation comprises a population set π of 50 individual genomes. We use random initialization with uniform distribution to construct the first generation. Each bit in the genome is set to either 0 or 1 with probability 0.5.
- *Genome representation.* Let $TR = \{t_1, ..., t_m\}$ be a training set of m classified images. We use a binary encoding for the genome representation, such that each genome represents a subset of the initial training set, e.g., the genome $S = [01101001...]$ represents the subset $S \subset TR, S = \{t_2, t_3, t_5, t_8, ...\}$. The genome length is equal to the size of the training set.
- *Fitness function.* In each generation we calculate the fitness function f_{fit} per individual. Each genome represents a subset S of the initial training set TR. The fitness function is defined as the accuracy rate of the ADC classifier.

- *Selection, crossover, mutation.* We use the entire population π as the mating set in order to maintain diversity. In our case the difference in fitness between individuals is small, and the use of the *fitness-proportionate selection* method might reduce selection pressure. Therefore, we use the *rank selection* mechanism. We apply an *elitism selection* mechanism (of one individual per generation), thus maintaining the best individual in the population set π for the next generation. *Single-point crossover* is performed with crossover rate $p_{cross} = 0.8$. Each individual is subjected to mutation. This operation inverts each bit in the genome with a probability of 0.05.
- *Termination criterion.* Though this is an anytime algorithm, for empirical reasons, we stop the algorithm after 50 generations.

To compare with our GA we run a *heavy random* stress test for kernel selection. Although random selection of kernels from the training set is apparently a naive choice, sometimes it is more justifiable than the more sophisticated techniques. The reason is that, especially in small sample size problems, intricate algorithms can easily lead to overfitting [6].

The Experiment. We have already seen that we can attain better performance on the same data set by using the SEMVision ADC, than by using the reference model. Therefore, in order to reduce overall run time, we allowed ourselves to use a smaller data set than that used for the reference model above.

The data consists of 1905 individuals from 5 different classes. Each of the individuals represents a defect image with 76 features. The data is separated into a training set of size 498, a validation set of size 946, and a test set of size 461. In addition to the genetic algorithm, we ran a *heavy random* stress test for kernel selection. The application runs on an XP OS with Xeon Dual-Core computer and 2Gb RAM.

Results. If we use the whole training set, as the CE currently does, classification accuracy reaches 0.870. With our GA we attained 0.897—with less than half the number of kernels, i.e., achieving a 0.297 improvement with half the complexity. The *heavy random* stress test shows poor results of 0.883 with almost no convergence over 50 generations, compared to the evolutionary model. Moreover, we left the *heavy random* to run for 150 generations, and no significant improvement was found. The total GA runtime is approximately 60 hours. Three separate runs were made for this basic model, showing almost identical behavior.

An improvement from 87.0% to 89.7% is much more impressive than, say, from 77.0% to 79.7%, for instance. Moreover, using 193 kernels found by the GA instead of 498 used by the manual method (CE), is a major improvement in terms of throughput, since the ADC buildup time and its classification processing time is exponential with the number of kernels inside the RBFN.

The total runtime of approximately 60 hours can be reduced dramatically with several optimization methods, such as parallel fitness evaluation, an efficient way to modify the ADC tool to learn a new training set without a new reconstruction on each evaluation, parsing elimination, and hardware improvement. We must remember that this optimization algorithm is an off-line problem, where the computational efforts can take place during the computer's idle time. Moreover, monitoring and optimizing the classification process in spaces of 2-3 day is good enough for most of the fabs.

6 Enhanced Evolutionary Model

Given our experience with the basic evolutionary model, and given the characteristics of the classification problem—poor and inaccurate data in a changing environment—and the requirement for an accurate, fast, and automatic classification tool, we formulate the following guidelines for a solution, in order to attain even better results:

– **Anytime algorithm.** *genetic algorithms* can be considered as a form of *anytime algorithm.* Our model must provide the best solution it can find by using the data that have been explored until the current moment.
– **Optimization of training set.** Our model should find the optimized subset from the given training set.
– **Generic model.** We treat the SEMVision ADC model as a black box. This way our optimization model can be used with different types of classification methods.
– **Reduced complexity.** The use of a subset, instead of the whole training set, reduces the complexity, improves throughput, and increases generalization.
– **Robust solution with no human intervention.** Our only requirement from the customer is a daily classification of a constant number of samples, which is a reasonable demand.

The Enhanced GA. The input for our model consists of a training set TR, a validation set VL, and a test data set $TEST$. We assume the data set can increase and change during the fab's lifetime. The main *genetic engine* module is responsible for the entire problem optimization. The input for the *genetic engine* module is the data sets, and the output is an optimal subset $S \subseteq TR$, which maximizes the classification rate of the given ADC classifier—as measured on the test data set $TEST$. The *genetic engine* module comprises two separate GAs working in parallel: the *defect controller* module is responsible for defining the problem we are trying to solve. The output of the *defect controller* module is a *hint* array with the relevant samples of the training set. The *ADC enhancer* module is responsible for optimizing the current problem. The *hint* array, produced by the *defect controller*, is part of the input for the *ADC enhancer*. Basically, we run two separate GAs in parallel—*defect controller* and *ADC enhancer*—the latter constantly receiving hints from the former, which help improve its performance.

The output of the *defect controller* GA module is a *hint* array with the relevant samples of the training set. If the training set consists of n elements, the output of the *defect controller* module is a *hint* array of size n with the fraction of the kernels' relevances inside the given training set TR. The *hint* array can dynamically change during the evolutionary process. The *ADC Enhancer* GA module uses the *hint* array to achieve better problem optimization. The output of the *ADC Enhancer* module is an optimal subset $S \subseteq TR$, which maximizes the classification rate on the validation data set VL. The use of the *hint* array in the *ADC Enhancer* module is through the mutation operation, as shown in Figure 2.

```
Mutate($gene$, $hint$)

parameter(s): $gene$ – the genome
parameter(s): $hint$ – the hint array
output: the mutated genome

$Initialization$ :
$i \leftarrow 0$
while $i < hint.length$
  do
    if $random < mutation\ rate$
      then
        if $random < hint(i)$
          then $gene(i) \leftarrow 1$
          else $gene(i) \leftarrow 0$
    $i \leftarrow i + 1$
return ($gene$)
```

Fig. 2. Pseudocode of the mutation operation

For example, if one of the samples inside the training set becomes obsolete, then its fraction will drop in the *hint* array, and the flipping probability for the equivalent bit inside the mutated genome will decrease as well. This way we can also explore new and old defects in the classification process.

The *defect controller* module dynamically produces the *hint* array in the following manner: a subset with the *best* result during a pre-defined window of time is maintained, i.e., the last fixed number of generations backward. We can define *best* results in two different ways:

1. Classification rate above a predefined value.
2. Classification rate in top of predefined fixed percentage of the results from the fixed number of generations backward.

For each of the kernels in the training set we calculate the fraction of usage inside the *best* subsets from the relevant generations. The *hint* itself is an array with the fraction of kernels usage in the *best* subsets. The higher the fraction the higher the relevance of the kernel in the current problem.

We should note that both *defect controller*, and *ADC enhancer* modules can only use the validation set VL during the evolutionary process, while the output of the *genetic engine* module is measured on the output of *ADC enhancer* module with an independent test set $TEST$. This way we assure that the output of our model estimates the classification rate that can be achieved in a real-time application.

In addition to the above two modules, we propose to apply a *fitness weight decay* process to the samples in the data sets. According to the images' creation time tag we weight the data sets, such that an old sample will have lower effect upon the overall classification rate. This way we can use the *fitness weight decay* process to attain better results for the current classified images, and we can also explore obsolete defects in the data sets. Unfortunately, our classifier doesn't generate time tags for the classified images. Therefore, we cannot implement *fitness weight decay*, and we leave it for future research.

The Experiment. We used the same data set as in the basic experiment in Section 5, with the same evolutionary parameters for both the *defect controller* and the *ADC optimizer* modules.

Results. Five separate runs were made for the enhanced model, showing almost identical behavior. Using the enhanced model we attained a classification rate of 0.901 after 15 generations, while with the basic model we obtained 0.897 after 50 generations. Both models attained significantly better results than the manual process, with half the complexity—a dramatic improvement in throughput terms.

As opposed to the *ADC enhancer* module, it is important to keep the *defect controller* module as objective as we can. Our goal in this module is to monitor the changing environment inside the fab, and not to achieve an optimized solution. For this reason we always look a fixed predefined number of generations backward, and we don't use the *hint* array during its own evolutionary process.

Both the reduced complexity, the automatic control process, and the higher accuracy attained, are practical benefits of our enhanced model. The major breakthrough of the enhanced model is its ability to independently fit itself to the changing environment inside the fab—with a deceptive environment of poor and inaccurate information—achieving a high classification rate, an increased throughput, and better generalization. Obsolete defects can be isolated using the *hint* array, and the input data set can dynamically change during the machine's lifetime.

By using the *hint* array we can not only identify obsolete and relevant samples in the data sets, we can also find an obsolete class of defects: when we find all the samples for a specific class of defects irrelevant we can eliminate it from the data sets.

7 Concluding Remarks and Future Research

Genetic algorithms are useful for numerous real-life classification challenges. When the search space is large, when the objective function is inaccurate and changes over time, when the data is impoverished, and when the problem we are trying to solve has subset-selection characteristics, a GA may be the answer. In our problem we find all the above elements. Finding an optimal subset out of a group of 498 elements is a hard problem with subset-selection characteristics. Some of our samples in the training set have been falsely classified so we have a deceptive objective function. When the defects we are trying to classify change during the fab life-time our objective function varies and we have to adapt to the new environment. In the experiments we performed, the genetic algorithm has proved itself as the best solution to date, as shown in Table 1.

Table 1. A comparison between the different optimization methods: 1) the manual method, where the CE optimizes the training set by hand, is the currently used method; 2) the heavy random stress test; 3) the basic evolutionary model; and 4) the enhanced evolutionary model. The best classification rate is shown in the first row, the number of kernels, which reflects the RBFN complexity and throughput, is shown in the second row, and the number of generations for the method to converge to an optimal solution is shown in the third row. We should remember that only the enhanced method can adapt itself to a changing environment.

	Manual	Random	Basic	Enhanced
Accuracy	0.870	0.883	0.897	0.910
Kernels	498	196	193	198
Convergence	Manually	50	47	15

In the future, when we can obtain a time stamp for each classified sample, we will be able to test the proposed *fitness weight decay* process. In addition, we hope that we can gather more data and test our new and obsolete defects exploration model.

References

1. David Sanchez, A.V.: Searching for a solution to the automatic RBF network design problem. Neurocomputing 42, 147–170 (2002)
2. Bilmes, J.: A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models (1997)
3. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
4. Buchtala, O., Klimek, M., Sick, B.: Evolutionary optimization of radial basis function classifiers for data mining applications. IEEE Transactions on Systems, Man and Cybernetics, Part B 35, 928–947 (2005)
5. Camacho, F., Manrique, D., Rodriguez-Paton, A.: Designing radial basis function networks with genetic algorithms. In: IASTED International conference artificial intelligence and soft computing, September 2004, vol. 451(8), pp. 398–403 (2004)
6. Kuncheva, L.I.: Initializing of an RBF network by a genetic algorithm. Neurocomputing 14, 273–288 (1997)
7. Kuo, L.E., Melsheimer, S.S.: Using genetic algorithms to estimate the optimum width parameter in radial basis function networks. In: American Control Conference, vol. 2, pp. 1368–1372 (July 1994)
8. Maillard, E.P., Gueriot, D.: RBF neural network, basis functions and genetic algorithm. In: Neural Networks International Conference, vol. 4, pp. 2187–2192 (June 1997)
9. Wai Mak, M., Wai Cho, K.: Genetic evolution of radial basis function centers for pattern classification, `citeseer.ist.psu.edu/8,1322.html`
10. Specht, D.F.: Probabilistic neural networks. Neural networks 3, 109–118 (1990)
11. Whitehead, B.A., Choate, T.D.: Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction. IEEE Transactions on Neural Networks 7, 869–880 (1996)