A System Model for Feedback Control and Analysis of Yield: A Multistep Process Model of Effective Gate Length, Poly Line Width, and IV Parameters

Edward A. Rietman, Stephen A. Whitlock, Milton Beachy, Andrew Roy, and Timothy L. Willingham

***Abstract*—We present a large system model capable of producing Pareto charts for several yield metrics, including effective channel length, poly line width, $I_{\rm on}$ and $I_{\rm sub}$. These Pareto charts enable us to target specific processes for improvement of the yield metric(s). Our neural network model has an accuracy of 80% and can be trained with a small data set to minimize the feedback time in the control loop for the yield. The system we describe has been implemented in a Lucent Technologies microelectronics lab in Orlando, FL.**

I. Introduction

THE MANUFACTURING of CMOS devices requires hundreds of processing steps. At the end of manufacturing, various yield metrics (e.g., transistor parameters and circuit tests) are determined by extensive electrical testing. It should be clear (and has been demonstrated [1]) that not all the processing steps have impact on each of the yield metrics. For example, interconnects between metal layers (called vias) require five processing steps for their manufacture. Along with the manufacturing of the vias, other structures known as via-chains are fabricated. These via-chains are used as a test structure to measure yield of the via manufacturing processes. In effect, the cluster of processes for via manufacturing could be viewed as a mini-fab that makes only vias, and the via chain resistance is the yield metric for this mini-fab.

Similar to the via mini-fab, in this paper, we focus our attention on a mini-fab devoted to manufacturing a transistor structure known as gates. The yield metric for our mini-fab is the effective channel length (and some other transistor parameters). Most of the process steps are in the early stages of the manufacturing and are not too scattered throughout the 300-step route during the manufacturing. Our basic conjecture is that only 22 manufacturing processes impact on gate fabrication and thus the effective gate length ($L_{\rm eff}$). This of course is reasonable, because we don't expect a metal deposition or a via etch process to have a direct impact on the transistor parameters. Supporting this conjecture with nonlinear mapping relations developed by learning machines is the primary focus of this project. With a learning machine model of these processes, we are able to conduct online Pareto analysis to find the processing steps that have significant impact on $L_{\rm eff}$ and we will be able to do this on a regular basis to understand the changes affecting $L_{\rm eff}$.

In this paper, after reviewing the literature, we discuss the database and statistical analysis of the data used for the study. Then we introduce some of the learning machine architectures and the difficulties of scaling them up to the size required for this project. The last part of the paper discusses the results and implementation issues.

II. Literature Survey

One of the main objectives of statistical process control is to provide feedback control with humans in the feedback loop. This *a posteriori* approach compares the end result of the process with the target specifications. By applying the rules of statistical process control (SPC), the engineers determine if a process is within specifications and adjusts the control parameters accordingly (cf. [2]). The feedback control effected is only as good as the statistical metrics and the engineering staffs' interpretation of these metrics. The major disadvantage of this approach is that corrective action is taken after processing. Thus, several batches may have already been processed by the time the corrective action has been taken. We can, at least, improve the decision-making capability of the engineers by providing them with machine learning programs that flag out-of-bounds conditions quickly to provide improved SPC metrics, thereby improving the feedback control. Moreover, with machine learning programs used for yield prediction, we can provide some level of feedforward control.

A few papers have appeared in the literature focusing on machine learning and SPC (cf. Beneke *et al.* [3], Guo and Doolen [4], Smith [5], Hwarng and Hubele [6], [7], and Baker *et al.* [8]). One of the best papers is by Hwarng [9]. It discusses a system of several parallel neural networks for detecting cyclic data in SPC control charts. Turner [10] has observed that there is a correlation between real-time monitored process parameters, such as current and voltage in a plasma reactor, and so-called "wafer-results" such as etch rate and etch uniformity.

Boskin *et al.* [11] have used a combination of regression models and physical device models to predict IC performance results prior to completion of manufacturing. Data from inline electrical test measurements and other inline manufacturing

Manuscript received March 8, 2000; revised August 14, 2000.

E. A. Rietman was with Bell Laboratories, Lucent Technologies, Orlando, FL 32819 USA. He is now with Starlab, 1180 Brussels, Belgium (e-mail: ear@starlab.net).

S. A. Whitlock, M. Beachy, A. Roy, and T. L. Willingham are with Cirent Semiconductor, Agere Systems, Orlando, FL 32819 USA (e-mail: swhitlock@lucent.com; alroy@lucent.com; mbeachy@lucent.com; tlwillingham@lucent.com).

Publisher Item Identifier S 0894-6507(01)01032-6.

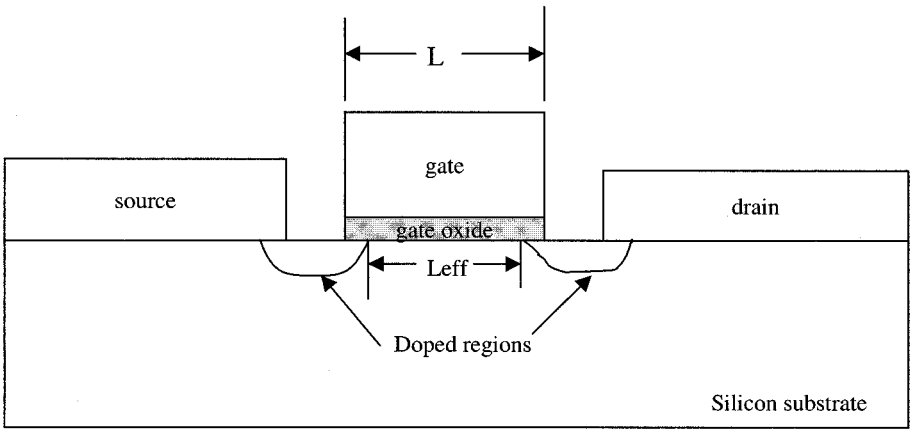


Fig. 1. Schematic of the cross section of a transistor—the length L is the printed length of the gate. The effective length is shorter because the doped regions diffuse under the gate.

data are used in linear regression models. The output of this is then fed into physical device/circuit models for electrical performance prediction.

Kim and May [12] present a very interesting feedforward approach to yield control. Their system model is designed for optimizing several yield metrics at the end of manufacturing vias for multichip modules. The system model consists of four submodules, each of which predicts a yield metric. The yield metric from the submodule is then fedforward into the next submodel to improve the prediction, and the output is used in a genetic algorithm to optimize the process recipe for each step. This approach could be utilized on many manufacturing processes. The problem with attempting to apply it to our task is that the number of processing steps between the first and the last is quite large. So, the mapping relation between the early steps and the final yield metric are very ill-posed maps, i.e., the same functional can be produced by many different function arguments, and the same argument can give rise to many different functionals. In short, this will look like noise and reduce any chance of success.

The work we report here does not use any physical device models. Furthermore, unlike Boskin *et al.* [11] or Kim and May [12], our models are not used in a predictive mode (i.e., feedforward), but rather in a feedback mode, and the output from our models is the computed transistor parameters. The results discussed here are exactly like those discussed in [1], except that it is a much larger system model and provides more detailed information in the Pareto charts.

It should be possible to use the Pareto charts to select specific subprocess modules to optimize for the yield metric of interest and to use that information to construct partial models, similar to the Kim and May model [12] for automatic feedback and feedforward control.

III. Effective Gate Length and the Mini-Fab Concept

A. Gate Length—Description

As the devices shrink on VLSI chips, the main dimension that summarizes the size is the gate width or channel length. Typically, a 0.5 μm technology has transistor gates printed at a width of 0.5 μm. This printed length or the physical gate length, L, is shown in Fig. 1. The region between the source and gate and between the drain and gate is doped with specific types of ions to enhance some of the characteristics of the transistors. These ions are "injected" by an implant device and then the ions are induced to diffuse by a thermal treatment known as annealing. Although great care is taken during the manufacturing to limit the diffusion of the ions under the gate structure, some spreading is inevitable. This has the effect of reducing the gate length. The effective length, L_{eff}, is measured by electrical tests of the speed of the transistor after manufacturing and is given by the relation

$$L_{\text{eff}} = L - 2\lambda$$

where λ is the lateral distance the source and drain ions extend under the gate and L is the "printed" length. The factors that influence effective gate length include product code (pattern density), operating voltage for the devices, and, of course, manufacturing processes such as implant, anneal, gate etch, lithographic development, etc. More details on L_{eff} can be found in Wolf [13].

From discussions with the engineers in the fab, and considering dozens of routing reports, we decided that there were 22 processing steps that had primary and secondary impact on L_{eff}. These processing steps comprise the key steps in the L_{eff} mini-fab. These 22 steps are the primary steps that define the gate structures on the integrated circuit.

B. L_{eff} Mini-Fab

The L_{eff} mini-fab is similar to the via mini-fab [1], but is much larger. In this case, we are concerned with the steps associated with manufacturing the gates for the transistors, and our L_{eff} mini-fab yield metrics are the effective linewidth, the poly linewidth, and the transistor parameters, I_{on} and I_{sub}. Table I lists the 22 processing steps comprising the L_{eff} mini-fab. Although these processes are scattered throughout the 300-step route during the manufacturing, our basic conjecture is that only these 22 manufacturing processes impact on gate manufacturing and thus the effective gate length (L_{eff}). Of course, in reality, interspersed between the 22 processes are other processes. We are simply isolating these as an abstract cluster of processes. If this conjecture is correct, then we can cluster these processes

TABLE I

step number	process
A05I005	implant P, for threshold voltage adjust
A06E052	hard mask etch, gate level
A06E176	poly gate etch
A07I007	implant P for lightly doped drain
A08I008	implant B, lightly doped drain
A09I006	implant As for source/drain
A10I008	implant B for source/drain
B05D008	deposit gate oxide
B07D010	anneal diffuse of lightly doped drain
B12R003	RTA reflow for window 1
B19D018	forming gas anneal
C05D009	diffuse P into poly
C05L002	deposit poly
C08E053	spacer etch
C10D015	flow BPTEOS dielectric 1 glass
C12R004	RTA metal 1 barrier anneal
D05L028	deposit hard mask for gate etch
DA1TL01	IV measurements, Leff, Ion, Isub, poly width
P05P010	apply photo resist for gate
P05P020	expose PR for gate
P05P030	develop PR for gate
P05P050	measure PR for gate
Y06Y001	measure line width after gate etch

as if they were an individual superprocess (mini-fab) with the wafers visiting these processes sequentially. The physics and engineering associated with these steps are described in Streetman [13] and Wolf [1414].

C. Objective and Goals for the Project

Our objectives are to find a regression "function" that will allow us to conduct sensitivity analysis. We would like to understand what are the driving factors for $L_{\rm eff}$, poly linewidth, $I_{\rm on}$ and $I_{\rm sub}$, and how these factors change from month to month, from product code to product code, from technology to technology, from lot to lot, and from tool to tool. With this information, we will have a better tool for engineers to control some of the most important yield metrics in IC manufacturing. In the following, we describe a system model we built and implemented in one of our fabs.

IV. Database Issues

In the following, we will refer to parm-measurements and IV measurements. The parm measurements are the result of parametric tests after individual processing steps. They are also called inline parametric tests and consist of measurements such as film thickness, sheet resistance, and line width. The IV measurements are current–voltage measurements at the end of manufacturing and are used to determine the electrical characteristics of the final chip. They are measurements on the transistors within the chip and represent the yield metrics for effective line width ($L_{\rm eff}$), the poly line width, and the transistor parameters $I_{\rm on}$ and $I_{\rm sub}$. (Hereafter, this set of four yield metrics will simply be called [$L_{\rm eff}$]. When specific elements of this set are needed, we will refer to them accordingly.)

We extracted approximately 800 megabytes, consisting of 34 ASCII flat files, from an SPC database. Each file contained one month of data on an old technology from our Orlando fab (0.5-μm technology). The data were from May 1995 through February 1998. Each line in a file contained the following: date (yymmddhhmm), lot number, processing-step number, technology, product code, equipment ID, operator ID-code (code numbers for the human operators), inline parametric (code names/numbers for the parametric tests) name, IV parameter name, IV parameter mean value, IV parameter standard deviation, and IV parameter sample size. Each lot of information consisted of 23 processes (22 processing steps and one "step" with yield data). There were 9 technologies (e.g., analog, BiCMOS-3V), 550 product codes (e.g., FPGA, DSP), 112 equipment IDs, over 1100 operator IDs, 55 inline parameter tests, and 23 IV tests. In the last two cases, the names for the tests are not standardized, and there may be several names for the same test or similar tests with different test structures.

We then obtained scalar numbers (RIM and NCHIP, described subsequently) from a lithography database. These two numbers are used in the computation of pattern density. Through extensive UNIX and C programming, all these files were joined and reorganized to result in a new ASCII flat file of 114 megabytes (about $8:1$ reduction in data base size). The categorical variables were converted to code numbers and finally into binary vectors (to be described subsequently).

The data were preprocessed so that all the data associated with one lot and one processing step was recorded in one record of an ASCII flat file. This new file contained 111 117 records and 5646 unique lots. ($5646^{*}23 = 129\,858$, therefore not all the lots were processed by all the steps.) As an example of one of the records, consider the following vector which is discussed and shown at the bottom of the next page.

The first four fields are a unix time stamp, the date/time, lot number, and the process code number (in this case, code 12 is C05-D009, diffuse phosphorous), respectively. The next 25 fields, a binary vector, represent the technology. In this case, the seventh element is set to $+1$. The remaining elements in the vector are set to -1. This indicates that the technology is 0.5 C-5 V. Although the data only have nine technologies the binary vector contains sixteen "empty" positions for future technologies and makes the model easily expandable.

The next two elements in the data vector are RIM and NCHIP, respectively. RIM is the amount of unpatterned silicon along the edges of the wafer, and NCHIP is the number of chips on the wafer. These two numbers are used as indicators of pattern density (actually weak indicators). The actual poly-density was not available from the lithography database. Our rational for using this is that pattern density is a measure of the amount of patterned material (photoresist or hardmask) on the wafer during the gate etch. High pattern density means a larger number of poly runners will be produced. Conversely, low pattern density means more of the poly is etched away, and less is left on the wafer after the etch. It is clearly dependent on the technology and product code. Some ASIC's will have a large number of tightly packed transistors and more open area to etch. Others will have a smaller number of transistors less tightly packed and therefore have a smaller open area. Obviously if we have a huge number of very small chips on the wafer we will have a greater open area to etch than if we have a small number of huge chips.

The number of chips per wafer is therefore an indicator (albeit, weak) of pattern density.

The next binary vector represents the tool code. The vector is long enough for possibly 25 different tools per process. In the case shown above, only the fourth element is $+1$, indicating the presence of tool ID# 726F2 all the other elements are -1.

The next 25-element binary vector represents the inline parameters measured at this process step. In this case, they map to the following inline measurements (see parm table in the Appendix):

21002:POLYTHICKNESS:STDRANGE
31001:MEANSHEETRESISTANCE:STDAVERAGE
31002:MEANSHEETRESISTANCE:STDRANGE
31003:MEANSHEETRESISTANCE:STDRANGE
41001:MEANSHEETRESISTANCE:STDRANGE
51002:STANDARDDEVIATION:STDRANGE.

The next 25-element vector contains scalar numbers representing the values for the above inline measurements (this vector is called the inline parm code vector). The data were normalized for unity standard deviation and zero mean. Elements not present are set to the normalized mean value.

The next 25-element binary vector represents the IV parameters (called the IVcode vector). For this particular example, they are: Le_N0.5X15_B, LE_P_0.6X15_B, N_0.5X15_H_Isub, N_0.5X15_V_Ion, PY1_Width_0.6. The first two are L_{eff} measurements. The next three are I_{sub}, I_{on} and poly line width, respectively. They are simply code numbers used in our database.

The next 50 elements are grouped into two 25-element scalar vectors. The first is the mean values, and the second is the standard deviations of the measured IV parameters.

All these elements are the relevant data for one lot at one process step. The data were partitioned into inputs and outputs and normalized on-the-fly while processing by the learning machines. The normalization was for zero mean and unity standard deviation.

V. Statistical Analysis: Preliminary Data Analysis

We conducted cross correlation, autocorrelation, and probability distribution studies for all the variables. In this section, we will summarize the highlights. The full set of statistical charts and graphs consists of two hundred pages and, of course, is beyond the scope of this paper.

Several interesting cross correlations were observed among the inline parameters. All the following correlations were

---

12716713 199503050528 9176040000 12
−1 −1 −1 −1 −1 −1 1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
9.780000 358.000000
−1 −1 −1 1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1 −1
−1 −1 1 1 −1 −1 1 −1 −1 −1 1 −1 1 −1 −1 −1 1 −1 −1 −1 −1 −1 −1 −1 −1
0.000000 0.000000 0.000000 38.889999 0.000000
0.000000 1.060000 0.000000 0.000000 0.000000
2.183333 0.000000 0.640000 0.000000 0.000000
0.000000 2743.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
−1 −1 −1 1 −1 −1 −1 −1 −1 −1 1 −1 −1 −1 −1 −1 −1 −1 1 1 −1 −1 1 −1 −1
0.000000 0.000000 0.000000 0.527000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.577000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 103.820000 5.626000
0.000000 0.000000 0.518000 0.000000 0.000000
0.000000 0.000000 0.000000 0.053700 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.011500 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 39.297600 0.529300
0.000000 0.000000 0.008000 0.000000 0.000000

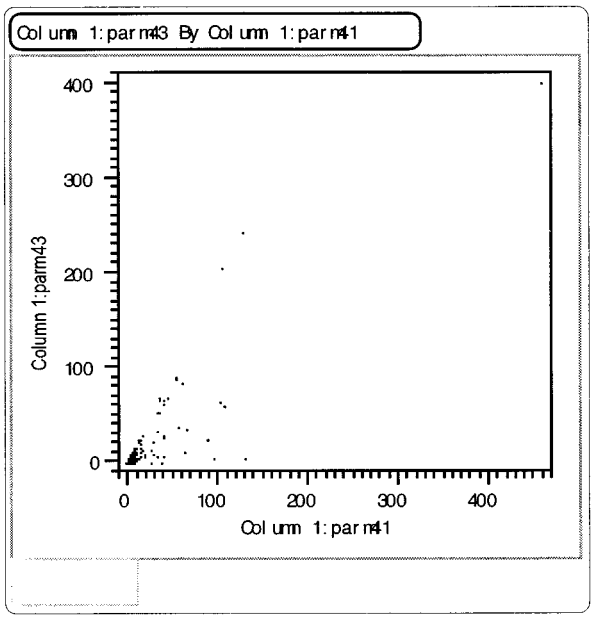


Fig. 2. Cross-correlation plot of parm43 and parm41 (see the Appendix).

greater than 0.3 in magnitude (i.e., $+/-$) (see parm table in the Appendix).

parm13-parm4: 0.32
parm31-parm5: −0.37
parm33-parm8: 0.31
parm34-parm8: 0.37
parm33-parm10: 0.38
parm47-parm12: −0.35
parm33-parm24: 0.30
parm29-parm25: 0.53
parm34-parm25: 0.33
parm33-parm28: 0.47
parm34-parm28: 0.37
parm38-parm35: −0.52
parm38-parm37: 0.53
parm39-parm37: 0.33
parm41-parm37: 0.33
parm42-parm37: 0.34
parm39-parm38: 0.67
parm42-parm38: 0.46
parm42-parm39: 0.43
parm43-parm41: 0.98

Statistically, these are significant, but in reality these cross correlations may have little or no predictive capability. For example, parm43-parm41, with a correlation of 0.98, is plotted in Fig. 2. The data are simply clustered in a small region of 2-space.

There were also a few interesting cross correlations among the IV parameters. These include IVcode20-IVcode4: −0.42, IVcode21-IVcode4: −0.34 and IVcode21-IVcode20: 0.37 (see Table II).

Cross correlations between the inputs and the outputs were also examined. There were no linear cross correlations larger than about 0.25. The largest cross correlation was between IVcode18 and parm41 (N_0.5X15_H_I_{sub} and particle_delta_std_avg). As shown in Fig. 3, there is little

TABLE II

item number	short name	description
1	iv2	Le_NHC.6X15__B
2	iv3	Le_N_0.5X15__B
3	iv4	Le_N_0.6X15__B
4	iv7	Le_N___.5X15__B
5	iv9	Le_PHC.6X15__B
6	iv10	Le_P_0.6X15__B
7	iv18	N_.5X15_H_Isub
8	iv19	N_.5X15_V_Ion
9	iv20	N_.6X15_H_Isub
10	iv21	N_.6X15_V_Ion
11	iv22	PY1_Width_0.6

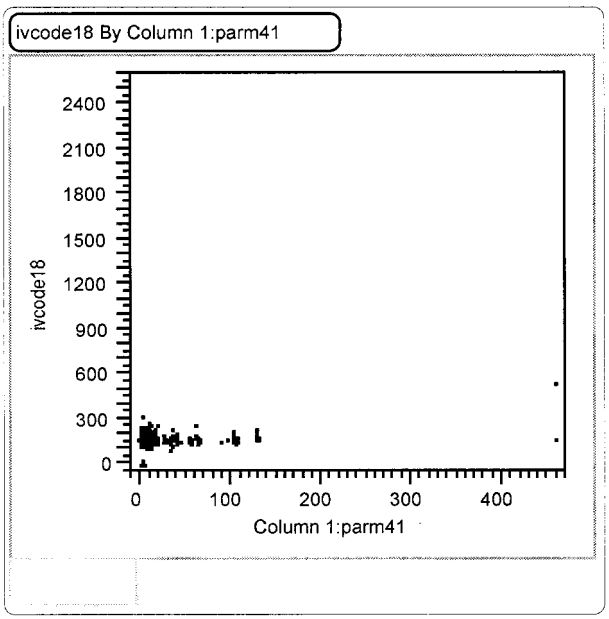


Fig. 3. Cross-correlation plot of parm41 and IVcode18 (see the Appendix and Table I).

predictive capability from this information. All this implies is that the desired mapping relations are nonlinear. A similar lack of linear cross correlations was observed in the via mini-fab model discussed by Rietman *et al.* [1].

Turning now to the probability distributions, we will first examine the highlights of the IVsigma data. That is, the distributions of the standard deviations of the IV measurements. There are a few outstanding data points, but for the most part the distributions are quite tight. The IVsigma22 is the most interesting with a mean value of 5.50 and a standard deviation of 63.65. Of course, a few samples can cause the standard deviation to "blow up." In addition, keep in mind that these numbers represent the moments for the distribution of the standard deviation of the IVsigma data.

There are no outstanding features in the distributions of the IVmeans. All the distributions are reasonably tight. The moments for these distributions, as well as the moments for all the other distributions, were coded in the learning machine software for normalizing the inputs and outputs prior to any learning iterations.

For the inline parametric measurements, the distributions are, for the most part, single-mode Gaussian or log-normal in structure. A few of them are multimodal. The following are

prominently bimodal: parm2, parm7, parm12, parm14, parm21, parm32 (see the Appendix for index of code).

In examining the autocorrelations for the IVparms (IVmeans), we find that there are no significant autocorrelations in the variables. The same cannot be said about autocorrelations on the inline parameters. Parm20 had a strong autocorrelation out to a time lag of 50. This autocorrelation suggests one could model the data with an ARMA model [15]. Based on the time-lag, we believe these autocorrelations are caused by operator shift changes and measurement style differences among the operators. In the newer generation of technology, these problems have been eliminated and there are no autocorrelations.

In summary, the inputs and outputs are vectors of analog and binary information. The elements of one input vector are not correlated with similar elements from another input vector. A similar observation has been made concerning the output vectors. The cross correlations that exist between vector elements are weak. Therefore, we can consider the input–output vectors as independent identically distributed (IID) random pairs. Some theoretical results show that even slightly dependent data pairs behave roughly the same as independent identically distributed data [16]. Furthermore, because of the near IID nature of the data, our machine learning models can have single or multiple outputs. (If the data were cross correlated we would need multiple output models.) The few autocorrelations suggest that these variables may be best entered as time-delay or recurrent connections to the input. As we show in the next section, only multiple outputs would be efficient for this project, and the input dimension may be large.

VI. Systems and Learning Machine Issues

In this section, we will briefly discuss a systems' view of the L_{eff} mini-fab and discuss the theoretical foundation for the learning machines investigated.

A. Systems View of L_{eff} Mini-Fab

The mini-fab is an abstract idea allowing one to cluster processes that are associated with one or a few yield metrics. For any given yield metric (e.g., L_{eff}), we ask which processing steps are the most important for determining this yield metric. These steps become the relevant mini-fab. From a systems perspective, wafers needing transistors are input to the mini-fab, and wafers with transistors are the output from the mini-fab. The processes listed in Table I are the subsystems in the mini-fab. After some of these processes, there is an inspection or measurement. These measurements are yield metrics pointing to the quality of the individual process (or the subsystems in the mini-fab). The collection of these subsystem metrics can be mapped to the entire systems' metrics. That is, the inline parametrics, or "parms," are mapped to the yield metric for the mini-fab—in this case, L_{eff}, poly line width, I_{on}, and I_{sub}.

Disregarding for the moment the architecture of the learning machine for the task, we want to build a regression machine to effect the appropriate system mappings. Consider that we have 22 processing steps. For most of the steps, we have an inline measurement parameter indicating how the tool used in that step performed. At one level, these measurements are the outputs from the processes. From another viewpoint, they are the inputs to the L_{eff} model. This suggests two approaches to modeling the L_{eff} mini-fab.

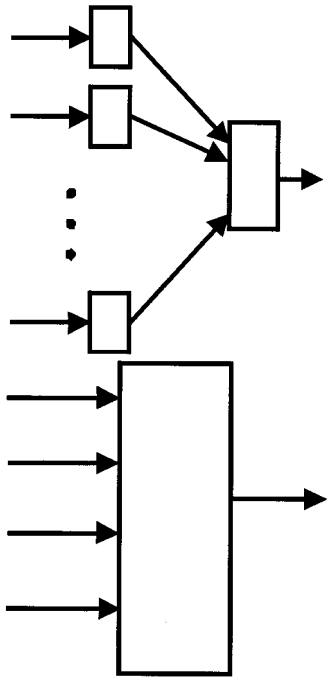


Fig. 4. Block diagram for two possible models of the L_{eff} mini-fab.

Fig. 4 shows a block diagram for two possible system models to address the L_{eff} problem. Model **A** is an hierarchical structure and Model **B** is not. Model **A** consists of individual process step models whose outputs feed into a model to compute the L_{eff}. The inputs to the process models may be time-delay windows of inline parameter measurements, tool data, product code information, NCHIP, RIM, time/date, and technology (i.e., voltage level). The outputs of the process models are inline parameter values one or more time units into the future. If time-delay is used, the models are predictors. If no time-delay is used, the models are regression estimators. We want to determine the effects of tools and product codes on the inline parms and the effects of tools, technologies, and inline parms on the [L_{eff}]. This system model has the disadvantage of passing errors from the subsystem models to the higher level model, but has the advantage of small numbers of inputs per subsystem.

In contrast, model **B**'s inputs are the inline measurements from each process, the tool data, product code information, time/date, and technology (i.e., voltage level and product code). In this case, the inline data, entered into the model, will be for all the processes of a given lot. The output would be the [L_{eff}] computation for that lot of wafers. Thus, the model would work on a lot-by-lot basis. The disadvantage of this model is the huge number of inputs at the input level. Its advantage is that errors are not compounded.

Mathematically these models could be expressed as follows. Model **A** is given by the system of equations

$$\left.\begin{array}{l} Z_1\colon X_1 \to Y_1 \\ Z_2\colon X_2 \to Y_2 \\ \quad\bullet \\ \quad\bullet \\ \quad\bullet \\ Z_{22}\colon X_{22} \to Y_{22} \end{array}\right\} \xrightarrow{\Omega} L$$

where Z_i indicates the process model, and X_i and Y_i are the full input and output vectors, respectively, for that process model. The Z maps are then combined with the Ω map to calculate the $[L_{\text{eff}}]$. Each of the Z maps does a $\Re^m \rightarrow \Re^n$ mapping, and m and n are on the order of ($m \sim 181$, $n \sim 25$). The Ωmap combines all the Y_i models so it does a $\Re^p \rightarrow \Re^q$and p and q are on the order of ($p \sim 578$, $q \sim 11$). These values are calculated as follows. There are 22 processing steps each with 25 scalars of data ($22^*25 = 550$). In addition, there are three scalars for the time, RIM, and NCHIP, and there are 25 binaries representing the product code or technology. The total input dimensionality is 578. The 11 outputs for q are given in Table II. It is important to realize that these 11 outputs, e.g., L_{eff}-N, L_{eff}-P, etc., are all members of the four classes of output types that fall into the $[L_{\text{eff}}]$ set.

Model **A** can be expressed more compactly as

$$A\text{: } X \xrightarrow{Z} Y \xrightarrow{\Omega} L.$$

Model **B** can be expressed more directly by

$$B\text{: } \chi \rightarrow L$$

but, in reality, there are more difficulties involved in finding this mapping.

The primary difficulty of this model is that it does a $\Re^{3982} \rightarrow \Re^q$ mapping. The dimensionality of 3982 comes about as a worst-case estimate. We have 181 fields per record and we would need 22 processing steps of data ($181^*22 = 3982$). Although this could be done with many types of learning machines, in practice it could be difficult with a small number of samples (we have 5646 samples). Notice, however, that many of the elements are binary elements representing active scalar inputs. For example, the inline parm code vector, discussed in Section IV, indicates which of the inline parameters are/are not in the inline data vector. Keep in mind that each inline data vector consists of 25 scalar elements, but all of them are not used for any given lot. In fact, only a few of them are used for any lot and most of the elements are simply <empty> for excess capacity in the model. The output vector from the system model is also padded with excess capacity. There are only a little over one hundred inputs active for training the learning machine. The rest of the inputs are excess capacity.

Let's examine a simple example to better understand the complexity of the models and how we can circumvent the curse of dimensionality (cf. Vapnik [17], Geman *et al.* [18], and Hassoun [19]). Assume we have an input vector of five scalar elements and an output vector of one element (i.e., a single scalar output). Associated with this five-element scalar vector is a five-element binary vector. Fig. 5 shows two neural network architectures. The filled nodes represent binary elements (i.e., -1 or $+1$), and the open nodes represent scalar elements. For this problem, the binary vector acts as signals to indicate which elements in the scalar vector the neural network is to "listen to" during the training. This approach is common in training neural networks when we want to associate specific binary information with specific scalar information (cf. Masters [20]). The input dimensionality on network **A** is 10. An alternative approach is shown in Fig. 5(b). Here we use the binary vector as a mask, or a gating

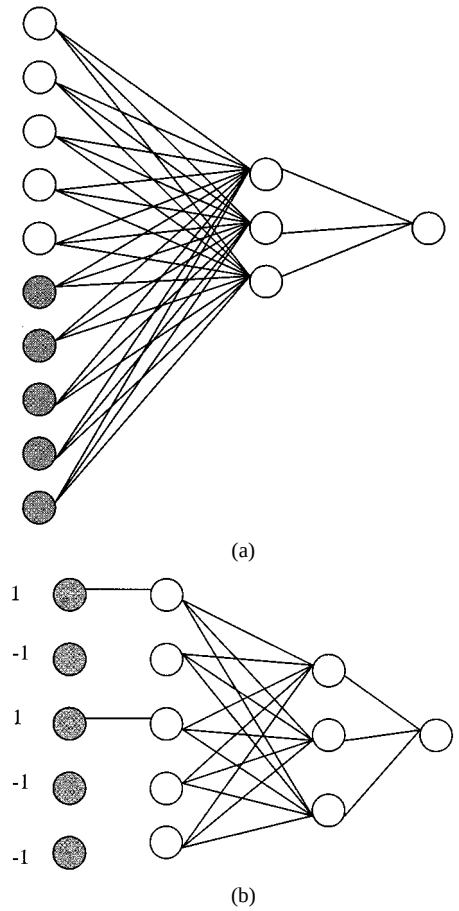


Fig. 5. Two neural network architectures described as examples.

network, to indicate which inputs in the scalar to feedforward. The others are simply not fedforward. The input dimensionality for network **B** is five. For the actual gating inputs shown in Fig. 5(b) the input dimensionality is two. Using this approach, we can have a huge reduction in the actual dimensionality for training.

None of our input scalar vectors (inline parameters) for any of the processing steps are filled with the full set of 25 elements. Never are they all full at the same time for a given lot. After we select a reasonable architecture offline, it is installed online and trained to work with current technology and processes. The excess capacity in the learning machine system model will allow it to adapt to new generations of technology, new processing tools being brought on-line, increased manufacturing capacity, new inline measurements, and new end-of-processing measurements.

To reiterate, the inputs are viewed in clusters. We have 22 sets of 25 scalar elements representing the inline measurements and the associated binary vector for gating the inputs. This gives a total of 550 inputs. In addition, to train the neural network we have a binary vector representing which technology/product code is represented by each lot of wafers. This is a 25 element binary vector (again including excess capacity) used in the input during training. In addition, we have a Unix time stamp normalized to a small number by dividing by a large number representing the year 3000 (a Y3K bug ?), and we have the scalar numbers RIM and NCHIP. This gives a total of 578 inputs (a

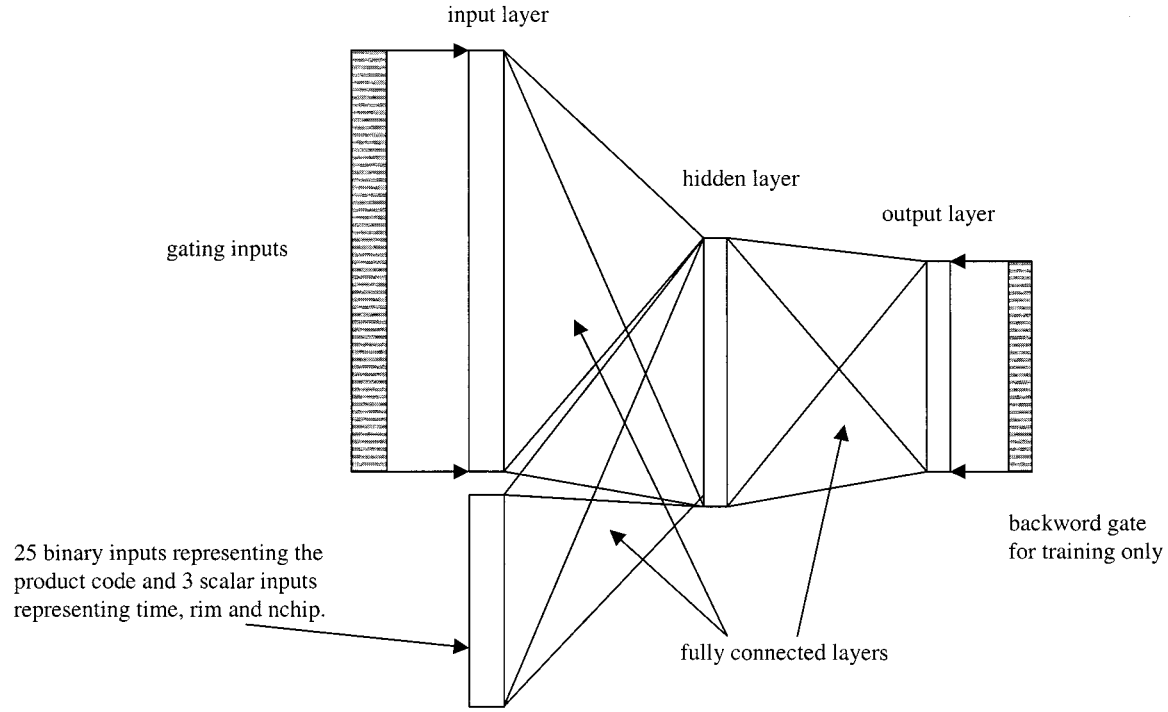


Fig. 6. The final L_{eff} mini-fab neural network architecture.

huge reduction from the 3982 calculated above). The Unix timestamp is the date for the start of the lot in the mini-fab, i.e., the time the lot was processed by the first step in the mini-fab. Of course, not all of these 578 inputs are active at any given time. Most of them are always inactive. As in the small example in Fig. 5(b) where the gating network reduces the input dimensionality to two, here the gating network reduces the dimensionality to 131, which leaves 447 inputs as excess capacity in the network.

Just as there is a gating network to determine which inputs will be fed forward, there is a gating network to determine which outputs are fed back during the backpropagation phase of training the network. Schematically the entire L_{eff} model is shown in Fig. 6.

B. Neural Network Learning Theory

The output of a neural network, r, is given by

$$r_k = \sum_j \left[W_{jk} \bullet \tanh\left(\sum_i W_{ij} \bullet x_i \right) \right]. \qquad (1)$$

This equation states that the ith element of the input vector x is multiplied by the connection weights W_{ij}. This product is then the argument for a hyperbolic tangent function, which results in another vector. This resulting vector is multiplied by another set of connection weights W_{jk}. The subscript i spans the input space. The subscript j spans the space of hidden nodes, and the subscript k spans the output space. The connection weights are elements of matrices and are found by gradient search of the error space with respect to the matrix elements. The cost function for the minimization of the output response error is given by

$$C = \left[\sum_j (t-r)^2 \right]^{1/2} + \gamma \|W\|^2 . \qquad (2)$$

The first term represents the rms error between the target t and the response r. The second term is a constraint that minimizes the magnitude of the connection weights W. If γ (called the regularization coefficient) is large, it will force the weights to take on small magnitude values. This can cause the output response to have a low variance and the model to take on a linear behavior. With this weight constraint, the cost function will try to minimize the error and force this error to the best optimal between all the training examples. The effect is to strongly bias the network. The coefficient γ thus acts as an adjustable parameter for the desired degree of the nonlinearity in the model. Details of neural networks can be found in Rumelhart *et al.* [21] and Hassoun [19], and details of learning with constraints can be found in Vapnik [22].

As described above, the neural network has 578 inputs and 25 outputs. Of these, only 131 are active inputs and 11 are active outputs. Both the input layer and the hidden layer also include a bias node set at constant 1.0. This allows the nodes to adjust the intersection of the sigmoids. The excess capacity in the learning machine allows for new product codes, tool sets, etc. The network has one "hidden layer" with 20 hyperbolic tangent nodes, and so there are a total of 2860 adjustable connections ($132^*20 + 20^*11$). (This calculation included the bias nodes in each layer.) We have almost exactly twice as many samples (5646) as we have adjustable parameters.

Though the number of connections, or adjustable parameters, in a learning machine is not the critical element determining

the generalization ability of the machine, it is certainly a critical element. Another critical element is the VC-dimension to which we refer the reader to the literature (cf. Vapnik [17], [23], Abu-Mosstafa [24], Baum and Haussler [25], Guyon *et al.* [26], Vapnik *et al.* [27], Holden and Niranjan [28], and Haussler *et al.* [29]).

The question of training neural networks with small data sets is significant. Huang *et al.* [30] discuss a neural network plasma etch model with 88 connections and 17 samples. They used the "leave-one-out" cross-validation procedure (cf. Cohen [31]) to develop the model. Kim and May [32] discuss a D-optimal experiment to design the network architecture of a plasma etch model, and Kim and May [33] use a modification of the backpropagation algorithm in which the cost function allows the network to slowly degrade or forget information that is no longer needed. This is essentially the same as weight regularization or weight minimization to reduce the network complexity. Newer methods of automatic model selection have essentially eliminated empirical design and selection of neural network and learning machine architecture (cf. Vapnik [17], Devroy *et al.* [16], and van der Vaart and Wellner [34]).

For our model, cross validation consisted of two steps. First, we cross validated by using randomly selected data from the database that was not used in training. Our second model validation consisted of studying the Pareto charts for data sets from which we expected specific results or trends. This validation will be discussed in detail in Section VII.

C. Sensitivity Analysis to Investigate the Driving Factors of L_{eff}

Once the model is built, we would like to understand its behavior. There are two approaches to investigate the response of the model. One of these is sensitivity analysis leading to the construction of a Pareto chart or bar chart, the other is response curves (surfaces). Neither method is very reliable for nonlinear systems. However, the methods can supply some insight into the process.

The sensitivity of the output with respect to the inputs is found from the partial derivative of the particular input of interest while holding the other inputs constant. The observed output is then recorded. By repeating this for all the inputs, it is possible to assemble response curves. The procedure has been described by Klimasauskas [35] and by Deif [36]. For response curves, the actual procedure consists of using a mean vector of the inputs and making small, incremental changes on the input of interest while recording the output. The first input, for example, is selected and a small value is added to it. All the other inputs are at their mean value, which should be very close to zero for normalized inputs. The vector is then fedforward to compute the output of the learning machine or system model. Further small values are added and the outputs are collected in a file for graphics. The final results can be represented as a curve of the change in the input value versus the network output.

The importance of the inputs can be ranked and presented in a bar chart known as a Pareto chart. Usually, the number of bars in the chart is equal to the number of inputs. Each bar represents the average sensitivity of that input. The procedure to construct this chart consists of using the real database vectors, adding a small quantity to one of the inputs and observing the output. Using this procedure, a matrix of the derivative of the response with respect to the input is created for the elements of each input vector. Each row in the database will give one row in the matrix. The number of columns will equal the number of inputs to the network or learning machine process model, and the elements in the matrix will be the derivative of the output with respect to the derivative of the input. The columns of the matrix are then averaged. The derivatives are signed so the absolute value is taken for each element in the vector. The resulting vector is used to construct the bar chart.

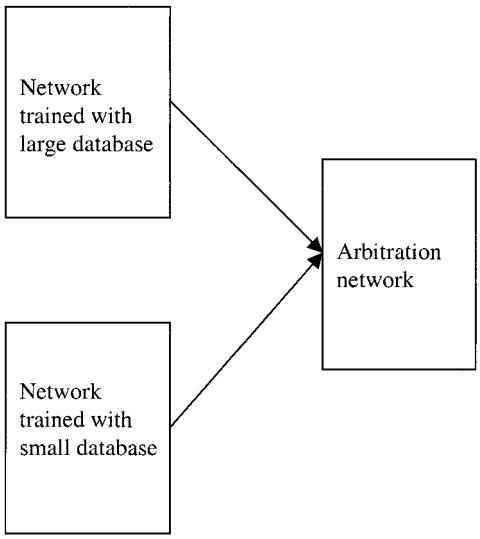


Fig. 7. Old and new neural networks connected in an arbitration network for sensitivity analysis. The old network is trained with a huge data set and the new network is trained with a small recent data set. The arbitration network is then trained with the new data set and incorporates the results from the more mature network.

Once the model is constructed the above procedures result in good, though first-order, sensitivity analysis. In the case of $[L_{\text{eff}}]$, the driving factors can be determined from a frozen model. In reality, the driving factors for $[L_{\text{eff}}]$ will vary from week to week. We know that the product code may be the primary factor one week, and gate etch the next. We would like a procedure similar to that used by the yield analysis engineers. They can draw on a vast knowledge base from years of experience. That knowledge is embedded in *their* biological neural network. They can examine a small set of new data from a recently manufactured product and deduce a likely factor causing the observed yield.

If we conduct sensitivity analysis on a frozen model of a process, the results will always be the same. Because the mathematical operations of neural networks are vector-matrix multiplications and if the matrix does not change (i.e., frozen model) and the same vectors are used in the multiplication, than the same results can be expected. Ideally, we would like to train a new model of the process on the small sample set sequenced in time (e.g., 100 batches of wafers) and use that for sensitivity analysis. The approach we take is similar to that used by the yield analysis engineers. We readapt the old network to the new small data set while using statistical regularization. We also train

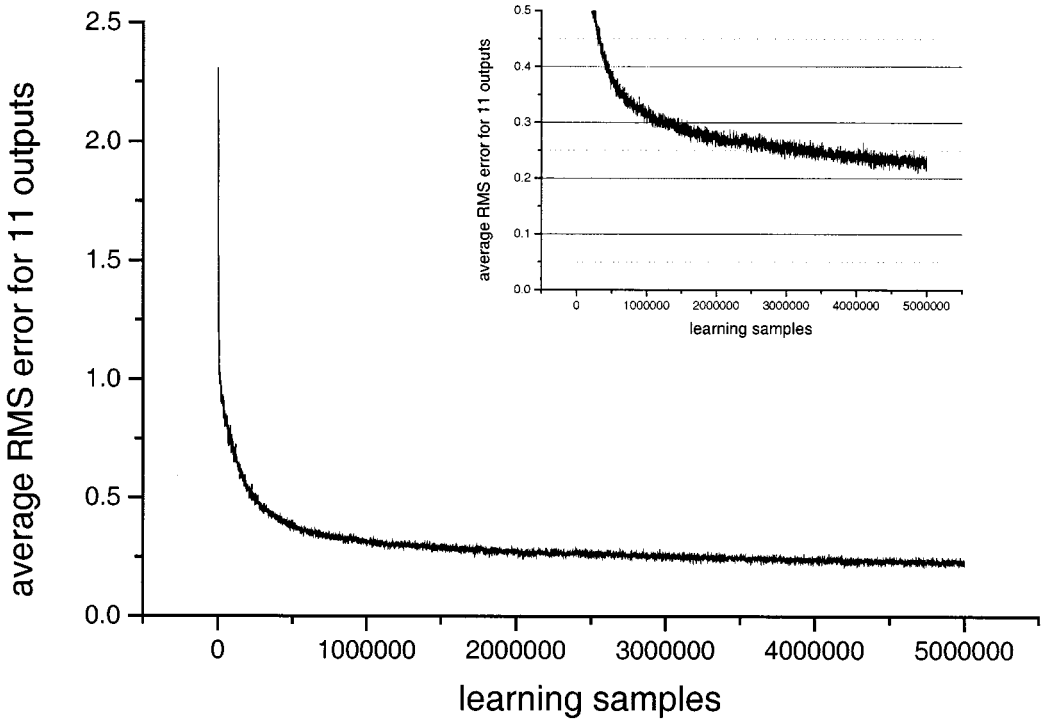


Fig. 8. Example of learning curve for $L_{\rm eff}$ neural network mini-fab.

a new network (or learning machine), of the same architecture, with the new data set. After these two networks are trained on the new data, their outputs are fed into an arbitration network (see Fig. 7). The final combined network model is used in the sensitivity analysis. Basically, what this amounts to is conducting sensitivity on the two networks and comparing the Pareto charts.

VII. Results and Discussion

Our (132-20-11) network was trained according to (2) with 5000 lots of wafers. Fig. 8 is an example of the overall learning curve. Keeping in mind that there are 11 active outputs, the curve represents the average of the rms error for each of these 11 outputs. That is, we compute the rms error for each output and then find the average of those. The curve shows that after 5 million learning iterations, the neural network was still learning, but that the average rms error is about 0.20. This implies that the model is about 80% accurate for the combined outputs. The accuracy of the model for the individual outputs is shown in the bar chart of Fig. 9, and the individual outputs are shown in Table II. We see, for example, that the model for the 0.6-μm technology, N-Channel $L_{\rm eff}$, is about 90% accurate. The other outputs are interpreted similarly.

We performed a validation on the model by first training with about 2/3 of the data and using the other 1/3 for validation. This 1/3 sample was selected at random from the data file. Once we saw that the validation error was at about the same level as the training error, we quickly went on to explore the sensitivity and Pareto. If the training error is acceptable and if the network was trained with a large number of samples, then our philosophy is to simply move on to the next stage testing in the real world. In short, we let the real world be our validation. Since the model

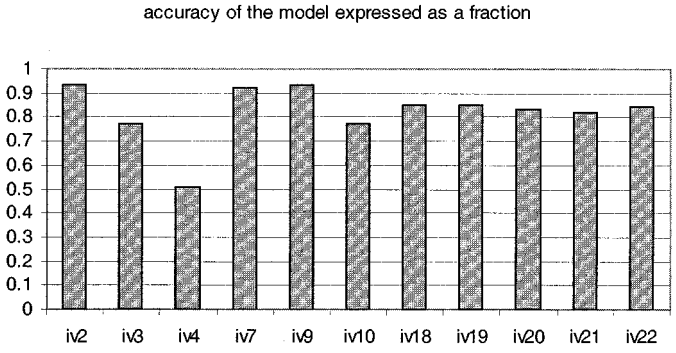


Fig. 9. Bar chart showing accuracy of the model.

is to be used for sensitivity analysis, we will use our intuition of the process(es) and realism of the Pareto for known sets of data as our validation. Of course, intuition and realism are difficult to quantitate, so we conducted several experiments in training with different sizes of data sets and biased data sets. But, in order to understand these results we first need to describe the outputs presented in Table II. Items 1 and 3 are 0.6-μm technology, N-channel $L_{\rm eff}$. Items 2 and 4 are 0.5-μm technology, N-channel $L_{\rm eff}$. Items 5 and 6 are 0.6-μm technology, P-channel $L_{\rm eff}$. Items 7, 8, 9, and 10 are transistor parameters, $I_{\rm sub}$ and $I_{\rm on}$ for 0.5-μm and 0.6-μm technology. Item 11 is the poly line width after manufacturing. In addition, there are two different test structures for items 1 and 3 and two different test structures for items 2 and 4.

Once we have a good model, we can conduct sensitivity analysis to determine the impact of each of the inputs on each of the outputs. Table I is a list of code names and a description of the individual processing steps. (They are not in processing sequence, but rather in alphabetic order by code name.)

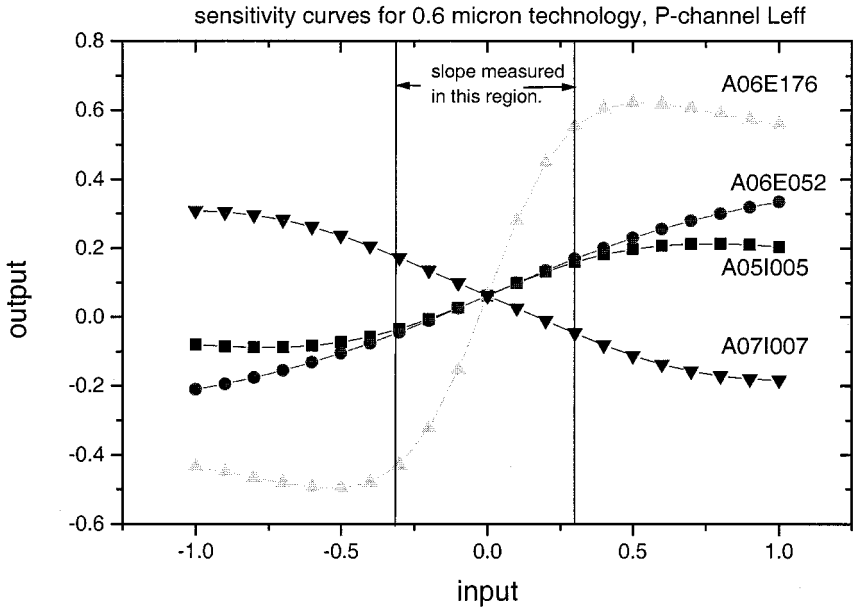


Fig. 10. Sensitivity response curves.

Sensitivity analysis is conducted as follows. Since for each of the individual processing steps, there are several inputs (e.g., mean value and standard deviation for film thickness and mean value and standard deviation of sheet resistance), and since these are clearly coupled to each other, it makes sense to "tweak" them in parallel. (as will be described below, this approach presented the most "believable" results). The "tweakings" were done by incrementally changing the relevant set of inputs by 0.1 starting at -1.0 and ending at 1.0. Thus, there were a total of 21 sets of incremental inputs. All the other inputs to the neural network were set to the mean value of 0.0 (recall the data are normalized for zero mean and unity standard deviation). After each feedforward, we observe the output and plot curves similar to those shown in Fig. 10. Here we see four sensitivity curves. The entire range of input results in sigmoid curves for the output. For example, The curve labeled "A06E176" shows the sensitivity of the poly gate etch process on the P-channel electrical line width ($L_{\rm eff}$) for a 0.6-μm technology. The other curves can be decoded with the use of Table I. When the curve has a positive slope, this indicates a positive correlation. For the "A06E176" curve (the gate etch), this indicates that more oxide remaining gives higher $L_{\rm eff}$. The other curves can be interpreted similarly.

For each process step listed in Table I, we could generate a whole set of sensitivity curves. By measuring the slope (in the indicated region of Fig. 10) we can produce the Pareto chart of the individual processing steps (Table I). Typically, one then takes the absolute value of the slope in preparing the Pareto chart. The direction of correlation is lost by this step, since typically one wants to find which processing step has greatest impact. The entire set of eleven Pareto charts is shown in the Appendix. We conducted three experiments with the learning machine model of the system. In one experiment, we trained the network with 5000 lots and then conducted the sensitivity. The Pareto charts for this experiment are in the Appendix and have the label "not biased, big data set."

In another experiment, we trained the learning machine on a small data set of 150 lots that were selected as a sequence in time. With this small data set, the actual training was done by selecting samples from the 150-lot set at random and feeding it forward into the neural network for training. The validity of training with a small data set was discussed above in the section on neural network theory. After training, a sensitivity analysis was done to compute the Pareto chart. The results are shown in the Appendix. The charts for this experiment are labeled "little data set."

The objective of the experiment in training with a small data set is to observe the [$L_{\rm eff}$] Pareto for a small group of lots. This would be equivalent to the yield analysis engineer sitting down with a week of data and figuring out which processing steps are having most impact on [$L_{\rm eff}$] that week. With our automated software program, this will give almost dynamic Paretos for the [$L_{\rm eff}$] set.

The third experiment consisted in biasing the training of 5000 lots. We selected all step data for spacer etch (C08E053) to be either -1 or $+1$ on the input. The deciding point was based on the target value for output number IV20 ($I_{\rm sub}$, for 0.5-μm technology). If the target value from the database was less than zero (the mean value), the inputs for C08E053 were set to -1. If the target was >0, the inputs were set to $+1$. This means that the film thickness after the spacer etch would be either too thin or too thick. So, if we look at the Pareto charts, the height of the bar for spacer etch should change significantly for the 0.5-$I_{\rm sub}$ chart, and because of interactions we should also see significant changes in the spacer etch for the 0.5-$I_{\rm on}$ chart. We would expect the spacer etch bar to increase in the biased experiment, and this is what was observed. Usually, actual directions of changes are difficult to predict because of the strong nonlinearity in the neural network model. The Pareto charts can only provide information as to which processing steps are most important. Subtle changes from one chart to the another are not interpretable.

VIII. Conclusion

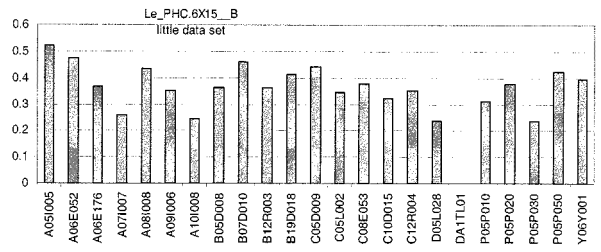
We have discussed an implementation in our fab of a large system model capable of generating Pareto charts that suggest which processing step is having most impact on four yield metrics: effective line width (L_{eff}), poly line width, I_{on} and I_{sub}. In addition, our system is capable of indicating which product code (e.g., 0.5AC, 0.5BCI-3V) is having most impact on this set of yield metrics. Due to space considerations, we did not show the 11 Pareto charts of product code and tool ID. The full set of Pareto charts act as suggestions for yield enhancement and feedback yield control strategies. The current large system model can also be used in a predictive mode to enable us to abort processing. This aspect of the model was not discussed in this paper, but is similar to that discussed in Rietman *et al.* [1].

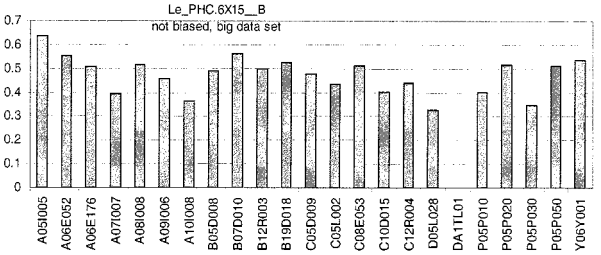
Our model is accurate to 80%, and it is capable of being trained on only a few days of lot data. This training on a small number of lots is significant, because it allows us to shorten the time involved in feedback control of these yield metrics. The next logical development would be to collect statistics on which processing steps are the problem areas and to target them for advanced process control.

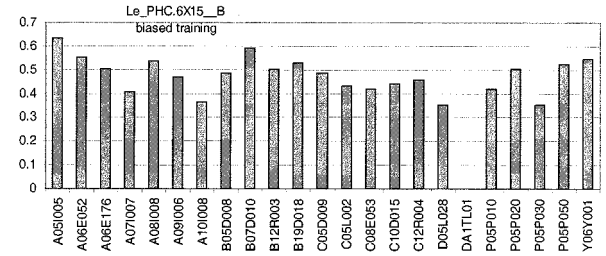
Appendix

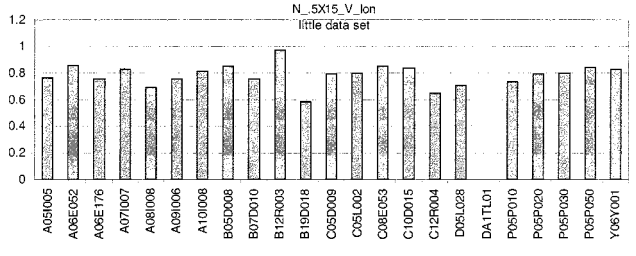
Parm Code Name Map

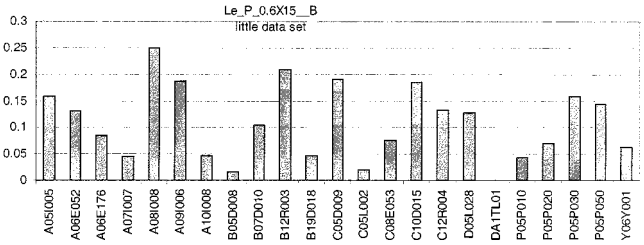
parm1 = "NA"
parm2 = "100020:LINEWIDTHSUMMARY: AVGFEATURES"
parm3 = "100021:LINEWIDTHSUMMARY: SIGMAFEATURES"
parm4 = "100024:LINEWIDTHSUMMARY: STEPPEREXPOSURE"
parm5 = "10017:SETUPDATA:FOCUS"
parm6 = "10018:SETUPDATA:EXPOSURE"
parm7 = "21001:MEANTHICKNESS:STDAVERAGE"
parm8 = "21001:POLYTHICKNESS:STDAVERAGE"
parm9 = "21002:MEANTHICKNESS:STDRANGE"
parm10 = "21002:POLYTHICKNESS:STDRANGE"
parm11 = "21002:REMAININGFOX:STDAVERAGE"
parm12 = "21002:REMAININGGATEOXTK: STDAVERAGE"
parm13 = "21002:REMAININGPOLY:STDAVERAGE"
parm14 = "21002:REMANINGGATEOXTK: STDAVERAGE"
parm15 = "21003:MEANTHICKNESS:STDRANGE"
parm16 = "21003:REMAININGFOX:STDAVERAGE"
parm17 = "21003:REMAININGFOX:STDRANGE"
parm18 = "21003:REMAININGGATEOXTK:STDRANGE"
parm19 = "21003:REMAININGPOLY:STDRANGE"
parm20 = "21003:REMANINGGATEOXTK:STDRANGE"
parm21 = "21003:TWU:AVERAGEITEM24&6"
parm22 = "21004:REMAININGPOLY:STDRANGE"
parm23 = "31001:%STANDARDDEV.:STDAVERAGE"
parm24 = "31001:%STANDARDDEVIATION: STDAVERAGE"
parm25 = "31001:MEANSHEETRESISTNCE: STDAVERAGE"
parm26 = "31001:POLY%STDDEVIATION:STDAVERAGE"
parm27 = "31002:%STANDARDDEV.:STDRANGE"
parm28 = "31002:MEANSHEETRESISTNCE: STDAVERAGE"
parm29 = "31002:MEANSHEETRESISTNCE:STDRANGE"
parm30 = "31002:POLY%STDDEVIATION:STDRANGE"
parm31 = "31003:%STANDARDDEVIATION:STDRANGE"
parm32 = "31003:MEANSHEETRESISTNCE: STDAVERAGE"
parm33 = "31003:MEANSHEETRESISTNCE:STDRANGE"
parm34 = "41001:%STANDARDDEVIATION: STDAVERAGE"
parm35 = "41001:MEANSHEETRESISTNCE: STDAVERAGE"
parm36 = "41002:%STANDARDDEVIATION:STDRANGE"
parm37 = "41002:MEANSHEETRESISTNCE:STDRANGE"
parm38 = "51001:%STANDARDDEVIATION: STDAVERAGE"
parm39 = "51002:%STANDARDDEVIATION:STDRANGE"
parm40 = "61001:LKBKFORDELTA:STDAVERAGE"
parm41 = "61001:PARTICLEDELTA:STDAVERAGE"
parm42 = "61003:LKBKFORDELTA:STDRANGE"
parm43 = "61003:PARTICLEDELTA:STDRANGE"
parm44 = "71001:HAZE:STDAVERAGE"
parm45 = "71003:HAZE:STDRANGE"
parm46 = "90030:WAFLWDATA:LINEAVG1"
parm47 = "90031:WAFLWDATA:LINEAVG2"
parm48 = "90032:WAFLWDATA:LINEAVG3"
parm49 = "90033:WAFLWDATA:LINEAVG4"
parm50 = "90034:WAFLWDATA:LINEAVG5"
parm51 = "90035:WAFLWDATA:LINESIG1"
parm52 = "90036:WAFLWDATA:LINESIG2"
parm53 = "90037:WAFLWDATA:LINESIG3"
parm54 = "90038:WAFLWDATA:LINESIG4"
parm55 = "90039:WAFLWDATA:LINESIG5"

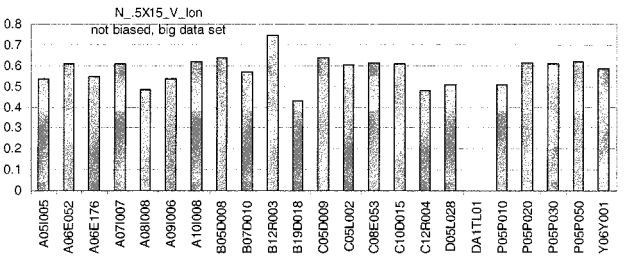


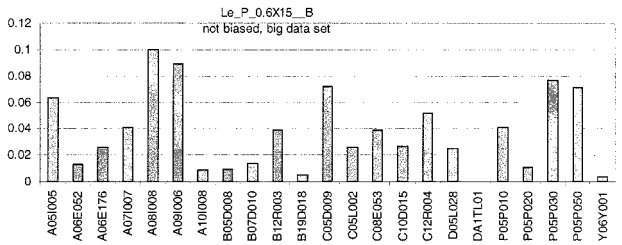


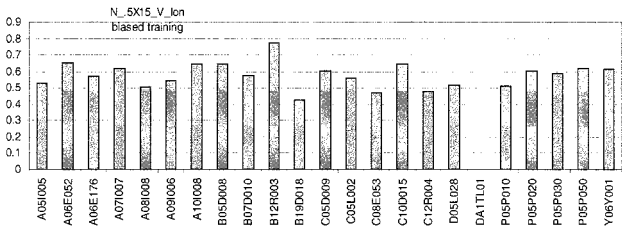

Le_PHC.6X15__B
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

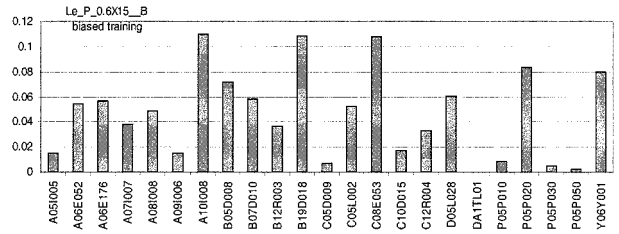

N_.5X15_V_Ion
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

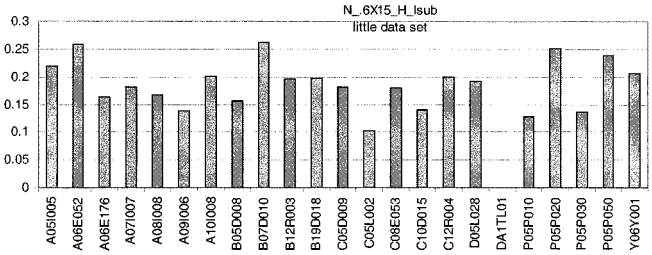

Le_P_0.6X15__B
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

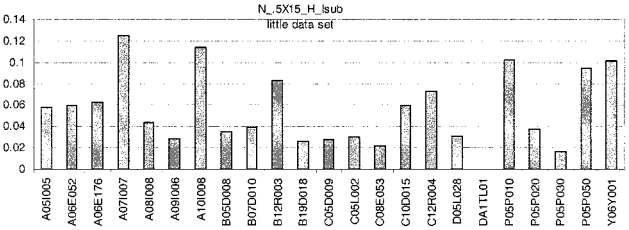

N_.5X15_V_Ion
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

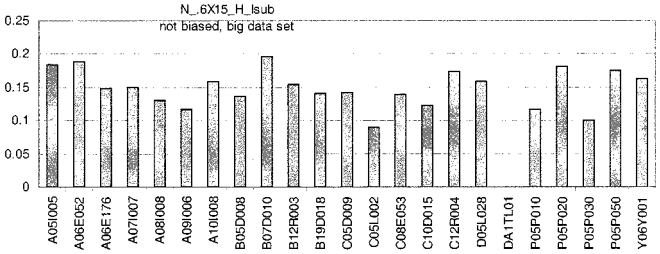

Le_P_0.6X15__B
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

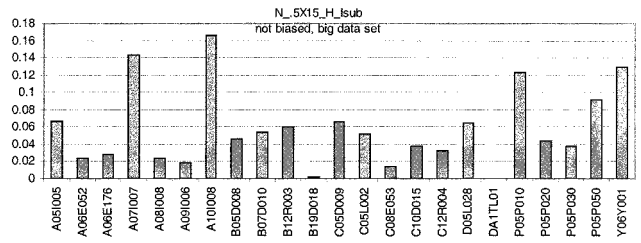

N_.5X15_V_Ion
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

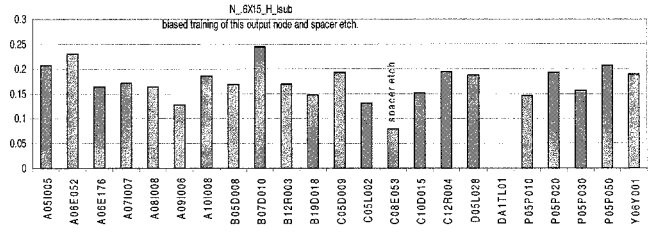

Le_P_0.6X15__B
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

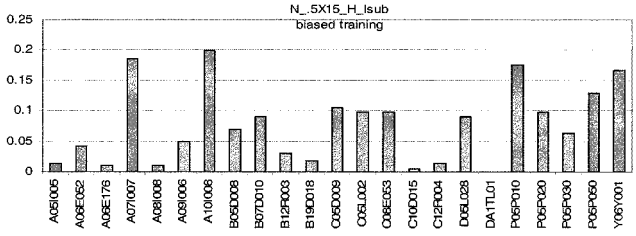

N_.6X15_H_lsub
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

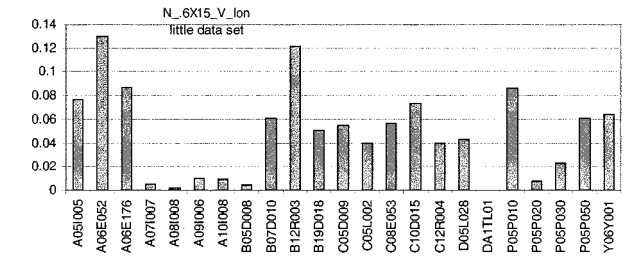

N_.5X15_H_lsub
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

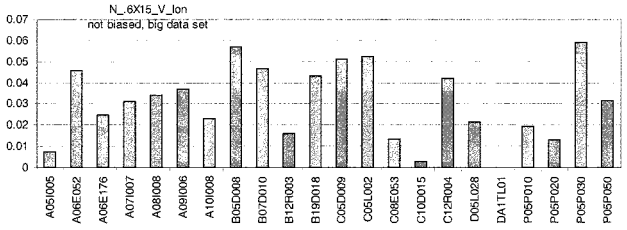

N_.6X15_H_lsub
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

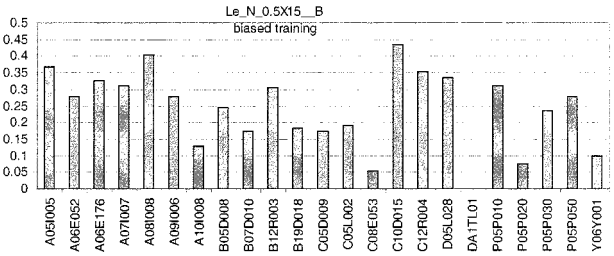

N_.5X15_H_lsub
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

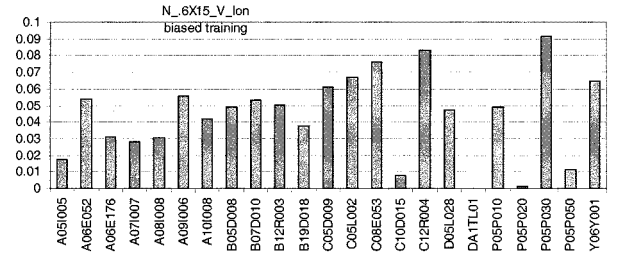

N_.6X15_H_lsub
biased training of this output node and spacer etch.
spacer etch
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

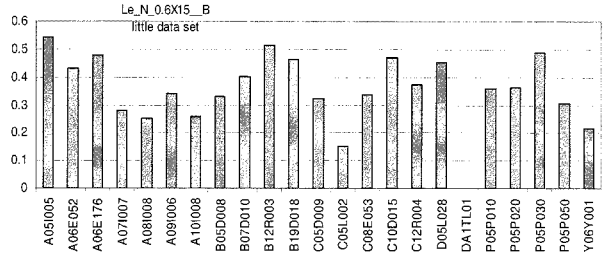

N_.5X15_H_lsub
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

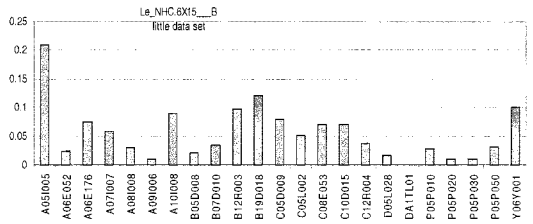

N_.6X15_V_Ion
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

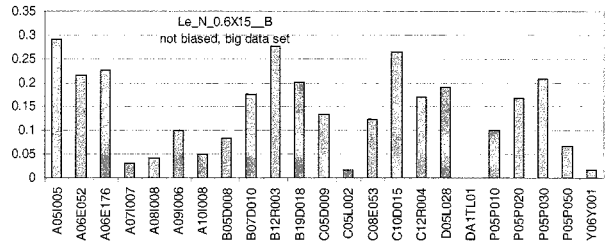

N_.6X15_V_Ion
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050

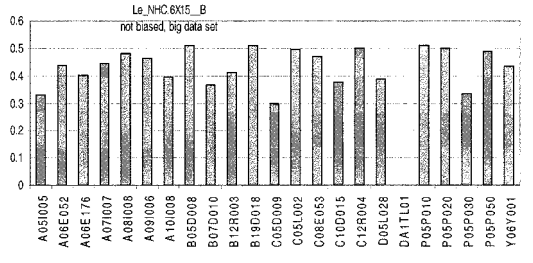

Le_N_0.5X15__B
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

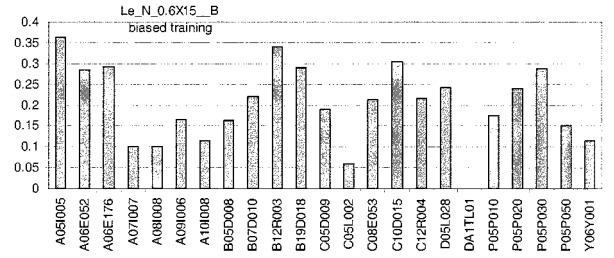

N_.6X15_V_Ion
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

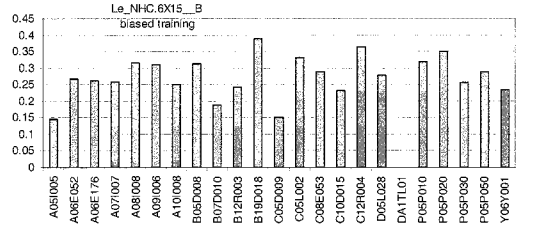

Le_N_0.6X15__B
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

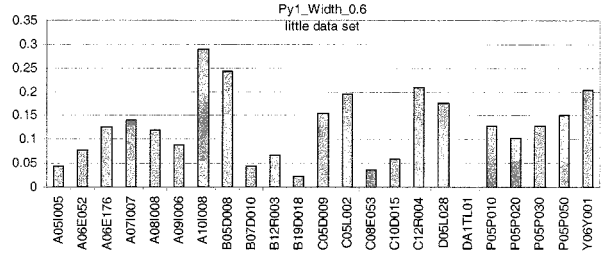

Le_NHC.6X15__B
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

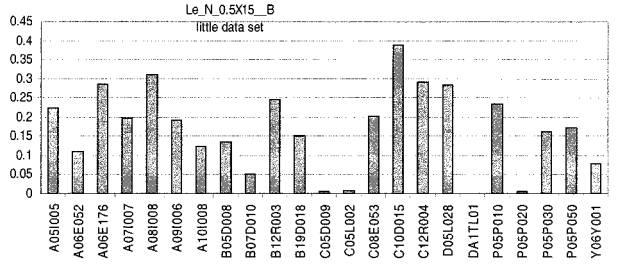

Le_N_0.6X15__B
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

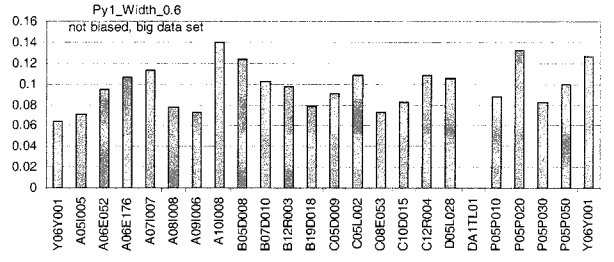

Le_NHC.6X15__B
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

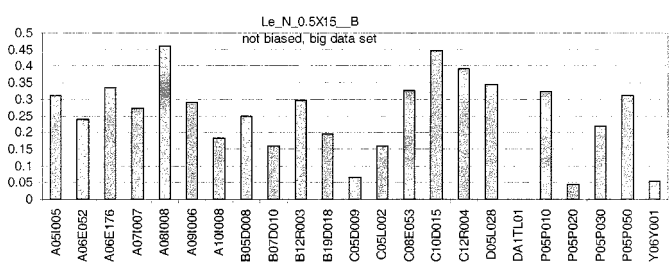

Le_N_0.6X15__B
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

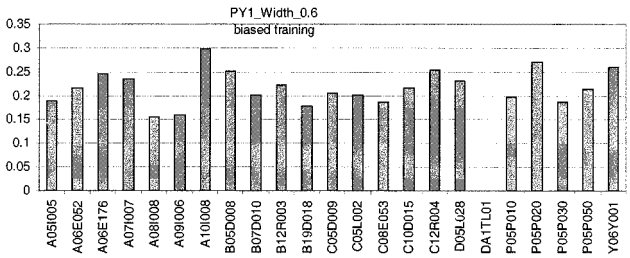

Le_NHC.6X15__B
biased training
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001

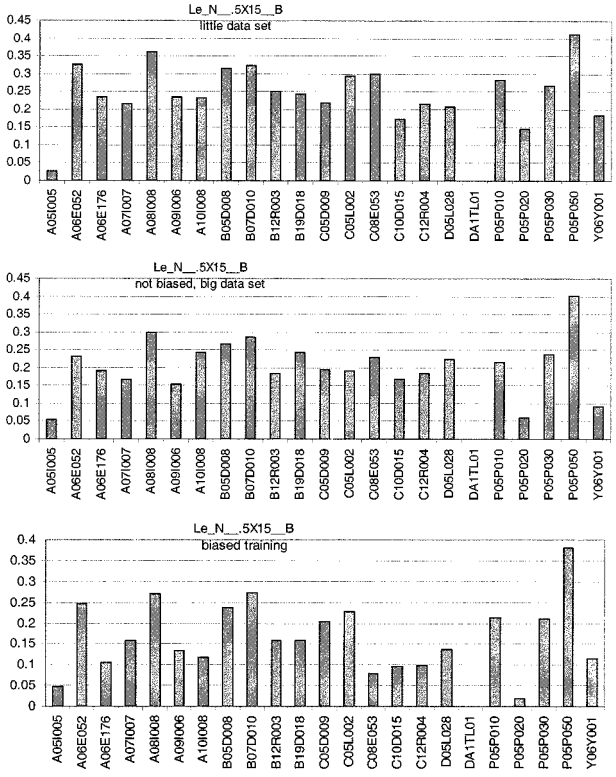

Py1_Width_0.6
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001


Le_N_0.5X15__B
little data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001


Py1_Width_0.6
not biased, big data set
Y06Y001 A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001


Le_N_0.5X15__B
not biased, big data set
A05I005 A06E052 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001


PY1_Width_0.6
biased training
A05I005 A06E062 A06E176 A07I007 A08I008 A09I006 A10I008 B05D008 B07D010 B12R003 B19D018 C05D009 C05L002 C08E053 C10D015 C12R004 D05L028 DA1TL01 P05P010 P05P020 P05P030 P05P050 Y06Y001



ACKNOWLEDGMENT

The authors thank D. Ibbotson, J. R. Hoyer, and K. Orlowsky for support of this project. They thank J. Harvey for technical discussions, R. Mohn for comments on databases, T. Frederick for assistance in obtaining data relevant to pattern density, and A. Guruprasad for assistance in installing the system on host computers in the fab.

EAR thanks the following people: J. Card, of NeuMath Inc., for discussions on arbitration networks, pareto analysis, and learning machine architectures, R. Frye and D. D. Lee, both of Bell Labs, for technical discussions relating to pareto analysis, dimensionality reduction and learning machine architecture, V. Vapnik, of AT&T Labs, for discussions on issues of learning with small data sets and comments on solving difficult practical problems, G. S. May, of Georgia Institute of Technology, and J. Plummer, independent consultant to the semiconductor industry, for reading the manuscript, and a big thanks to C. J. Burges, Bell Labs (now at Microsoft Research), for reading an early version of the manuscript, for many technical discussions, and for letting Mr. Rietman bounce crazy ideas off him.

REFERENCES

[1] E. A. Rietman, D. J. Friedman, and E. R. Lory, "Pre-production results demonstrating multiple-system models for yield analysis," *IEEE Trans. Semicond. Manuf.*, vol. 10, pp. 469–481, 1997.

[2] L. A. Doty, *Statistical Process Control*. New York: Industrial Press, 1991.

[3] M. Beneke, L. M. Leemis, R. E. Schlegel, and B. L. Foote, "Spectral analysis in quality control: A control chart based on the periodogram," *Amer. Stat. Assoc. Technometrics*, vol. 30, no. 1, pp. 63–70, 1988.

[4] Y. Guo and K. Dooley, "Identification of change structure in statistical process control," *Int. J. Prod. Res.*, vol. 30, no. 7, pp. 1655–1669, 1992.

[5] A. E. Smith, "X-bar and R control chart interpretation using neural computing," *Int. J. Prod. Res.*, vol. 32, no. 2, pp. 309–320, 1994.

[6] H. B. Hwarng and N. F. Hubele, "X-bar chart pattern recognition using neural networks," *ASQC Qual. Congr. Trans.*, pp. 884–889, 1991.

[7] ——, "Back-propagation pattern recognizers for X-bar control charts: Methodology and performance," *Comput. Ind. Eng.*, vol. 24, no. 2, pp. 219–235, 1993.

[8] M. D. Baker, C. D. Himmel, and G. S. May, "Time series modeling of reactive ion etching using neural networks," *IEEE Trans. Semicond. Manuf.*, vol. 8, pp. 62–71, 1995.

[9] H. B. Hwarng, "Multilayer perceptrons for detecting cyclic data on control charts," *Int. J. Prod. Res.*, vol. 33, no. 11, pp. 3101–3117, 1995.

[10] T. R. Turner, "Correlation of real-time monitored process module parameters and wafer results," *SPIE*, vol. 1593, pp. 145–156, 1991.

[11] E. D. Boskin, C. J. Spanos, and G. J. Korsh, "A method for modeling the manufacturability of IC designs," *IEEE Trans. Semicond. Manufact.*, vol. 7, pp. 298–156, Aug. 1994.

[12] T. S. Kim and G. S. May, "Intelligent control of via formation by photosensitive BCB for MCM-L/D applications," *IEEE Trans. Semicond. Manuf.*, vol. 12, pp. 503–515, 1999.

[13] S. Wolf, *Silicon Processing for the VLSI Era, Volume 2—Process Integration*. Long Beach, CA: Lattice Press, 1990.

[14] B. G. Streetman, *Solid State Electronic Devices*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1980.

[15] W. W. S. Wei, *Time Series Analysis*. Redwood City, CA: Addison-Wesley, 1990.

[16] L. Devroye, L. Gyorfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*. New York, NY: Springer, 1996.

[17] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[18] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[19] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. Cambridge, MA: MIT Press, 1995.

[20] T. Masters, *Practical Neural Network Recipes in C++*. New York: Academic, 1993.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986.

[22] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[23] ——, *Estimation of Dependencies Based on Empirical Data*. New York: Springer-Verlag, 1982.

[24] Y. S. Abu-Mostafa, "Hints and the VC dimension," *Neural Computation*, vol. 5, pp. 278–288, 1993.

[25] E. B. Baum and D. Haussler, "What size net gives valid generalization?," *Neural Computation*, vol. 1, pp. 151–160, 1989.

[26] I. Guyon, V. N. Vapnik, B. Boser, L. Bottou, and S. A. Solla, "Structural risk minimization for character recognition," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufman, 1992.

[27] V. N. Vapnik, E. Levin, and Y. Le Cun, "Measuring the VC-dimension of a learning machine," *Neural Computation*, vol. 6, pp. 851–876, 1994.

[28] S. B. Holden and M. Niranjan, "On the practical applicability of the VC-dimension bounds," *Neural Computation*, vol. 7, pp. 1265–1288, 1995.

[29] D. Haussler, M. Kerns, M. Opper, and R. Schapire, "Estimating average case learning curves using bayesian statistical physics and VC-dimension methods," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992.

[30] Y. L. Huang, T. F. Edger, D. M. Himmelblau, and I. Trachtenberg, "Constructing a reliable neural network model for a plasma etching process using limited experimental data," *IEEE Trans. Semicond. Manuf.*, vol. 7, pp. 333–344, Aug. 1994.

[31] P. R. Cohen, *Empirical Methods for Artificial Intelligence*. Cambridge, MA: MIT Press, 1995.

[32] B. Kim and G. S. May, "An optimal neural network process model for plasma etching," *IEEE Trans. Semicond. Manuf.*, vol. 7, pp. 12–21, Feb. 1994.

[33] ——, "Reactive ion etching modeling using neural networks and simulated annealing," *IEEE. Trans. Comp., Packag., Manuf. Technol.*, vol. 19-C, pp. 3–8, Jan. 1996.

[34] A. W. van der Vaart and J. A. Wellner, *Weak Convergence and Empirical Processes*. New York: Springer-Verlag, 1996.

[35] C. C. Klimasauskas, "Neural nets tell why," *Dr. Dobb's Journal*, pp. 16–24, Apr. 1991.
[36] A. Deif, *Sensitivity Analysis in Linear Systems*. Berlin: Springer-Verlag, 1983.

Edward A. Rietman, photograph and biography not available at the time of publication.

Stephen A. Whitlock, photograph and biography not available at the time of publication.

Milton Beachy, photograph and biography not available at the time of publication.

Andrew Roy, photograph and biography not available at the time of publication.

Timothy L. Willingham, photograph and biography not available at the time of publication.