Integration of artificial neural networks and simulation modeling in a decision support system

Kenney LeCroy
klecroy@hotmail.com
INTEGRATION OF ARTIFICIAL NEURAL NETWORKS AND SIMULATION MODELING IN A DECISION SUPPORT SYSTEM

by
Kenney LeCroy
B.I.E., Auburn University, 1992

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Industrial Engineering College of Engineering University of Central Florida Orlando, FL

Spring Term 1994
ABSTRACT

A simulation based decision support system is developed for AT&T Microelectronics in Orlando. This system uses simulation modeling to capture the complex nature of semiconductor test operations. Simulation, however, is not a tool for optimization by itself. Numerous executions of the simulation model must generally be performed to narrow in on a set of proper decision parameters. As a means of alleviating this shortcoming, artificial neural networks are used in conjunction with simulation modeling to aid management in the decision making process. The integration of simulation and neural networks in a comprehensive decision support system, in effect, learns the reverse of the simulation process. That is, given a set of goals defined for performance measures, the decision support system suggests proper values for decision parameters to achieve those goals.
ACKNOWLEDGEMENTS

At the University of Central Florida, much thanks goes out to Dr. Jose Sepulveda and Dr. Ralph Rogers who took the time to be on my thesis committee. I would also like to thank Abel Fernandez, who was responsible for coordinating my acceptance into the fellowship program at U.C.F. and handled all questions which arose during my stay at U.C.F. Special thanks is extended to my advisor, Dr. Mansooreh Mollaghasemi, who worked so hard and cared so much to allow me to finish my Masters degree in the two years that was allotted under my fellowship. I would also like to thank Grisselle Centeno, my partner in crime, for making work at AT&T and school life, much more enjoyable.

I would like to thank the probe personnel at AT&T MicroElectronics for their unquestioning assistance in allowing the completion of this project. I would especially like to thank JoAnn Newman and Mary Jansen who were instrumental in the success of this project and who were responsible for hiring me on at the Orlando facility and giving me the two year Masters Fellowship at the University of Central Florida.

I would especially like to thank my parents. Without them I would not have accomplished much in life. Thanks is given to them for allowing me to be where I am today.
# TABLE OF CONTENTS

LIST OF TABLES ...................................................................................................................... vi

LIST OF FIGURES ...................................................................................................................... vii

CHAPTER ONE - INTRODUCTION ......................................................................................... 1
  Need for Research .................................................................................................................. 4
  Objectives of the Research ................................................................................................. 6
  Organization of the Thesis ................................................................................................... 8

CHAPTER TWO - LITERATURE REVIEW ............................................................................. 9
  Simulation Optimization .................................................................................................... 9
    Single Response Optimization ......................................................................................... 10
    Gradient Estimation ....................................................................................................... 14
    Iterative Search Methods .............................................................................................. 18
  Multiple Response Optimization ...................................................................................... 21
  Decision Support Systems ................................................................................................. 23
    Category I Systems ......................................................................................................... 23
    Category II Systems ....................................................................................................... 25
    Category III Systems ..................................................................................................... 26

CHAPTER THREE - SIMULATING THE TEST FLOOR ............................................................ 33
  Scope of the Simulation .................................................................................................... 33
  Modeling the Test Floor .................................................................................................... 34
  Simulating the Test Floor ................................................................................................. 38

CHAPTER FOUR - BACKGROUND ARTIFICIAL NEURAL NETWORKS ...................... 40
  Overview and Description ............................................................................................... 40
  Training Neural Networks .............................................................................................. 45

CHAPTER FIVE - EXPERIMENTING WITH NEURAL NETWORKS ............................. 50
  Methodology -- Preparing for Experimentation .............................................................. 50
  Methodology -- Final Experimentation ........................................................................... 57
  Results of Experiment ..................................................................................................... 63

CHAPTER SIX - INTEGRATION OF SIMULATION AND NEURAL NETWORKS IN THE DSS 73
  The Menu System ........................................................................................................... 73
Integration of Artificial Neural Networks into the DSS .................. 86
CHAPTER SEVEN - CONCLUSIONS FROM THE RESEARCH ............... 90

APPENDICES ................................................................. 95
1. Test Floor Process Steps ............................................. 96
2. Simulation Model Assumptions and Details .......................... 100
   Product Mix and Arrival of Lots ................................. 101
   Routing of Products Through the Test Area ....................... 102
   Scheduling Strategies Which May Be Employed .................... 106
   Maintenance Issues .................................................. 109
   Assumptions About Operational Metrics Predicted ............... 112
   Probe Personnel as They Relate to the flow of Lots ............. 115
   Detailed Description of Lot Flow Through Test Area .......... 119
4. SIMAN code for Simulation Mode ................................. 135
   Model File Listing .................................................. 136
   Experimental File Listing ......................................... 147
5. Comparisons Between Neural Networks and Other Methods ....... 152
LIST OF TABLES

1. Simulation Outputs and Their Binary Representation .................................. 55
2. Simulation Inputs and Their Binary Representation ....................................... 55
3. Values Used for Simulation Inputs When Generating Training Pairs .............. 56
4. Conversion of an Example Training Pair -- Inputs ...................................... 61
5. Conversion of an Example Training Pair -- Outputs ..................................... 61
6. Desired Simulation Outputs for Initial Test Case .......................................... 68
7. Simulation Inputs Suggested by Neural Network ........................................... 68
8. Comparison of Desired and Actual Simulation Output ................................... 69
9. Comparison of Neural Network and Other Methods ..................................... 71
10. Work Schedules and Queuing Strategies ..................................................... 77
## LIST OF FIGURES

1. Test Floor Layout ................................................................. 35
2. Test Floor Product Flow .......................................................... 36
3. Neural Network Schematic View ............................................... 43
4. Sigmoid Transfer Function .................................................... 46
5. XERION Code Used to Create a Neural Network ......................... 51
6. Neural Network Architecture as Detailed by XERION Code in Figure 5 53
7. Neural Network Architecture for a 2 Output to 4 Input Mapping .......... 58
8. Neural Network Used for Experimentation .................................. 59
9. Neural Network Error During Training .................................... 64
10. Neural Network Architecture Used for Processing Desired Outputs ...... 66
11. Neural Network Architecture Used for Determining Proper Inputs ........ 67
12. Test Floor Simulation Menu .................................................... 74
13. Output from Option 1 of Test Floor Simulation Menu .................... 75
14. Sample Input File ................................................................. 78
15. Output from Option 3 of Test Floor Simulation Menu .................... 80
16. Sample Demand File ............................................................. 81
17. Sample Code Information File ................................................ 82
18. Output from Option 6 of the Test Floor Simulation Menu ................ 83
19. Output from Option 7 of the Test Floor Simulation Menu ................ 85
20. Histogram of Lot Interarrival time -- 100 data points .................................................. 103
21. Histogram of Lot Interarrival time -- 500 data points .................................................. 103
22. Histogram of Lot Interarrival time -- 999 data points .................................................. 103
23. Histogram of Production Lot Sizes ..................................................................................... 105
24. Histogram of Engineering Lot Sizes .................................................................................... 105
25. Distribution of Time Between Machine Failures ................................................................. 113
26. Distribution of Time to Repair .............................................................................................. 113
27. Typical Triangular Distribution .............................................................................................. 117
28. Histogram for Probe Rework Time ....................................................................................... 127
29. Histogram for Inspection Time ............................................................................................... 127
30. Moving Average Plot for Lot WIP ......................................................................................... 132
31. Correllologram for Lot WIP .................................................................................................. 132
32. Confidence Interval for Lot WIP .......................................................................................... 133
33. Confidence Interval for Cycle Time. ..................................................................................... 133
CHAPTER ONE
INTRODUCTION

Decision making techniques are becoming increasingly more important in the management of all businesses. In order to be competitive, a company's management must make efficient use of all its available resources. Whenever possible, companies use decision making tools which are designed to help in optimizing the use of their resources. However, most decision making tools which guarantee optimality address problems which are fairly straightforward and simple. Often times these problems involve only one single objective which the decision maker wants to achieve by employing his resources in an optimal fashion. Techniques, such as linear programming, nonlinear programming, and statistical techniques have proven themselves as efficient means of optimizing decisions which involve a single objective. However, most decisions made in business situations involve a number of objectives which are all important to the decision maker. By using the techniques listed above, a decision maker is not able to incorporate all the objectives which he or she may feel is important. In recent years techniques have been developed in the area of decision making which address those decisions involving more than one objective. Some of these multi-attribute decision techniques include the analytical hierarchy process (AHP), which attempts to assess the relative importance of the decision maker's objectives in a hierarchical framework. Other techniques use straightforward weighting schemes and a set of appropriate utility functions. Utility functions are used to
directly assign weights to user objectives and then these weights are incorporated into an additive or multiplicative model as a means of evaluating each alternative. Although these techniques are quite useful, they are only able to assess the performance of a few, predefined alternatives. In many cases, there is an infinite or very large number of alternatives which the decision maker must evaluate. Furthermore, the decision maker is often not aware of all the alternatives which are available.

In addition to the complexities listed above, it is often difficult [if not impossible] to describe a decision scenario in terms of a mathematical model. The techniques stated thus far require the decision maker to interpret a real life system as an abstract model represented by a mathematical formula. When a problem does not lend itself easily to mathematical formulation, other techniques must be used. One such technique is simulation. Unlike mathematical modeling, simulation does not require the abstraction of a problem into a well defined formula. When using simulation, the decision maker recreates (or simulates) an actual real world system in a computer model. If constructed properly, the computer model allows the decision maker to predict what will happen in the real world system if a certain course of action is followed. Such a tool is very powerful. It allows the user to have a working model of his or her decision situation. However, when doing analysis with simulation output, the decision maker is faced with an enormous set of alternatives from which he or she must choose. Although any given alternative can be tested by the computer model, how does one decide on which alternatives to test. Techniques such as statistical analysis have proven themselves useful in this area. By
designing an appropriate experiment and analyzing the results, an analyst can develop an equation which allows the responses of the simulation to be predicted. Using these statistical methods (known as Response Surface Methodology), an analyst can gain a fairly well defined idea of what alternatives work the best. In such an experiment, output from the computer model is analyzed by statistical methods in much the same way as real life data would be analyzed. Even though statistical techniques are useful in this area, the number of alternatives which must be tested by the computer model is quite large, and the time required to do the testing and subsequent analysis is often prohibitive. Furthermore, as stated above, statistical techniques are best suited for those decisions which consider only one objective. However, in defense of the techniques listed thus far, it must be stated that many real world decisions have been made with the aid of these techniques, and many of these algorithms have been implemented into helpful computer programs.

In response to the difficulties addressed with most real world decision making situations, much research has been done. Most of this research is aimed at developing techniques which will address real life scenarios that involve many alternatives and many objectives. Some of the more recent developments involve the use of artificial intelligence in the form of rule-based systems, expert systems, or case-based reasoning systems. These forms of artificial intelligence allow the knowledge of the decision maker to be represented in a computer program. By empowering the computer with the knowledge of an expert, increased reliability, consistency, and speed can be obtained. In many situations such a computer tool is invaluable. Many computer based control systems, computer aided
design systems, and decision support systems for management have been developed using these tools. One of the difficulties involved with these systems, however, is the need to explicitly define the knowledge of experts in such a way that this information can be encoded into the serial-type processing environment of the digital computer.

Another form of artificial intelligence which has been receiving a great deal of attention lately is artificial neural networks (ANN's). Artificial neural networks are based on the idea of biological neurons. Unlike other AI techniques, ANN's are able to "learn". That is, the need to abstract or explicitly define a system is no longer needed.

The research presented here addresses the integration of neural networks and simulation in a decision making situation. The system being developed is to be used by the management at AT&T's Microelectronics production facility in Orlando, Florida.

**Need for Research**

The desire to develop a decision support system for AT&T was first proposed by management. Upper level management at the microelectronics fabrication facility felt the need to be able to objectively justify any change made in business operations. These operations, at the manufacturing level, involve the highly complex tasks of fabricating microelectronics devices for both logic and memory semi-conductor devices and the 100% testing of every fabricated component. In fact, these two operations (fabrication and testing) are separated within the plant. Each operation has its own set of engineers,
production operators, and management personnel. It is the management of the testing facility which has suggested the research being presented.

In order to make use of its resources in the most beneficial manner, management must continuously review and study data produced from normal, everyday business operations. Management has, at its disposal, a team of engineers and production supervisors which aid in the management task. This team collects and analyzes data, and generates reports for management personnel. The availability of this kind of expertise and easy access to an abundance of data contribute greatly to the success of future business operations. As an ongoing process of continuously seeking to improve overall company performance, the management team has proposed the development of a tool which would have the ability to forecast performance based on decisions being made. Such a tool would allow management to predict and better prepare for future events. In addition, a decision being made would be quantifiably justified as a result of analysis with the new decision support tool.

Difficulties that arise when trying to predict the behavior of test floor operations are due to constantly changing product mixes. The requirements placed on testing resources can vary greatly as does the time required to process each type of product. For example, a microelectronic circuit, or individual chip, could take as little as 1/2 second to be tested or as much as 6 seconds to be tested, depending on the level of technology involved and the degree of complexity in the circuit. Furthermore, the routing of different products through the test operation can vary among different technologies. Some technologies require certain testing processes that others do not.
Along with these normal, predictable occurrences, some products may have their routing changed during processing in order to correct or analyze a problem which may have occurred in a prior processing step. In short, the task of predicting behavior in such a dynamic system is quite difficult at best.

**Objectives of the Research**

The main goal of this research was to develop a useful and practical decision support system for AT&T's management. This system is easy to use and allows for the effect of any decisions made by management to be predicted before they take place. Additionally, the stochastic nature of test floor occurrences are taken into account when any predictions are made. Furthermore, as product mixes change, the ability to analyze the effects on test operations was of utmost concern. As a result, the system facilitates easy input of different product mixes for analysis. The analyst (the person using the decision support system) also has the ability to analyze the effects of changing types and quantities of resources, changing work schedules, changes in scheduling strategies associated with product flow for a given product mix and varying levels of rework that may be required. To aid the analyst, a means of easily entering this type of information was made available. With a system which facilitates the changing of simulation scenarios, the user of the system can quickly analyze a number of alternatives under consideration. The relative performance of alternatives are based on measures of effectiveness such as work-in-process inventory levels, cycle times, throughput, and other related performance criteria.
In addition to allowing the easy analysis of alternatives, the system being developed will include a means of analyzing predictions being made. With a test operation as complex and dynamic as the one being analyzed, a large number of alternatives are often available. Deciding which alternatives to use, even if all are known to be available to the analyst, is often difficult. In response to this difficulty, the ability to direct the analyst toward a proper set of alternatives is included. This ability allows the user to test alternatives which he or she may or may not have previously considered. In essence, the system suggests a well suited strategy for accomplishing management objectives under the given set of operating conditions (product mix, routing, and level of rework). The main advantage of such a system is that it allows for the integration of both a predictive model and an output analyzer to aid in the decision making process. Increased speed and ease of analysis should be the main advantages gained by the decision maker. However, unlike other decision support systems which have been developed, the system being constructed here attempts to use the technology of artificial neural networks as a means of processing simulation output. The application of artificial neural networks as an output processor in a decision support system is a new idea and is, consequently, the main focus of this thesis. The following section discusses the organization of the thesis and lists the major steps taken in attempting to meet the objectives outlined thus far.
Organization of the Thesis

The work presented here is divided into seven chapters. Chapter two discusses some of the research which has been done in the areas of simulation optimization, development of decision support systems, and the applications of artificial neural network technology. Chapter three describes the test floor at AT&T MicroElectronics and the simulation model used in the decision support system (DSS) to model this system. Chapter four introduces the concept of artificial neural networks, and chapter five details experiments which were performed in exploring the applicability of neural networks in a DSS. Chapter six involves the integration of the components that were designed and constructed for the DSS. Finally, chapter seven offers conclusions that can be drawn from this research.
CHAPTER TWO
LITERATURE REVIEW

The material being presented in this chapter focuses on three areas of research:
(1) simulation optimization (2) development of decision support systems, and (3) applications of artificial neural network technology. The research being presented serves as a background for the three study areas mentioned above and, in doing so, points out an area of study which has received little attention. This area involves the application of neural networks as a means learning simulation input-output relationships. The research also suggests that no attempt has been made to integrate the aforementioned technology in a real world decision support system.

Simulation Optimization

The topic of simulation optimization has become increasingly popular in recent years. Much attention has been drawn to this area of study as a result of the need for efficiency and reliability in predicting simulation input-output relationships. Additionally, the specific area of simulation optimization is not like other areas of input-output modeling. With simulation, in general, there may exist a mathematical function which represents the behavior of the simulated system and thus can be directly minimized or maximized. But, however, the analyst must be careful to use the proper techniques to come up with the proper equation(s). An additional concern is that the output generated
by a simulation model is stochastic in nature even though it may be based on deterministic input variables (Azadivar, 1992). Furthermore, it is often the case that it is not economically feasible or sufficient time is not available to iteratively run a simulation as a means of predicting the behavior of the system being modeled. As a simulation model grows in complexity and requires the consideration of more and more decision (input) variables, the number of possible combinations of input variables increases dramatically. But, by modeling the input-output relationships of the simulation model, an analyst is more able to efficiently use the simulation by predicting its outcome.

This review offers a survey of the tools that are currently being used or have been considered as a means of modeling simulation input-output relationships. Two broad areas of simulation optimization are discussed. These are (I) Single Response optimization and (II) Multiple Response optimization. Much work has been done in the area of single response optimization due to the abundance of tools available to analyze a one-dimensional response vector. Multiple response optimization is a much more open field of study due to the complex nature of relationships that may exist between output variables.

**Single Response Optimization**

Analysis of simulation output which involves only one response variable can be categorized into two general areas: (1) analytical/mathematical modeling of the input-output relationships, and (2) iterative approaches that do not attempt to directly
model input-output relationships, but rather converge on a maximum or minimum value through a strategic and repetitive experimentation procedure. Both of these approaches have as their goal the direct minimization or maximization of some output value $Y$ which is based on choosing the appropriate values of the input vector $X$. The first area discussed will be analytical/mathematical modeling. One of the most widely accepted and popular methods of analytically modeling input-output relationships is response surface methodology (RSM). This method is based on the mathematical model of a polynomial response surface. Polynomial response surfaces are of the form:

$$f(x) = \sum_{k=1}^{n} \beta_k p_k(x)$$

In this equation $f(x)$ represents the response, $p_k(x)$ represents a power function of $x$ such as $x_1^2, x_1^3, x_2^3 x_4^2$, etc., and $\beta_k$ represents the coefficient of the $p_k(x)$ term. This mathematical equation stems directly from the fact that any continuous function can be approximated by a Taylor series expansion. Once the polynomial response surface is identified, search techniques are used to optimize the function. This equation can take on many forms. A linear response surface model could be of the form:

$$f(x) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$$

Often times this model is adequate enough for the given situation. However, if curvature is detected in the response surface, a quadratic model may be needed:

$$f(x) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \beta_i x_i^2 + \sum_{i=1}^{n} \beta_{ij} x_i x_j$$

(where $i$ and $j$ are not equal)
In this equation, curvature in the response is represented by the last two terms which involve a power of two in a single input variable or an interaction of two or more input variables. The response surface is identified by starting with an initial point which is away from the optimal point. This point is referred to as a center point. The area immediately around this center point is investigated by means of experimentation. At first a linear model is used as the form of the equation which describes the response surface in the vicinity of the center point. An assumption being made in the experiment is that all input factors under consideration are represented by orthogonal vectors. This means that the cross product of the vectors which represent the effects of these factors is equal to zero and that they are at right angles to each other in the n-dimensional state space. Using experimental data, the coefficients of the n input factors can be estimated using the least-squares method, maximum likelihood method, or other appropriate method. If the linear model appears to be adequate in the vicinity of the center point (i.e. little evidence of curvature -- interaction coefficients are approximately 0, quadratic term coefficients can be estimated as being sufficiently close to zero, or evidence from a t-test does not indicate a significant difference between the models predicted values and the actual experimental values) then, by using the estimated coefficients, the experimenter can find the path away from the center point which has the steepest ascent (or descent). Once the steepest path is found, additional experimental points are tried which are directed along this path. Each additional experimental point is found as follows:

\[ x_{next} = x_{current} + \alpha [\nabla(x)] \]
In this equation, $\alpha$ represents the step size taken along the steepest path and $\nabla(x)$ is the gradient of the objective polynomial function computed from the estimated coefficients. A new center point is then found by searching along this path for the maximum (or minimum) value obtained during the additional experimental runs. The region in the vicinity of this new center point is then explored and the process repeats until the linear model no longer allows movement of the response towards a minimum or a maximum. At this point, a quadratic model is fit to the response surface and the coefficients of this polynomial expression are estimated through experimentation. Again an estimate of the gradient in this region of the state space is estimated, and improvements are made in the response by moving along the path of steepest ascent (descent). The procedure of finding a center point, moving along the path of the estimated gradient, and doing further experimentation continues until the estimated gradient becomes sufficiently close to zero and no improvement in the response is possible.

Some major advantages of using response surface methods in determining simulation input-output relationships are its wide acceptance and easily understood procedures which are based on statistical theory. Additionally, RSM often performs well in fitting most functions. However, the performance of this method does decline when complex functions are involved which have sharp ridges or flat valleys as denoted by Azadivar and Talavage (1980). A further consideration which might keep the simulation analyst from using RSM is the large number of simulation runs required.
Gradient Estimation

One thing that distinguishes many types of mathematical modeling from iterative search techniques is the need for estimating the gradient of an underlying polynomial function and then using this estimate in optimizing the response variable. In realizing the need for gradient estimation many techniques have been devised which attempt to quantify the change in the response variable as a function of the change in the input $x_i$. This estimate represents the following true quantity:

\[
\text{effect of } x_i = \frac{\delta y}{\delta x_i}
\]

One way of estimating this value is by using the finite difference equation:

\[
\frac{\delta y}{\delta x_i} = \frac{\left[ f(x_1, \ldots, x_i + \Delta x_i, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n) \right]}{\Delta x_i}
\]

All partial derivatives are estimated using this method and the resulting gradient is computed. It can be seen that this method is computationally easy, but it is a rather crude approach to estimating the gradient. Furthermore, it would take 1 initial simulation run + n additional runs to perturb each of the n input variables just to obtain a point estimate for each partial derivative.

Another way of estimating gradients is by using a simple procedure based on likelihood ratio estimators. This method is based on the concept of maximum likelihood estimators which work to minimize the residual sum of squares from a fitted model:

\[
S = \sum_{i=1}^{n} (y_{obs} - y_{pred})^2
\]
In this equation $S$ is the residual sum of squares, $y_{obs}$ is the observed response from the simulation run, and $y_{pred}$ is the predicted response using the mathematical model of the simulation and $n$ is the number of experimental points. The task for fitting a straight line ($y = \beta x$) translates into finding the value of $\beta$ which minimizes the value of $S$:

$$S = \sum_{i=1}^{n} (y_{obs} - \beta x_{exp})^2$$

In this equation $x_{exp}$ is the experimental input point used in the simulation run. Glynn (1987) notes ways of using likelihood ratio estimators as a means of estimating gradients for simulation optimization as does Reiman and Weiss (1986) and Rubinstein (1986, 1989). Like other statistically based methods, this approach also requires a large number of simulation runs to estimate gradients.

An interesting approach to gradient estimation has become increasingly popular as a research topic in the past few years. This technique is known as infinitesimal perturbation analysis (IPA). It was originally developed for real systems by Ho, Eyler, and Chien (1979). A more thorough compilation of IPA issues can be found in Ho and Cao (1991) and Ho (1992). This technique involves the perturbing of a single input variable and tracking the propagation of this change through the system until the end result is realized in the output response. One of the underlying assumptions is that the perturbed input variable is changed only an infinitesimal amount and as a result the order of events in the system is not changed. The main advantage of this approach is that it requires only one simulation run to estimate all gradients of the objective function. One drawback -- in the
case of more complex systems being simulated -- is that suitable algorithms have not been devised which allow the tracking of propagation of perturbed input variables. As noted by Glasserman (1988) and others the applicability of IPA is limited to a small number of simple models. Ho (1992) states that reliable IPA tracking algorithms have been devised for simple discrete-event dynamic systems such as (1) simple queuing networks, (2) simple networks with multi-class customers, (3) networks with blocking, and (4) some networks with state-dependent routing mechanisms. Johnson and Jackman (1989) have also presented algorithms for tracking propagation in simple serial transfer lines. Their approach involves the addition of variables in a SIMAN simulation model to collect perturbed values as they are propagated through the simulation model and realized by downstream elements of the system (or upstream processes if finite buffer capacities are involved and the possibility of blocking is present). The limiting factor with IPA, however, seems to be the ability to know when IPA can yield adequate results, what conditions are necessary for the application of IPA, and what algorithms are available for tracking the propagation resulting from perturbed input. Further research, however, was presented by Glasserman (1988) in an attempt to present sufficient structural conditions for the application of IPA and different IPA propagation tracking algorithms.

Another approach to the estimation of gradients is the use of a concept known as frequency domain analysis. The concept of frequency domain analysis involves the sinusoidal oscillation of input variables during a single simulation run as a means of detecting the sensitivity of an output variable in response to these oscillations. Work with
this subject has been done by Jacobsen and Schruben (1989). Additionally, Schruben and Cogliano (1987) address the application of frequency domain analysis as a means of identifying a response surface model that should be used to mathematically model a simulated system. In their approach input parameters were varied according to sinusoidal oscillations with each parameter having its own "driving frequency" with which it is oscillated. The theory is this: if responses of the simulation were sensitive to oscillations of the input parameters then predictable oscillations could be induced in those responses. Furthermore, varying the values of unimportant input parameters would not effect changes in the response. By viewing the spectral analysis which uses comparisons of a control run with the experimental run (in which the values of input factors are oscillated) important factors, powers of factors, and combinations of factors could be identified. The frequencies with which the response in the simulation displayed maximal values of the spectral signal-to-noise ratio (which reflects differences observed in the experimental run that were not present in the control run) coincided with the driving frequencies of the input parameters or combinations of the driving frequencies. The beauty of this approach is that the important factors present can be discovered and a rough estimation of their effect on the response can be found using just two simulation runs. The difficulty of this technique is that adequate means of varying input parameters and observing the effected change in a response variable are not easily implemented into existing simulation software. However, Jacobsen and Schruben (1991) have used this approach directly with the Newton search method as a means of optimizing a simulated system.
Besides the use of polynomial based methods discussed thus far, other mathematical models of simulation input-output relationships have been devised. A good overview of some of these techniques are presented by Barton (1992).

**Iterative Search Methods**

One well known iterative procedure is stochastic approximation. The advantage of this approach is that it allows the convergence of a regression function to a minimum or maximum based on stochastic observations (observations containing noise). This is a major plus when working with simulation output data. Early work on this method of search was done by Robbins and Monro (1951), Hotelling (1941), Friedman and Savage (1947), and Kiefer and Wolfowitz (1952). A simple form of a recursive relation used in stochastic approximation might take the form:

\[ x_{n+1} = x_n - a_n [y(x_n) - \alpha] \]

where \( \alpha \) is the expected or the desired value, \( x_n \) is the input value, and \( y(x_n) \) is the theoretical function, \( a_n \) is a series of decreasing real numbers whose sum as \( n \) approaches infinity is less than infinity. Based on this simple design, an iterative procedure can be performed which converges on \( \alpha \). Likewise a recursive relation can be set up which converges on a maximum or minimum such as the relation used by Robbins and Monro (1951) / Kiefer and Wolfowitz (1952):

\[ x_{n+1} = x_n + (a_n / 2c_n) [f(x_n + c_n) - f(x_n - c_n)] \]
where $a_n$ and $c_n$ are series of real numbers such that: $a_n < \infty$, the limit as $n$ goes to infinity of $(c_n) = 0$, and the limit as $n$ goes to infinity of $(a_n / c_n)^2 < \infty$. This recursive relation has been proven to converge on a maximum or a minimum of the given function.

The advantages of this approach lies within its ability to converge on a desired value despite the absence of a mathematical model which defines the inner workings of the simulation. A major disadvantage of this approach, like other approaches listed here, is the large number of simulation runs required to converge on an optimal response. An application of stochastic approximation method to simulation optimization can be found in Azadivar and Talavage (1980). Further reading on stochastic approximation and stochastic optimization can be found in books by Wasan (1969) and Rubinstein (1986).

Another iterative search technique which is quite similar to stochastic approximation is simulated annealing. This technique is described by Eglese (1990). The basis of simulated annealing is that it has as its ultimate objective the achievement of a global maximum. Often it is the case that the path of steepest ascent or descent will not lead to an overall optimal value. For this reason, simulated annealing allows travel in a direction other than along the path of the steepest gradient. Instead of simply following the steepest path, simulated annealing uses a controlled stream of random variables, which belong to a certain distribution, to determine movement of the experimental point. By using random numbers, movement in the direction of the steepest gradient is not assumed with a probability of one as is the case in other search techniques. To clarify this point, a standard simulated annealing algorithm is presented by Metropolis (1953). In this
procedure an initial starting point $X_0$ is chosen at random and another point $X_{1_{\text{trial}}}$ is selected using a random number and a probability distribution. If the trial state has a lower cost (value of the response function when searching for a global minimum) it is accepted with probability 1. Else it is accepted with a lower probability determined by relative costs. The possibility of accepting the next $X_{\text{trial}}$ even when it yields a worse value allows the escape from a local minimum. A form of simulated annealing based on the work of Metropolis (1953) would have a Markov process defined by:

$$
\Pr \left\{ X_{k+1(\text{trial})} = X_{k+1(\text{trial})} \mid X_{k+1(\text{trial})}, X_k \right\} = \begin{cases} 1 & \text{if} \ f(X_{k+1(\text{trial})}) < f(X_k) \\ e^{-\frac{f(X_{k+1(\text{trial})}) - f(X_k)}{T_k}} & \text{otherwise} \end{cases}
$$

The value $T_k$ is known as the "temperature" and its sequence of values as $k$ goes from 0 to infinity is known as the temperature schedule. Simulated annealing is the special case where $T_k$ goes to 0 slowly so that the state of the Markov chain converges in probability to the states which yield a globally minimum value of the process (function) under study (Mitra, Romeo, and Sangiovanni-Vincentelli, 1985). The effectiveness of using simulated annealing, reduces to the problem of generating an initial experimental point $X_0$, a random variable $X_k$, and an appropriate temperature schedule such that the convergence to minimum (maximum) is completed in as little time as possible. A variation of the standard simulated annealing method is used by Musser, Dhingra, and Blankenship (1993) in optimizing a general cost function associated with decisions made in the scheduling of jobs in a job shop.
Another search method is known as the simplex search. This search involves the construction of a simplex in the experimental space having \( p + 1 \) vertices (given \( p \) input factors). Each vertex of the simplex represents an experimental point. Simulation runs are performed at each of the vertices and the vertex yielding the worst result is dropped. The dropped point is then replaced by a point which is found by projecting the worst point through the centroid of the original simplex. The procedure is repeated until no improvement is seen in the response by deleting additional points. This search technique stems from the work of Nelder and Mead (1965). An extension of this method is the complex search which involves the addition of constraints to form a simplex in a feasible region. The drawback of these methods, other than the need for many simulation runs, is that the worst point chosen to be eliminated could actually not be the worst point. This is realized due to the fact that the responses recognized by the simulation are stochastic in nature. A modification of this approach is offered by Azadivar and Lee (1988) in which a program was developed that statistically compares the responses observed at the vertices of the simplex in an attempt to only eliminate those points which are shown to be significantly worse than the other points.

**Multiple Response Optimization**

Unlike single response optimization, multiple response optimization is not well understood. The difficulties which arise are due to the decisions regarding the relative importance of the decision criteria involved. Additionally, the stochastic nature of
simulation adds an extra burden on the analyst when trying to make decisions involving multiple response measures. Some of the approaches used in attacking this problem involve the formation of utility functions which assign weights to the decision criteria and the formation of cost functions which attempt to include multiple criteria in a single objective function. The application of a cost function and a linear programming is used by Moore, Apgar, and Lee (1990) in an attempt to optimize the scheduling of factory. They use simulation to generate product cost and factory performance measures which are then used to determine the coefficients in a linear program optimization equation. The solution from the linear program is used to determine the parameter settings for the next simulation run and the cycle is repeated. In this manner scheduling can be optimized in terms of the criteria included in the cost function.

Other approaches to multi-response optimization have included using a single primary response as the basis for optimization while maintaining certain minimum levels for the other responses under consideration (Biles 1975, 1977). Variations of goal programming have been used by Biles and Swain (1980) and others. Another approach uses multi-attribute value function methods (Mollaghasemi, Evans, and Biles 1991/Mollaghasemi and Evans, 1992). Another procedure based on the complex method was devised just for the problem of simulation optimization by Teleb and Azadivar (1992). This method incorporates the stochastic nature of simulation responses as a means of eliminating points which show low probability of belonging to the random vector which represents the best value for all objective functions.
Another approach to multiple response simulation optimization is presented in a paper by Shahraray and Maeschke (1990). Their work includes the construction of a complete simulation-based decision support system. An integral part of this system is the interaction with a user (usually management personnel) who is able to define the relative importance of decision criteria used in determining an optimal operational policy.

**Decision Support Systems**

The research being presented here on decision support system involves three different types or categories of systems. The first category involves systems which do not incorporate simulation in the objective analysis but have some form of optimization or search algorithm implemented into the system. The second category includes those systems which are simulation-based but do not include any type of simulation output processor or optimization routine. The third and final category involves those systems which are simulation-based and do include an output processor or optimization routine. Research revealed that this latter type of decision support system is the most common.

**Category I Systems**

Four type one decision support systems were found in the literature. The systems found varied widely in application, purpose, and scope. The first system involved applications in the food-processing industry. A computer production support system was constructed which utilized linear programming to select the most cost efficient allocation
of raw materials given a demand schedule for final products (Chan, Hui, and Sculli, 1991). This system also incorporated the use of a management information system which was used to update the parameters of the linear programming model. Another non-simulation-based decision support system discovered was one developed for a PC-board assembly plant at Texas Instruments in Austin. This system, known as INSITES (Integrated Scheduling, Inventory, and Throughput Evaluation System), uses the inclusion of intelligent heuristics to generate a set of high quality schedules which are designed to meet current multi-objectives of management. These schedules are generated through the use of GRASP. This tool, developed by the authors presenting the INSITES system, works in two phases. In phase one a greedy function is used to generate an initial feasible schedule. In the second phase a search is performed in the neighborhood of the initial solution for improvement and another schedule is constructed. Each schedule generated by GRASP is then evaluated according to user criteria. In order to support GRASP, the authors also developed INSITES. INSITES provides real-time monitoring of the production floor and, consequently, current input data to the schedule generator. In this system, the final decision of which schedule to implement is up to the user who must incorporate the immediate needs of the company into the system (Feo, Bard, and Holland, 1993). Another decision support system addressed the construction of simulation metamodels (statistical models of a simulation process). This decision system, known as PCRSM is a computer-based system which incorporates the use of response surface methodology as a means of determining simulation input-output relationships (Meidt and
Bauer, 1992). The authors discuss the use of this system in an application involving the simulation of a personnel system for the Airforce. The fourth and final non-simulation-based decision support system involves the use of a variation of simulated annealing and a generalized cost function in an attempt to optimize the scheduling of a job shop (Musser, Dhingra, and Blankenship, 1993). The authors have implemented their scheduling method in software known as ABES (Annealing Based Experiment in Scheduling). This software is currently being used at a Texas Instruments PC board manufacturing facility in Johnson City, Tennessee.

**Category II Systems**

The second type of decision support system to be discussed involves those systems which are simulation based but rely heavily on the use of interactive and iterative sessions with the user of the system in order to narrow in on a final decision. In other words, these systems offer the easy use of simulation as an analysis tool, but they do not offer any type of implemented algorithms or heuristics for guiding the decision maker. Two of these systems were found during research. The first system involves the use of a database and a simulation model (Norman and Norman, 1986). The database holds information concerning resources, part types, various scheduling rules, and task selection rules. The simulation model is fed by information from the data base and on-line data collection system on the production floor. Once the relevant information is fed into the simulation model, the user inputs varying scenarios into the system with the aid of the database and a
user-interface and then evaluates these alternatives using output generated by the simulation. This system allows the user to look at the tradeoffs of work-in-process inventory levels, utilization, cycle time, etc. In an iterative manner, the user is able to develop a suitable schedule which accomplishes goals that he or she has in mind. In addition to everyday scheduling of production, the user is able to perform long-term planning in the form of capacity analysis. Capacity analysis is used to point out any potential bottlenecks in the system.

In another application, the use of a simulation-based decision support system is presented by Aggarwal (1990). This system is used as a means of developing a schedule system for the credit department of a retail store. Aggarwal recognized the importance of simulation in being able to model situations which do not lend themselves to traditional analytical techniques. As a result he developed a model-centered decision support system for analyzing business systems. In his system, a user interface is provided which allows updating of system parameters and facilitates what-if analysis. Additionally, the output generated from simulation runs is presented to the reader in both a text and graphical format for easy interpretation.

Category III Systems

The third classification of decision support systems includes those systems which are simulation-based and offer some sort of output processing as an aid to the decision maker. As stated before, most decision support systems which have been developed are of
this type. One such decision support system was constructed by Shahraray and Maeschke (1990). Their system, known as SBDSS (a Simulation Decision Support System) incorporates the use of simulation, DSRs (Dynamic Scheduling Rules), and AHP (Analytical Hierarchy Process) in arriving at an optimal decision. AHP is used to form a hierarchy of goals and subgoals of the company in making a decision. DSRs are used to generate alternative scheduling strategies based on the decision maker's criteria, and the simulation model is used to test an appropriate set of alternatives generated by the system. Once the simulation runs are complete, the user is queried to assess the relative importance of criteria and a hierarchy of goals is formed using AHP as a basis. Once the hierarchy has been constructed, a MDCM (Multiple Criteria Decision Making) module is used to recommend the alternative which best meets the decision maker's objective.

The combination of an optimization algorithm and continuous process simulation is the basis of a decision support system developed by Bui and Ouellet (1993). Their system, while requiring no special programming skills or mathematical modeling of the user, allows the user to optimally control the process of continuously annealing an aluminum sheet and an aluminum casting furnace. Instead of constructing and cranking through mathematical models which may or may not be able to handle complexities of the process, a simulation model is used to capture system complexity, and the optimization algorithm is used to solve an appropriate optimization problem.

The application of artificial intelligence and simulation in a decision support system for obtaining Total Capacity Management (TCM) is presented by Pritsker and Yancey
(1991). In their paper, Pritsker and Yancey define a total capacity management plan which is based on simulation. They describe two applications of their system in which an intelligent model-based system, known as FACTOR, is used to develop schedules for production. With the construction of model-based schedules, total capacity management is realized through the determination of better operational policies for long-term planning, for better short term capacity planning, for better logistics scheduling, and for better short-term production scheduling. The two applications discussed by Pritsker and Yancey are compressor blade manufacturing area at Pratt and Whitney and the production of machined steel products at BethForge.

A framework for developing a goal-directed simulation system is offered by Shannon and Prakash (1990). This system involves the use of an intelligent back-end to aid the decision maker in analyzing simulation output. The intelligent back-end contains both domain specific knowledge and analytical search techniques. The domain knowledge is used to reduce the number of possibilities of decision variables to meet the desired objectives, and the search is used to find the best solution in the feasible region defined by the domain knowledge. The domain knowledge is represented in the form of a rule base and response surface methodology is used to find the optimal value of a defined cost function.

Another system which uses artificial intelligence has been proposed by Chaturvedi, Hutchinson, and Nazareth (1991). The goal of this system is to optimize the scheduling of a flexible manufacturing system (FMS). Their approach includes the integration of a
knowledge base which is updated through on-line machine learning. As simulation runs produce output for a given schedule and situation, the schedule is evaluated by a LISP/GoldWorks based program. This program categorizes and formulates new knowledge for the knowledge base. In this way, meaningful concepts concerning prior scheduling strategies and their outcomes can be used for comparison in order to obtain the overall goals of the company. Additionally the top-level goal to be achieved is seen as an aggregation of subgoals in a hierarchy of objectives. These objectives, or lowest level goals, are satisfied by certain levels seen in the simulation output. By iterating through the simulation and machine learning cycle, the ultimate goal(s) of the company can be accomplished.

A system for optimizing waste management capacity planning is offered by Baetz (1991). In this system simulation does not play the main role. Instead a dynamic programming model is formed. This model is used to find the best overall plan for the regeneration level and timing of a landfill facility, and the concurrent expansion of a waste-to-energy conversion facility. The dynamic program seeks to maximize utilization of both types of facilities. In executing the dynamic program, an embedded linear program is used. This linear program is solved to find the costs associated with each expansion evaluated as an alternative by one particular iteration of the dynamic program. However, the time increment used for study with the dynamic program is one year. To better understand the fluctuations or variations which may occur during a single year, a simulation model was constructed. By integrating the simulation model, linear program,
and dynamic program, a comprehensive study of the waste management problem could be adequately performed.

The construction of a manufacturing-oriented simulation software is proposed by Umeda (1992). This system will have an easy to learn language, a report generating system, and will include a model generator, a graphics post processor, and a schedule evaluator to aid the decision maker. This system is proposed as a means of performing both short term scheduling and long-term planning for any manufacturing-oriented system. The author's proposed system (just the core simulation software at the time of the writing of this paper), was applied to a production cell which contained more than twenty automatic machines and a hybrid push-pull product flow system.

The listing and descriptions of the systems presented here are by no means exhaustive, but they do give a good idea as to what types of approaches have been taken in designing decision support systems. The next section describes a form of AI which is rapidly becoming popular and may aid in the development of decision making tools.

In relation to the topic of this thesis, two articles were found which attempted to use artificial neural networks to learn simulation input-output relationships. The first article addresses the application of neural networks in designing a manufacturing system (Chryssolouris et. al., 1990). Specifically, the authors are attempting to determine an appropriate number of resources to assign to each of three work centers in a manufacturing system. The system being modeled was simulated assuming a certain workload, a certain number of workcenters in the system, and specific rules by which tasks
are assigned to resources. Under the assumptions made, the authors proved the applicability of neural networks in learning the reverse of the simulation process. In other words, given a set of predefined, desired levels of performance measures, they were able to determine appropriate levels of decision variables (in this case, the number of resources at each of three work centers). In another work published, the use of simulation and neural networks is used to determine operational policies for a manufacturing system (Chryssolouris, Lee, and Domroese, 1991). Operational policies dictate how tasks are to be carried out. However, to determine the appropriate operational policy for a manufacturer, decisions must be made concerning the relative importance of decision criteria in order to obtain certain levels of manufacturing performance. In this paper, the authors use simulation to generate data which is fed into a neural network for training. In doing this the relationship between decision making criteria and related performance measures are mapped out. As a result of this mapping, which is quite difficult, if not impossible to do by analytical means, proper weightings of decision criteria can be found such that decisions made at the local task level allow the overall goals of the system to be achieved.

The literature found during research suggests that any method which attempts to learn simulation input-output relationships requires much work on the part of the analyst. The analyst must perform a number of iterations using any of the methods listed thus far. What is needed is a means by which the analyst can set up a single experiment and execute it such that information needed to make decisions can be obtained from the single
experiment. In this manner, minimal effort would be required on the part of the analyst.
Perhaps one method of accomplishing such a task involves the use of artificial neural networks. If a single experiment could be setup and ran such that a neural network is able to map the relationships between multiple simulation inputs and outputs, then the effort required of the analyst is reduced by a significantly large amount.
CHAPTER THREE
SIMULATING THE TEST FLOOR

This chapter introduces the reader to the test floor operations which are supported by the Decision Support System. The simulation model which is at the heart of the system is also described briefly in this chapter with a detailed description being given in appendices one and two. The chapter is divided into three sections. The first section explains the scope of the simulation model and provides the reader with an idea of the type questions the simulation model is designed to answer. The second section describes the general flow of products through the test area. The third section discusses the actual simulation of the test floor.

Scope of the Simulation

When work on the simulation model was begun, it was most important to realize the scope and the purpose of the model. In addressing the questions that management was posing, it was considered necessary to include logic which addressed the following issues:

What effect does adding or deleting any type of resource have on the test floor
How does a certain demand affect test floor performance
How does a certain product mix affect test floor performance
What is the effect of changing the work shifts

33
How does different levels of rework affect test floor performance

What effect do different scheduling strategies have on test floor performance

Performance of the test floor was determined to be based on the following metrics:

Utilization of resources
Cycle time of lots through the test area
Throughput of lots in test area
Work-in-process inventory levels

**Modeling the Test Floor**

Keeping the above issues in mind, the modeling process was begun. The first order of business was to become familiar with the test floor and its operations. To accomplish this task, interviews were conducted with the test floor supervisors, process analysts, and test floor operators. In addition, much time was spent on the test floor to gain additional insight into how the test floor actually operated. Once an adequate understanding of operations was gained, a diagram showing the test floor layout and product flow was constructed as is shown in figures 1 and 2 respectively. A complete description of each processing step required is given in Appendix 1.

Semi-conductor products start in the manufacturing cleanroom as lots of fifty wafers. Each wafer contains any number of dies, typically around 100 to 600 dies per wafer. These lots of wafers, upon completion of manufacturing, arrive at the "Apply Tape" process step in the test area. Once in the test area, the wafers of each lot are loaded
Figure 1 -- Test Floor Layout
Figure 2 -- Test Floor Product Flow
onto an automatic tape application machine and tape is applied to the back of each wafer.

Next, the wafers are sent to the back grind machines where each wafer is ground to a specified thickness. Upon completion, the wafers are put through an automatic rinser/dryer cycle to remove contaminants. After being cleaned, the wafers are loaded onto an automatic detaping machine which removes the tape from the back of each wafer. The wafers are then rinsed and dried again in a long cycle automatic rinser / dryer. All of these steps are performed in the BackGrind area as shown in Figure 1. After all operations in the Back Grind area are complete, the lot is sent to the TVS process step and/or the IV process step for diagnostic tests. After these steps, the lot of wafers is sent to the Probe (optical test) process step so that each individual die on every wafer can be visually inspected. After the probe step, the lot of wafers is sent to in-process inspect or to off-line ink. The off-line ink step is required to visually mark bad dies. (Only certain products require this step.) If off-line ink is required, the lot of wafers is sent to be baked, cooled, and rinse/dried again, and it is sent to in-process inspect. The wafers are then added to finished goods inventory and stored until the end of the week when shipments are made to customers. In addition to the steps listed here, there are also rework steps which may require a certain process to be done more than once. Furthermore, there may be hold steps which are required for evaluating a certain quality problem. The general flow of lots, including rework and holds can be seen Figure 2.

Using the product flow and taking into consideration all information gained thus far, programming of the simulation model was begun. In an incremental fashion, each of
the processing steps as detailed in appendix one and shown in figures 1 and 2 were modeled and verified. Additionally, data was gathered on an as-needed basis from one of several sources. These sources included data from a company databases, off-line reports not supported by company databases, estimates of the test floor personnel, and direct observations of operations.

**Simulating the Test Floor**

The actual simulation program was written is SIMAN (Pegden, 1990) general purpose simulation language. The listings for the two required files for the simulation model are included in appendix four. The two required files for the SIMAN simulation language are the model file and the experimental file. The model file contains all relevant modeling logic of the system being simulated. The experimental file contains all the relevant parametric data needed to run the simulation. Once an appropriate model file and experimental file have been created for the simulation scenario under study, the SIMAN program is called to compile and link the two separate files into one executable simulation program. This program is then executed and simulation output is generated. Simulation outputs include such metrics as utilization, cycle times, WIP, throughput, etc.

Upon completion of the modeling and initial trial runs, statistical analyses were performed to ensure that predictions for operational metrics (performance measures) were within 10% of the predicted value from the simulation. Additionally, validation was conducted by comparing simulation output with data obtained from the company database
whenever this was possible. When data was not available for comparison, test floor personnel offered assistance with validation. All of the relative assumptions made, the processing times used, product flow routings incorporated into the model, rework routings incorporated into the model, and all system related information that was incorporated into the simulation model are detailed in Appendix two. Appendix Three details the relevant statistical analysis used to ensure that the simulation output would be within the desired 10% range of the predicted mean. The next 2 chapters discuss neural networks and how they were constructed and trained to learn the input-output relationships of the simulation model.
CHAPTER FOUR
BACKGROUND ON NEURAL NETWORKS

In order to provide the reader with an understanding of the research being presented on artificial neural networks, a general overview of this form of artificial intelligence is presented in this chapter. The chapter is divided into two sections. These sections are:

Artificial Neural Networks Overview and Description
Training of Neural Networks

Overview and Description

Artificial neural networks stem from early research aimed at representing the activity of the human brain. As researches have noted, the brain is composed of a very large number of interconnected neurons or cells. Each one of these cells is responsible for a simple activity. Together these simple activities enable the brain to learn and process stimuli. In the same manner, artificial neural networks are composed of many small processors, called neurons, which are responsible for simple computations with numerical data. Weighted connections between these processors, called arcs, determine the effect that one neuron has on another neuron. By presenting stimuli, or numerical input data, to the network and activating the network, information can be propagated
through the neurons and their connecting arcs until output data is obtained. Furthermore, neural networks have the ability to learn relationships between given sets of data. When presented with sets of input and output pairings contained in a training set, the network is able to develop relationships by simply changing the weights of its interconnections. Additionally, when presented with data other than the data used to train the network, the network has the ability to generalize. To generalize means the network can generate a set of outputs from a set of inputs even though that set of inputs was not used in the training data set. This is the great advantage that neural networks have over other types of data processing. With this type of processing, no details of the logic or innerworkings of a system are needed, there is no need to explicitly define any rules of operation, and there is no need to model whatsoever. The network has the ability to learn the relationship straight from examples. The main difficulty faced, however, in any application of neural networks deals with how to configure the network for the problem at hand. Any number of neurons, layers of neurons, type and degree of interconnections, individual computational functions of neurons, weighting schemes, and learning algorithms can be used when constructing a neural net. For certain types of applications, a specific type of neural net configuration will work quite well, but it may fail miserably when applied to another type of problem.

Application areas of neural nets include real-time process control, optical character recognition, language translation, and other areas in which a well defined method is either not available or not easily understood. Some specific applications of
neural networks that have been detailed in research include the controlling of a galvanized steel manufacturing process (Schmidt, Haddock, and Wallace, 1993), power prediction in a nuclear power plant (Cheon and Chang, 1990), recognition of radar targets by electronic and optical computers (Farhat, 1986), automobile engine diagnosis (Marko, 1989), economic prediction of the stock market (White, 1989), medical diagnosis (Bounds and Lloyd, 1989), and Robot Control (Sobajic, 1989). Other applications of neural networks involve the fitting of nonlinear curves to data sets (Bishop and Roach, 1992), the development of an add on tool for learning data in a spreadsheet program such as Lotus 1-2-3 for windows or Microsoft Excel for windows (Caudill, 1993), planning of end user involvement in developing information systems (Lodewyck and Deng, 1993), and various forecasting applications including bankruptcy prediction (Fletcher and Goss, 1993).

As stated, artificial neural networks have simple processing units or neurons which are responsible for performing a simple numerical calculation on input so that an appropriate output is found. The net input to a neuron or node is either direct input from a keyboard, file, machine, or other component, or is the output from other neurons. The interconnections which exist between nodes (neurons) are the means by which output from one node is propagated to other nodes. With a group of these interconnected simple processing units, a network of neurons is formed. Additionally, most networks consists of layers of neurons. In most all networks, the interconnections between neurons are present only between adjacent layers of neurons. A typical network is shown in figure 3. This figure shows a network consisting of three layers of neurons. Connections exist
Figure 3 -- Neural Network Schematic View
between the left most layer and the middle layer and the middle layer and the right most layer. The left most layer is known as the input layer. Inputs to the neural network occur through the neurons in this layer. Through the use of a simple transfer function, numerical input is transformed by these nodes into an appropriate output which is then propagated to the middle layer of neurons. This middle layer of neurons is known as the hidden layer. Each node in the hidden layer receives input via its connections from all the nodes in the input layer. The total amount of input to each of the nodes in the hidden layer is a function of the output from all nodes from which it receives input and the weighted links which connect the node with nodes in the previous (input) layer. The total input into a hidden layer node is then transformed into an output which is then propagated to the next layer in the network. The next layer (right most) is known as the output layer. The neurons in this layer perform appropriate transformations on input received from the hidden layer, and the result is the net output of the network. In the network shown in Figure 3, the network has 4 nodes which make up the input layer and 5 nodes which make up the output layer. This means that the network is able to receive 4 input values (one form each node) and transform these inputs into 5 output values. Thus neural networks are able to map multiple inputs into multiple outputs. This is true as long as the appropriate weights are attached to the interconnections which exist between neurons. The ability to find the correct weights is a process called training which is discussed in the next section.
Training Neural Networks

In order for a neural network to be of any use, it must be able to appropriately map a set of inputs into a set of outputs. As stated in the previous section, the means by which a network does this is through the determination of interconnection weights. The process of determining these weights is known as "training" the network. There are two main methods of training neural networks. These are supervised and unsupervised training. The method used in this research and the one being described here is the former of the two. With supervised training, a number of input-output vector pairs are introduced to the network. These input and output vectors are known as training pairs. The values contained in these vectors come directly from the process which the network is trying to learn. The training pairs which are presented to the network represent a subset of all the possible training pairs which could possibly be generated by the process being learned. By using a subset of all possible values, a neural network is able to learn not only the relationship which exists between the input and output in the training pairs, but it is also able to learn the general relationship between any input and output which may be indicative of the process under study. This latter statement is one of the very powerful realizations which comes from using neural network technology. The ability to perform in this manner is known as generalization. Once trained, a neural network is able to generalize or predict appropriate output for input which it has never seen. The actual training process is described in the following paragraphs.
In order to train a network, each vector of inputs is propagated through the network and a vector of output values is obtained. The following equations help to describe how output is determined by a node in the network. Input is given by:

\[ h_i = \sum_{k=1}^{n} w_{ik} \zeta_k \]

where net input to a node is denoted by \( h_i \) and represents the sum of all inputs to node \( i \) from all connecting nodes, \( w_{ik} \) is the interconnection weight between \( i \) and node \( k \), and \( \zeta_k \) is the net output of node \( k \). Output is given by:

\[ o_i = g(h_i) \]

where output is denoted by \( o_i \) and \( g(h_i) \) is a nonlinear differentiable activation function for node \( i \). A quite common transfer function used in neural networks is the sigmoid function. This graph of this function is shown in figure 4.

![Figure 4 Sigmoid Transfer Function](image-url)
When training, the output values for all nodes are compared with the desired output contained in the training pair. In doing so, an error is computed based on the difference between the desired output and the actual output. The error computed for the entire network takes the form of the following equation:

\[ E(w) = \sum_{i=1}^{n} \frac{1}{2} [t_i - o_i]^2 \]

In this equation \( E(w) \) is the total network error with respect to the vector of weights currently present in network, \( t_i \) is the target output value for node \( i \), and \( o_i \) is the actual output value for node \( i \). (The value of \( \frac{1}{2} \) used in this equation is simply a constant which has been determined by earlier researches to robust for most back-propagation neural networks.) Once this error is computed, it is propagated backwards through the network and the weights which are attached to the links between the neurons are changed. The change made is done such that the overall error experienced by the network is reduced. The change that is made to each of the weights is, in general, proportionate to the contribution that weight made to the overall network error. In other words the following relationship holds:

\[ \Delta w_{ik} = -\eta \frac{\delta E}{\delta w_{ik}} \]

In this equation \( \frac{\delta E}{\delta w_{ik}} \) is the portion of the overall network error due to the interconnection weight between node \( i \) and node \( k \), and \(-\eta\) is a constant used to obtain a change in the connection weight which is a multiple of the portion of the error attributable to and reduces the overall error. (i.e. A move in the opposite direction of the error function's gradient is desired.) The actual expression for changing weights is given by...
\[ \Delta w_{ik} = \eta \delta_{ik} \zeta_k \]

where \( \eta \) is again a constant, \( \zeta_k \) is the input from all nodes in layer \( k \) to node \( i \), and is given by...

\[ \delta_i = [\tau_i - o_i]g'(h_i) \]

where \( \tau_i \) is the target output from node \( i \), \( o_i \) is the actual output from node \( i \), and \( g(h_i) \) is the derivative of the transfer function of node \( i \) evaluated at \( h_i \) (net input). By iteratively propagating input and output vectors forwards and backwards through the network, and calculating weight changes, the neural network learns the appropriate mapping between inputs and outputs.

In short, the training process is reduced to a search problem. This search attempts to find weights for the network which find a global minimum of an error surface. This error surface is a function of the particular network architecture (i.e. number of nodes, layers, transfer functions, etc.) and the particular system which the network is trying to learn. To begin the learning process, a set of random weights is assigned to all the links between nodes. After initial weights are set, a forward pass of input is made and an error is calculated. Appropriate changes in each of the network's weights are then calculated as a backwards propagation of the error is performed. This change is a function of the learning algorithm being used to train the network. The speed and accuracy with which a network is able to learn a process is directly dependent on this algorithm. Choosing the appropriate algorithm is at best a trial and error process. However, having an
understanding of neural networks, knowing how they work, and acquiring experience in training them can greatly reduce the time needed to find an appropriate algorithm. The learning algorithm used for the application presented in this thesis is known as the back-propagation learning algorithm. The widespread use and excellent understanding of this type of learning is the reason for choosing this method.

The use of neural networks in learning the simulation model of the test area is the focus of the experimentation presented in Chapter Five. The terms used, the methods employed, and the course taken in doing the research all stem from the basic understanding of neural networks -- knowledge which the current chapter has attempted to convey to the reader.
CHAPTER FIVE

EXPERIMENTING WITH NEURAL NETWORKS

The purpose of the experiments which are discussed in this chapter is to explore the applicability of neural networks in the area of semiconductor test operations. The ability of neural network technology to learn the reverse mapping of the simulation model is the reason for the experimentation. With substantial proof that a neural network can learn the relationship between user inputs and simulation output for the test floor of AT&T, a powerful tool can be placed in the hands of the decision maker.

Methodology -- Preparing for Experimentation

Once the simulation model of the test floor (as discussed in chapter three) was constructed, verified, and validated, the next logical step was to perform the experimentation with neural networks. An appropriate software package was found for computer simulation of the artificial neural networks. This package known as XERION is a versatile neural network simulator which was developed by Drew van Camp, Tony Plate, and Geoffrey Hinton at the University of Toronto. This package allows the user to design, train, test, and use a host of different types of neural networks. These networks differ in architecture (quantity of neurons, connection schemes between neurons, number of layers, etc.), learning algorithms, and types of training paradigms. The software offers different modules which can be invoked to allow the user to test the different training
paradigms and learning algorithms. Some modules are already present when the simulator is installed. Other modules can be developed to run with XERION using the guidelines in the help manuals provided by the authors. For the experimentation presented here, it was decided to use the back-propagation module. This module, already implemented into the XERION package, employs the widely accepted and well understood back-propagation learning algorithm.

Once the back-propagation module was chosen for experimentation, some familiarization with the software and this particular training paradigm had to occur. Specifically, the ability to create, train, and test a neural network had to be learned. The particular code used to create a network can be seen in Figure 5.

```
addNet "k55_1"
useNet "k55_1"
addGroup -type INPUT input 30
addGroup hidden 26
addGroup -type OUTPUT output 13
connectGroups input hidden
connect Groups hidden output
addExamples -type TRAINING k55_1.ex randomize 1.0
```

Figure 5  XERION Code Used to Create a Neural Network

This code shows that the network designed has 30 input nodes, 26 hidden layer nodes, and 13 output nodes. Additionally, this code fully connects the nodes between adjacent layers,
assigns random weights to these connections, and enters an appropriate file which contains input-output pairs for training. A schematic drawing showing the conceptual view of this network is shown in figure 6. For the experimentation performed, this particular architecture was eventually chosen. This architecture, allows 5 inputs to be transformed into 4 outputs.

The reader may wonder why a network with 30 input nodes and 13 output nodes is used for learning the relationship between 5 inputs and 4 outputs. The explanation for this lies in the understanding of how a neural network learns and the limits which must be placed on the network. A neural network, in most instances, is used to learn processes in which a binary representation is used. That is, a node either has a value of 1 or 0 which corresponds to a true or false answer. Additionally, training neural networks to learn continuous value to continuous value mappings is quite a bit more difficult than training them to learn binary to binary mappings. Also the transfer function used by the neurons in the network is the sigmoid transfer function which has as its range \((0,1)\) (See Figure 4 in Chapter Four). Thus a binary representation seems most intuitive when dealing with neural networks. However, the problem at hand still requires the transformation of continuous input values (such as cycle time, work-in-process inventory levels, utilization, etc.) into discrete output values (such as quantity of resources, queuing strategies, etc.) Note that the discrete output values mentioned here are actually input values to the test floor simulation model. Likewise the continuous input values to the neural net are the outputs from the simulation model. In order to represent these values in the neural
Output Layer
Nodes
13 in all
each having a value between 0 and 1

FULLY CONNECTED LAYERS (every node in hidden layer is connected to every node in output layer)

Hidden Layer
Nodes
26 in all

FULLY CONNECTED LAYERS (every node in input layer is connected to every node in hidden layer)

30 Input Layer nodes
each having a value of 0 or 1

Figure 6 -- Neural Network Architecture as Detailed by XERION Code in Figure 5
network it was decided to define the continuous input values as belonging to certain ranges. In this manner, if an input value falls into one of several defined ranges, the node of the network which represents that particular range for that particular input value has a value of 1 or true. All other nodes which represent that input variable will have a value of 0 or false. In a similar manner the discrete output values are also represented in the neural network. For example, if three possible quantities of a resource are possible, there will be three nodes in the neural network which represent the quantity of this resource. Specifically if 1, 2, or 3 testers of Type B are possible, and the chosen quantity is 2, then the first and third nodes will have a value of 0 or false, and the second node will have a value of 1 or true. By coding the inputs and outputs in this manner, a binary representation of the simulation input and output can be understood and processed by the neural network. The specific ranges and coding scheme used in the experimentation for the 5 input/4 output problem are shown in Table 1 and Table 2.

These tables show that 6 ranges were defined for each of the continuous simulation outputs (neural network inputs), and 3 to 4 ranges were defined for each of the simulation inputs (neural network outputs). The ranges as defined in this table were chosen in order to obtain an accurate mapping of input-output pairings while still allowing for the range of all possible values to be accounted for. However, before ranges could be explicitly defined, the actual ranges for the simulation output had to be understood. For this reason, output from some initial simulation runs were studied.
### Table 1  Simulation Outputs and Their Binary Representation

<table>
<thead>
<tr>
<th>TYPE OF OUTPUT</th>
<th>RANGE</th>
<th>BINARY REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle Time</td>
<td>&lt; 1600</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>1600 - 1700</td>
<td>0 1 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>1700 - 1800</td>
<td>0 0 1 0 0 0</td>
</tr>
<tr>
<td></td>
<td>1800 - 1900</td>
<td>0 0 0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>1900 - 2000</td>
<td>0 0 0 0 1 0</td>
</tr>
<tr>
<td></td>
<td>&gt; 2000</td>
<td>0 0 0 0 0 1</td>
</tr>
<tr>
<td>Work-in-Process Inventory</td>
<td>&lt; 700</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td>(wafer quantities)</td>
<td>700 - 800</td>
<td>0 1 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>800 - 900</td>
<td>0 0 1 0 0 0</td>
</tr>
<tr>
<td></td>
<td>900 - 1000</td>
<td>0 0 0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>1000 - 1100</td>
<td>0 0 0 0 1 0</td>
</tr>
<tr>
<td></td>
<td>&gt; 1100</td>
<td>0 0 0 0 0 1</td>
</tr>
<tr>
<td>Tester Utilizations</td>
<td>&lt; 10%</td>
<td>1 0 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>10% - 30%</td>
<td>0 1 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>30% - 50%</td>
<td>0 0 1 0 0 0</td>
</tr>
<tr>
<td></td>
<td>50% - 70%</td>
<td>0 0 0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>70% - 90%</td>
<td>0 0 0 0 1 0</td>
</tr>
<tr>
<td></td>
<td>&gt; 90%</td>
<td>0 0 0 0 0 1</td>
</tr>
</tbody>
</table>

### Table 2  Simulation Inputs and Their Binary Representation

<table>
<thead>
<tr>
<th>TYPE OF INPUT</th>
<th>VALUE</th>
<th>BINARY REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUING STRATEGY</td>
<td>1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>QUANTITY OF TESTERS</td>
<td>1</td>
<td>1 0 0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0 1 0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

**NOTE:** queuing strategies are (1) first in / first out (2) shortest processing time per lot (3) highest demand, and (4) lowest slack time for lot. Also values given for cycle time are unitless for reasons of disclosure.

*Figure 5-4 Simulation Inputs and Outputs and their Binary Representation*
In order to do some initial experimentation, some training data had to be available.

In order to generate training data, several test floor simulations had to be ran. Additionally, input to the simulation model had to be varied for each run so that a range of simulation input and output pairs could be generated. To accomplish this task, a UNIX shell script was written which randomly varied the inputs to the simulation model, ran the simulation model, and gathered the output. The inputs that were varied included the quantity of each type of Tester and the queuing strategy used to process lots of wafers through the test floor. Table 3 shows the input variables which were varied and the possible values which could be inputted to the simulation.

Table 3  Values for Simulation Inputs When Generating Training Pairs
Queuing strategies used are (1) FIFO (2) shortest processing time (3) highest demand, and (4) lowest slack value

<table>
<thead>
<tr>
<th>INPUT VARIED</th>
<th>POSSIBLE VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queuing Strategy</td>
<td>1, 2, 3, or 4</td>
</tr>
<tr>
<td>Qty of Type A-1 Testers</td>
<td>1, 2, or 3</td>
</tr>
<tr>
<td>Qty of Type A-2 Testers</td>
<td>1, 2, or 3</td>
</tr>
<tr>
<td>Qty of Type A-3 Testers</td>
<td>1, 2, or 3</td>
</tr>
<tr>
<td>Qty of Type B Testers</td>
<td>1, 2, or 3</td>
</tr>
<tr>
<td>Qty of Type C Testers</td>
<td>1, 2, or 3</td>
</tr>
</tbody>
</table>

Upon completion of the simulation runs, output was saved in flat text files for later processing. In order to allow for a wide range of input-output pairings, it was decided to
run the simulation a fairly large number of times. In fact, three hundred simulation runs were made by varying the inputs as shown in Table 3. It was believed that this number of runs would allow for an adequate number of training pairs to be available for the neural network. Additionally, mappings of various combinations of input-output pairings could be tested with this many runs.

**Methodology -- Final Experimentation**

After the three hundred simulation runs were made, experiments were begun to explore the feasibility of the use of neural networks. Several different input-output pairings were tried to see if a neural network could learn the reverse of the test floor simulation model. For example, the mapping of work-in-process inventory and cycle time to queuing strategy and quantity of type A-1, A-2, and A-3 testers was tried. The conceptual view of this neural network structure is seen in Figure 7. Other similar mappings were explored by varying the number of input variables and the number of output variables. However, in all cases, the levels of the variables (i.e. ranges defined by each node in the input or output layer) were not varied. That is each simulation output was represented by a collection of 6 nodes, each quantity of each tester type was defined by 3 nodes, and the queuing strategy used was defined by 4 nodes. Finally, an adequate size problem was found which included the mapping of 5 simulation outputs to 4 simulation inputs. The schematic of this neural network is shown in figure 8.
Figure 7 -- Neural Network Architecture for a 2 Output to 4 Input Mapping
Figure 8 -- Neural Network Architecture Used for Experimentation
In order for the neural network to learn the relationship between the inputs and outputs, appropriate simulation runs had to be chosen. In the initial three hundred simulation runs, the level of 6 input variables were varied. These inputs were queuing strategy and quantity of the 5 different types of optical testers used. However, the neural network was trying to learn the relationship between a given set of output and 4 input variables. To alleviate this problem, a subset of the three hundred simulation runs had to be chosen. This subset was chosen such that the 2 input variables (quantity of two of the tester types used) that were not included in the neural network input layer were held constant while the other 4 input variables were varied. In this manner the neural network would be able to associate variations in simulation output with the variation in the 4 simulation inputs.

Once an appropriate subset of input-output pairs was chosen from the initial 300 runs, both the input and output were preprocessed so that the neural network could interpret the data. This preprocessing (changing to a binary representation) was done as explained in the previous section of this chapter. A UNIX shell script was employed to preprocess the simulation inputs and outputs into the required binary format. The conversion of an example training pair can be seen in Tables 4 and 5. Through an iterative process, all training pairs from the simulation runs were converted to this format, and the resulting file was used as input to the neural network.

Even though the inputs and output were ready to be loaded into the neural network, some other preparational work had to be done with the neural network
architecture itself. Specifically, the various parameters used in training the network had to be explored. Even though it was decided to use the back-propagation learning paradigm, the specific search technique used to reduce network error and change the interconnection weights had to be chosen. Additionally, with the search technique chosen, it was necessary to determine the proper levels of the parameters in order to ensure the highest level of accuracy possible while reducing the time needed by the search technique to learn the data in the training set. Furthermore, even though the number of input and output

<table>
<thead>
<tr>
<th>Type of Input</th>
<th>Value</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queuing Strategy</td>
<td>3</td>
<td>0010</td>
</tr>
<tr>
<td>Qty of Type A-1 Testers</td>
<td>3</td>
<td>001</td>
</tr>
<tr>
<td>Qty of Type A-2 Testers</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Qty of Type A-3 Testers</td>
<td>2</td>
<td>010</td>
</tr>
</tbody>
</table>

Table 4 Conversion of an Example Training Pair -- Inputs

<table>
<thead>
<tr>
<th>Type of Output</th>
<th>Value</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle Time</td>
<td>1,697</td>
<td>0100000</td>
</tr>
<tr>
<td>WIP</td>
<td>747</td>
<td>0100000</td>
</tr>
<tr>
<td>Type A-1 Testers' Utilization</td>
<td>0.61</td>
<td>00010000</td>
</tr>
<tr>
<td>Type A-2 Testers' Utilization</td>
<td>0.35</td>
<td>0010000</td>
</tr>
<tr>
<td>Type A-3 Testers' Utilization</td>
<td>0.25</td>
<td>0100000</td>
</tr>
</tbody>
</table>

Table 5 Conversion of an Example Training Pair
nodes were determined by the nature of the problem, the number of nodes in the middle or hidden layer of the network had to be decided upon. Determination of the afore-mentioned parameters was done by a trial and error approach. Several search techniques were tested, and varying numbers of hidden layer nodes were used. The levels of the parameters needed for the execution of the search techniques were also experimented with in a trial and error manner. The levels which consistently allowed the network to learn quickly were found and used in the final experimentation. The search technique found to be the fastest and most consistent in minimizing the network error was chosen for final experimentation as well. Also, the number of hidden nodes which allowed for the lowest error to be reached was determined after choosing the proper search technique. Once all pertinent network parameters were decided upon through experimentation, sample data was loaded into the network and the training process was begun.

Upon completion of the training process, the network had to be tested or verified for precision and accuracy. The final weights which were assigned to the connections in the network had to be tested in order to find the level to which the network had learned the reverse of the simulation process. Additionally, the ability of the neural network to generalize, or produce proper output from input data which was not in the training set, had to be evaluated. Initially, some of the training pairs used during the learning process were fed into the neural network. The ability of the network to learn these mappings was determined first in order to make sure that the network had indeed learned during the
training process. Afterwards, several groups of simulation output values not used during training were introduced and propagated through the trained network. The result of this activity was output from the neural network which showed the levels of simulation input needed in order to obtain the desired simulation output which was initially fed into the trained network. This "suggested" vector of input values was then fed back into the simulation model and ran. The actual output from the simulation run was then compared against the desired output fed into the neural network and some error measurements were made in order to assess the performance of the network. In order to gain some idea of the value of the neural network as a learning tool, some comparisons were made between the neural networks performance, some best guesses by personnel familiar with test floor operations, and some random guesses. Each of these methods had as their goal to suggest appropriate levels of simulation inputs to obtain the desired level of outputs. The next section discusses the details of the experimental results and offers some analysis of the results obtained.

Results of Experiment

The objective was to train the network. In doing this, the total network error was observed to never reach zero. The results of the training process is shown graphically in figure 9. This figure shows that the total network error decreased as the number of training iterations increased. The level part of the graph shows that the learning process slowed down quite considerably around the 40th iteration. Although the error never
Figure 9 -- Neural Network Error During Training
reached zero, this was to be expected. This later fact stems from the realization that multiple simulation inputs could be used to obtain the same levels of simulation output. In other words, the process which is being simulated does not have a one to one mapping (i.e. a convex function). However, it was decided to test the network with this level of error to see exactly how useful the trained neural network could be.

An initial vector of desired outputs was formed in order to begin the testing process. The levels of these outputs were determined using the range of output values from the initial three hundred simulation runs as a guideline. These desired outputs were fed as input to the trained network and suggested levels of simulation input were obtained. Figures 10 and 11 shows the schematic representation of the input-output mapping that was obtained for an initial test case. The actual values obtained for the neural network output are seen in figure 11. The reader should note that the values shown in the output nodes of the neural network in figure 11 are not exactly 0 or 1. Because of the networks inability to reduce the overall error to zero during training, the extreme values of 0 and 1 are not obtainable. In order to alleviate this problem, the node which contained an output greater than the output of the other nodes was chosen as the proper output. This method is a sort of "winner take all" approach. An example of this might be that the four nodes for queuing strategy show values of .1, .2, .9, and .1. The node with the value of .9 would be considered to be the winning node and the third queuing strategy would be the suggested value for this particular simulation input (network output). Likewise, the other output nodes in the network would use the same "winner take all" strategy.
Figure 10 -- Neural Network Architecture Used for Processing Desired Outputs
Figure 11 -- Neural Network Architecture as Detailed by XERION Code in Figure 5 and Used for Determining Proper Simulation Inputs
The desired simulation output used as input to the neural network for this initial test case is shown below in Table 6:

Table 6 Desired Simulation Outputs for initial test case

<table>
<thead>
<tr>
<th>SIMULATION OUTPUT</th>
<th>Desired Range</th>
<th>Value Used in Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIP</td>
<td>less than 700</td>
<td>650</td>
</tr>
<tr>
<td>Cycle Time</td>
<td>1600 to 1700</td>
<td>1,650</td>
</tr>
<tr>
<td>A-1 Testers Utilization</td>
<td>70% to 90%</td>
<td>80%</td>
</tr>
<tr>
<td>A-2 Testers Utilization</td>
<td>30% to 50%</td>
<td>40%</td>
</tr>
<tr>
<td>A-3 Testers Utilization</td>
<td>10% to 30%</td>
<td>20%</td>
</tr>
</tbody>
</table>

The reader should also note that the desired values represent the midpoint of the range for each of the possible 6 ranges defined by the neural network. For instance, the desired cycle time of 1650 represents the midpoint of the second range of cycle times. These desired values were fed into the trained network and the suggested levels of input shown in Table 7 were obtained.

Table 7 Simulation Inputs Suggested by the Neural Network

<table>
<thead>
<tr>
<th>Simulation Input</th>
<th>Suggested Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queuing Strategy</td>
<td>3</td>
</tr>
<tr>
<td>Qty of A-1 Testers</td>
<td>1</td>
</tr>
<tr>
<td>Qty of A-2 Testers</td>
<td>1</td>
</tr>
<tr>
<td>Qty of A-3 Testers</td>
<td>3</td>
</tr>
</tbody>
</table>
Using these values as input to the simulation, the test floor simulation was reran and new output was obtained. The comparison of the desired output versus the simulation output is shown in Table 8. Both the binary representation and the continuous value representation are included. In addition to the neural network suggested input, some "best guesses" and random guesses were made at the appropriate level of simulation input to obtain the desired simulation output. Also, an error was computed in an attempt to quantify the amount by which these suggested inputs failed to meet the desired output.

**Table 8  Comparison of Actual Simulation Output and Desired Simulation Output**

<table>
<thead>
<tr>
<th>Simulation Output</th>
<th>Value Desired</th>
<th>Actual Value</th>
<th>Binary Form Desired</th>
<th>Actual Binary Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle Time</td>
<td>1,650</td>
<td>1,604</td>
<td>010000</td>
<td>010000</td>
</tr>
<tr>
<td>A-1 Testers Util.</td>
<td>80%</td>
<td>71%</td>
<td>000010</td>
<td>000010</td>
</tr>
<tr>
<td>A-2 Testers Util.</td>
<td>40%</td>
<td>30%</td>
<td>001000</td>
<td>010000</td>
</tr>
<tr>
<td>A-3 Testers Util.</td>
<td>20%</td>
<td>30%</td>
<td>010000</td>
<td>010000</td>
</tr>
<tr>
<td>WIP</td>
<td>650</td>
<td>691</td>
<td>100000</td>
<td>100000</td>
</tr>
</tbody>
</table>

In computing this error, the goal was to weight all outputs at the same level. This means that missing the cycle time range by 1 network node (i.e. an adjacent node to the correct node was chosen to be the proper output node) or missing the work-in-process range by 1 network node would be weighted equally in computing the overall performance error for the suggested input. The error which was computed is defined as follows:
ERROR = \frac{\text{abs}[(\text{desired output} - \text{actual output})]}{\text{range}}

where n = \text{number of outputs}

and

range = \text{the span of values covered by a single network node for a given output}

This error, in effect, gives an estimate of the total number of nodes by which the method of selecting inputs was in error. The following example illustrates the calculation of the error associated with the neural networks suggested input:

\[
\text{ERROR} = \Sigma \left( \frac{\text{abs}[691 - 650]}{100} + \frac{\text{abs}[1650 - 1604]}{100} + \frac{\text{abs}[80 - 71]}{.20} \\
+ \frac{\text{abs}[40 - 30]}{.20} + \frac{\text{abs}[20 - 30]}{.20} \right)
\]

\[= 2.32\]

The comparison of the different methods for obtaining proper simulation input are shown in Table 9, with each method having its error computed. A total of 20 test cases were used in all. Error values were computed for these cases and the results are shown in Appendix five. It should be noted that although some desired output vectors may have a rather low error rating (5 or less), others do not. This stems from the fact that some levels of output may be infeasible for the system being simulated. In such a case no set of input values will allow the desired output to be obtained. The reader should keep in mind that an error rating of 5 or less is good. (An error of 5 suggests that the desired output was missed on average by 1 network node for each of the 5 outputs.) It should also be observed that the neural network performed best in 11 of the 20 test cases. Additionally,
Table 9  Comparison of Neural Networks and other Methods for Determining Simulation Inputs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>650</td>
<td>1,650</td>
<td>80</td>
<td>40</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>691</td>
<td>1,604</td>
<td>71</td>
<td>30</td>
<td>30</td>
<td>2.32</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1,713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>2.74</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1,823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>5.9</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1,879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>5.33</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1,807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>5.92</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1,611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>4.32</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1,757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>4.43</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1,713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>2.73</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1,958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>6.99</td>
</tr>
</tbody>
</table>
the times that the network did not perform the best, it did perform at a level which was a close second or third to the method with the best error rating.

In short, the ability of the neural network to learn the input-output relationship does show some potential. If neural networks can perform at a level which is better than a conventional trial and error approach then this technology is definitely worth exploring further. Furthermore, the ability to map multiple outputs into multiple inputs is a quite difficult task at best. Having a tool which decreases the computational burden on the analyst, suggests input levels in order to become sufficiently close to desired output levels, and requires little or no understanding on the part of the analyst is in itself very valuable. Using artificial neural networks in this manner, however, is not by any means a tool for direct optimization. The network used for experimentation in this research is simply a way of determining the most appropriate levels of input to obtain prespecified levels of output. By specifying the levels of output desired, the analyst is able to directly enter measures of performance into the neural network. No formal tradeoff analysis must be performed each time a different set of objectives is defined. As long as the neural network is designed so as to incorporate all the goals of the decision maker, analysis is a simple and straightforward process which requires little effort on the part of the analyst.
CHAPTER SIX
INTEGRATION OF THE DECISION SUPPORT SYSTEM

This chapter deals with the construction of the entire decision support system. At the heart of the decision support system is a simulation model. Supporting the simulation model is a series of UNIX programs which provide a menu and facilitate use of the simulation model. The discussion of these programs and how they work together to provide a useful system is the first topic of this chapter. The second topic addresses the integration of the neural network software and the decision support system.

The Menu System

The menu for the decision support system is shown in figure 12. This menu system was constructed using various UNIX shell scripts and AWK scripts which interact with other each other. Through the menu the user is allowed to input a schedule of products to the system, change operating parameters, load product information, and ultimately run a simulation scenario. Input to the decision support system is done solely through this menu when the simulation model is being used in an interactive mode. Although the menu allows input through a series of programs which query information from the user, a user can also load a text file (provided that it is in the right format) directly into the simulation model. These text files may contain information on products
as well as schedule information. The loading of these files is also done through the menu.

A more detailed discussion of each of the functions of the menu in the text that follows.

---

**WELCOME TO THE TEST FLOOR SIMULATOR**

**SIMULATOR SYSTEM MENU**

(1) **RUN THE CURRENT SIMULATION SCENARIO**

(2) **LOAD AN INPUT FILE**

(3) **CREATE AN INPUT FILE**

(4) **LOAD A DEMAND FILE**

(5) **LOAD A CODE INFORMATION FILE**

(6) **INPUT A SCHEDULE TO THE SYSTEM**

(7) **UPDATE REWORK/ROUTING PERCENTAGES**

(8) **EXIT THE SIMULATOR**

**PLEASE ENTER YOUR CHOICE [ and press the ENTER key] :**

---

**Figure 12**  Test Floor Simulation Menu

---

**1) Run the Current Simulation Scenario**

Once the proper files have been created for the current simulation scenario under study, this option is chosen. Commands activated by choosing this option are responsible
for extracting relevant information from text files which are created by the user through interaction with the menu (using other options). Additionally, the experimental files and the model files which are required by the SIMAN program are constructed based on the information in the text files. After these two required SIMAN files are generated, the SIMAN compiler and linker programs are executed to make an executable simulation program. Once an executable file is formed, the simulation program is ran and output is sent to a file specified by the user. Example output from this command can be seen in Figure 13.

```
Please Enter the Name of the File for the Simulation Output:

PLEASE WAIT

... (PREPARING THE SIMULATION MODEL FOR EXECUTION)
```

**Figure 13** Output From Option 1 of the Test Floor Simulator Menu

(2) **Load an Input File**

This option allows the user to load a flat text file with decision inputs into the simulation model. Decision inputs includes such things as queuing strategies, quantity and types of resources, and work schedules. A sample file is shown in Figure 14. This file details all the relevant information needed to test a set of given decision inputs. The first part of this file lists the various queues at which lots of products can accumulate. The number entered beside each of these queues represents a particular queuing strategy which
is to be used during the simulation run. The second part of this file lists the resources used in the simulation run and the desired quantity of each type of resource is entered beside the appropriate resource name. The third and final part of the file contains information on the operator work schedule which is to be employed during the simulation. The number beside the work schedule represents a certain type of work schedule to be used. Table 10 lists the work schedules and queuing strategies which are used in the simulation runs and the number in the input file which represents their occurrence in the simulation run.

(3) Create an Input File

This option calls a routine which queries the user for information to create the input file. By using this option, the user is able to create a number of simulation scenarios and save them. Use of the input files can be done by using option (2) (described above) which loads information in an input file into the simulation model. An example of an interactive session using this option is shown in Figure 15.

(4) Load a Demand File

This option allows the user to enter a schedule for 12 weeks of production into the simulation model. Provided the file is in the proper format, the user is able to enter information concerning anticipated product mixes into the DSS. The information in this file and the code information file work together to allow the proper products and the information needed to process these products to be known to the simulation system.
The creation of the demand file can be done via option (6) of the menu, or it can be created in an external program and imported into the simulation system. An example of this file (created with Microsoft EXCEL) can be seen in figure 16.

Table 10  Work Schedules and Queuing Strategies used in the Simulation Model

<table>
<thead>
<tr>
<th>Type of Input</th>
<th>True Value</th>
<th>Input File Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Schedules</td>
<td>10 hour shifts</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>12 hour shifts</td>
<td>2</td>
</tr>
<tr>
<td>Queuing Strategy</td>
<td>First In / First Out</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Last In / First Out</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Lowest Lot Processing Time</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Highest Lot Processing Time</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Products with Highest Demand</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Products with Lowest Demand</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Most Test Time to Fill Demand</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Least Test Time to Fill Demand</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Smallest Slack Time for Product</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Largest Slack Time for Product</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Smallest Lot Slack Time</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Largest Lot Slack Time</td>
<td>12</td>
</tr>
</tbody>
</table>

(5) Load a Code Information File

By using this option, the user is able to load information about products (codes) into the simulation program. The codes or products which are found in this file are also
RANKINGS FOR QUEUES:

ApTapeQ : 1
IVQ : 1
TVSQ : 1
ProbeQ : 1
OffLineInkQ : 1
BakeInkQ : 1
InspectorQ : 1

QTY of RESOURCES:

BGMach : 2
DeTape : 1
ApTape : 1
sBGRinDry : 2
LBGRindDry : 2
TVSMach : 1
IVMach : 1
MemTester : 3
BakRD : 2
Inker : 2
Baker : 1
Tester_A_1 : 2
Tester_A_2 : 1
Tester_A_3 : 2
Tester_B : 1
Tester_C : 1

Figure 14   Sample Input File

found in the demand file. As stated in the previous section, the information in these two
files are needed to process products in the test floor simulation. Like the demand file, this
file can be created via option (6) of the menu, or it can be created by an external program and imported into the simulation program using this option. An example of this file as created with Microsoft EXCEL is shown in figure 17.

(6) Input a Schedule to the System

This option allows the user to interactively load a demand file and a code information file into the simulation program. Programs called by this option invoke routines which query the user for information concerning the anticipated demand/schedule of production and the information which is needed to process the products. Part of an interactive session with the user is shown in Figure 18.

(7) Update Rework/Routing Percentages

Parameters which dictate the percentage of rework and how products may be processed through test floor operations are entered using this option. Through iterative prompting of the user, the program called by this option obtains information needed to update the simulation model for future executions. Refer to Figure 20 for a viewing of this interactive session.
Please answer the following questions about the Wafer Probe Configuration which you wish to simulate:

- How Many BackGrind Machines:
- How Many Apply Tape Machines:
- How Many DeTape Machines:
- How Many IV Machines:
- How Many TVS Machines:
- How Many Short-Cycle Rinser/Dryers in the BackGrind Area:
- How Many Long-Cycle Rinser/Dryers in the BackGrind Area:
- How Many Memory Testers:
- How Many Rinser/Dryers at BakeInk Step:
- How Many Ink Probers:
- How Many Ovens for Baking Ink:
- How Many Type A-1 Testers:
- How Many Type A-2 Testers:
- How Many Type A-3 Testers:
- How Many Type B Testers:
- How Many Type C Testers:

Enter the Work Schedule for the simulation...

(1) 12 hour shifts (A,B,C, and D shifts)
(2) 10 hour shifts (A,B,C, and D shifts)

Please Answer Questions about the Ordering of Lots in Queue at the different processing stations

Please answer the questions by using the following codes:

(1) FIRST IN / FIRST OUT
(2) LAST IN / FIRST OUT
(3) Lots with Lowest Lot Test Time are First
(4) Lots with Highest Lot Test Time are First
(5) Codes with least Quantity of Lots Needed are First
(6) Codes with Highest Quantity of Lots Needed are First
(7) Codes which require the least amount of test time to fill that weeks demand are first
(8) Codes which require the highest amount of test time to fill that weeks demand are first
(9) Lots with the Highest Slack Time are first
(10) Lots with the Lowest Lot Slack Time are first
(11) Lots with the Highest Lot Slack Time for that Code (i.e. Duedate - Test Time required to fill demand)
(12) Lots with the Lowest Slack Time for that Code

Please Choose Queue Ordering for the BackGrind Area:
Please Choose Queue Ordering for IV:
Please Choose Queue Ordering for TVS:
Please Choose Queue Ordering for Probe:
Please Choose Queue Ordering for Off-Line Ink:
Please Choose Queue Ordering for Bake Ink:
Please Choose Queue Ordering for Inspection:
Please Enter the name of the file in which you wish to save the input:

---

Figure 15   Output from Option 3 of the Test Floor Simulation Menu
<table>
<thead>
<tr>
<th>Product</th>
<th>YIELD</th>
<th>SITES</th>
<th>WK1</th>
<th>WK2</th>
<th>WK3</th>
<th>WK4</th>
<th>WK5</th>
<th>WK6</th>
<th>WK7</th>
<th>WK8</th>
<th>WK9</th>
<th>WK10</th>
<th>WK11</th>
<th>WK12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7836</td>
<td>499</td>
<td>0</td>
<td>17595</td>
<td>0</td>
<td>17595</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.7167</td>
<td>300</td>
<td>77400</td>
<td>58050</td>
<td>48375</td>
<td>0</td>
<td>31283</td>
<td>93848</td>
<td>0</td>
<td>10428</td>
<td>62565</td>
<td>41710</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.7303</td>
<td>330</td>
<td>0</td>
<td>10845</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.4353</td>
<td>85</td>
<td>4995</td>
<td>3330</td>
<td>3330</td>
<td>6660</td>
<td>5384</td>
<td>7178</td>
<td>0</td>
<td>5384</td>
<td>0</td>
<td>1795</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0.9075</td>
<td>681</td>
<td>0</td>
<td>27810</td>
<td>27810</td>
<td>0</td>
<td>0</td>
<td>29973</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.438</td>
<td>121</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2385</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2571</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.8635</td>
<td>359</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15035</td>
<td>0</td>
<td>0</td>
<td>15035</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0.5833</td>
<td>180</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20370</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0.5348</td>
<td>187</td>
<td>4500</td>
<td>4500</td>
<td>4500</td>
<td>9700</td>
<td>0</td>
<td>19400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0.7973</td>
<td>809</td>
<td>29025</td>
<td>0</td>
<td>29025</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>31283</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0.9466</td>
<td>412</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2E+05</td>
<td>56745</td>
<td>0</td>
<td>0</td>
<td>18915</td>
<td>56745</td>
<td>18915</td>
<td>1E+05</td>
<td>132405</td>
</tr>
<tr>
<td>13</td>
<td>0.5579</td>
<td>233</td>
<td>126100</td>
<td>37830</td>
<td>25220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0.6619</td>
<td>210</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6741</td>
<td>40499</td>
<td>47191</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20225</td>
<td>13483</td>
<td>13483</td>
</tr>
<tr>
<td>15</td>
<td>0.6881</td>
<td>218</td>
<td>14550</td>
<td>0</td>
<td>36375</td>
<td>29100</td>
<td>29100</td>
<td>7275</td>
<td>0</td>
<td>0</td>
<td>21825</td>
<td>1E+05</td>
<td>65475</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0.8084</td>
<td>488</td>
<td>38315</td>
<td>38315</td>
<td>95788</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>95788</td>
<td>95788</td>
<td>0</td>
<td>38315</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0.7208</td>
<td>240</td>
<td>0</td>
<td>0</td>
<td>92296</td>
<td>41953</td>
<td>67124</td>
<td>25172</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>41953</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0.6854</td>
<td>312</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0.8556</td>
<td>471</td>
<td>0</td>
<td>19546</td>
<td>0</td>
<td>19546</td>
<td>0</td>
<td>19546</td>
<td>0</td>
<td>19546</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0.7394</td>
<td>426</td>
<td>15278</td>
<td>45833</td>
<td>15278</td>
<td>15278</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15278</td>
<td>15278</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0.5982</td>
<td>326</td>
<td>18915</td>
<td>28373</td>
<td>28373</td>
<td>37830</td>
<td>9458</td>
<td>0</td>
<td>0</td>
<td>9458</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0.5656</td>
<td>244</td>
<td>8245</td>
<td>0</td>
<td>12368</td>
<td>4123</td>
<td>8245</td>
<td>4123</td>
<td>4123</td>
<td>8245</td>
<td>4123</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0.4645</td>
<td>183</td>
<td>6693</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6693</td>
<td>6693</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>0.3561</td>
<td>132</td>
<td>0</td>
<td>2280</td>
<td>4559</td>
<td>0</td>
<td>2280</td>
<td>2280</td>
<td>0</td>
<td>2280</td>
<td>0</td>
<td>11398</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0.8418</td>
<td>335</td>
<td>41031</td>
<td>13677</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>27354</td>
<td>54708</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0.8985</td>
<td>473</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20613</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 16 -- Sample Demand File Constructed Using Microsoft EXCEL
<table>
<thead>
<tr>
<th>CODE</th>
<th>SITES</th>
<th>B-Test. GDTT</th>
<th>B-Test. BDTT</th>
<th>C-Test. GDTT</th>
<th>C-Test. BDTT</th>
<th>A-Test. GDTT</th>
<th>A-Test. BDTT</th>
<th>Thick</th>
<th>Test</th>
<th>Yield</th>
<th>Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>499</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.7836</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.7167</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>330</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.7303</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>85</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.4353</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>681</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.9075</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>121</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.438</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.333</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>359</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.8635</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>180</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.5833</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>187</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.5689</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>090</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0.3256</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0.2568</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>330</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>14</td>
<td>0.8654</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>210</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>10</td>
<td>0.6619</td>
</tr>
<tr>
<td>15</td>
<td>218</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>10</td>
<td>0.9654</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.524</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>880</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0.358</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0.584</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0.854</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.954</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>70</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.755</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>260</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.568</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>260</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.568</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 17 -- Sample Code Information File Constructed Using Microsoft EXCEL
Please Answer the Following Questions by" Keying in your response and pressing the Enter Key

How Many Codes Are in This Schedule :
Please Enter a Code :

Please Enter Demand for the Code"
[Enter chip quantities for probed codes and wafer quantities for" unprobed and engineering codes]

Please Enter Week 1 Demand :
Please Enter Week 2 Demand :
Please Enter Week 3 Demand :

Please Answer the following questions about the Code you entered :

Please Enter the proper routing for this Code

(1) Engineering
(2) Unprobed
(3) Probed with No Ink
(4) Probed with Ink

Choice is:"

Is the code you entered a Memory Code [y or n] :
Enter the Planned Yield for this code: [e.g. .45 for a 45% chip yield per wafer]
What Size Wafer is the code on ( (1) 4 in. or (2) 5 in. ) :

What Thickness is the Wafer

(1) Thick (requires only 1 BackGrind Spindle)
(2) Medium (requires 2 BackGrind Spindles)
(3) Thin (requires 3 BackGrind Spindles)

How Many Sites are on Each Wafer :

Figure 18  Output from Option 6 of the Test Floor Simulator Menu
What Tester is Required for This ASIC code:

Choose one of the following:

1. Type B Tester
2. Type C Tester
3. Type A-1 Tester
4. Type A-2 Tester
5. Type B Tester or Type C Tester
6. Type B Tester or Type A-1 Tester
7. Type B Tester or Type A-2 Tester
8. Type C Tester or Type A-2 Tester
9. Type C Tester or Type A-1 Tester
10. Type A-2 or Type A-1 Tester
11. Type B Tester, Type C Tester, or Type A-2 Tester
12. Type B Tester, Type C Tester, or Type A-1 Tester
13. Type C Tester, Type A-2 Tester, or Type A-1 Tester
14. Type B Tester, Type A-2 Tester, or Type A-1 Tester (128pin)
15. Type B Tester, Type C Tester, Type A-1, or Type A-2 Tester

Choice is:

Enter Type B Tester Good Die Test Time: 
Enter Type B Tester Bad Die Test Time: 

Is The Following Information Correct" for the Code you Entered: 

Code:
Required Tester:
Wafer Size:
Wafer Thickness:
Total Sites:
Type B Tester Good Die Test Time =
Type B Tester Bad Die Test Time =
Planned Yield:
Required Routing:

Yes or No [y or n]: 

Figure 18 (continued) Output from Option 6 of the Test Floor Simulator Menu
(8) Exit the Simulator

Choosing this option allows the user to exit from the interactive mode of the
decision support system.

<table>
<thead>
<tr>
<th>PLEASE ANSWER THE FOLLOWING QUESTIONS CONCERNING REWORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Portion of Lots Require 100% IV Testing:</td>
</tr>
<tr>
<td>What Portion of Lots Require 100% PreTest:</td>
</tr>
<tr>
<td>What Portion of Lots Require 100% IV and PreTest:</td>
</tr>
<tr>
<td>What Portion of Lots Have to be Reprobed:</td>
</tr>
<tr>
<td>What Portion of Lots Have to Be Reworked After Probe:</td>
</tr>
</tbody>
</table>

Figure 19 Output from Option 7 of the Test Floor Menu

Through the use of these menu options, the user is able to create various simulation
scenarios and execute them. In this manner the user is freed from the burden of having to
be familiar enough with the simulation language to be able to directly alter the
experimental or model files required by SIMAN. Furthermore, anyone who is familiar
with SIMAN, will find it mush easier to use the interactive programs as a means of
modifying the simulation program.
Integration of Artificial Neural Networks into the DSS

Although the potential for artificial neural networks has been shown, the integration of this technology into the decision support system has not been completed as of the writing of this work. However, plans for doing this are presented here so as to convey to the reader the ideas that the author has concerning the means by which this task can be accomplished.

To allow easy use of the neural network technology, the options needed for neural network analysis would be added to the existing menu. These options would allow the user to decide which input and output variables of interest are to be used in the analysis. In this way the appropriate amount of input and output nodes could be used in the design and construction of an appropriate neural network. Furthermore rules for determining the proper size of the hidden layer would have to be devised. The parameters for training would also need to be determined. It might turn out that the training parameters used in the experiments presented in chapter five are robust enough to be employed for the training of any neural network. Additionally, the level to which a network is trained must follow some predefined guideline.

With regard to generating simulation input-output pairs for the simulation, there are many issues with which to be concerned. One such issue involves the number of simulation runs which need to be performed in order to provide an adequate training set to the neural network. With the neural network that was experimented with in chapter five, a total of 29 of a possible 108 input vectors were used. This means that roughly one third
of the possible inputs that could have been simulated were simulated and used for training the network. For one class of simulations, a one third ratio may be adequate, for other classes a higher or lower ratio may be required.

In the manner in which it is used here, a class of simulations refers to a certain demand schedule with certain product types, under constant operating parameters (rework percentages, routing percentages, etc.) This brings up another interesting aspect. When the parameters of operation are changed or the schedule of products is changed, a neural network must be retrained. In addition, if the analyst (or user) wishes to vary the levels of a set of different simulation input variables, a new network must be constructed. If the analyst uses the neural network to learn the relationship between decision inputs and simulation output, he or she is only concerned with the impacts that decisions will have on test floor performance. Variation of the operating parameters is something which the analyst or decision maker has no control over, and these parameters are considered part of the test floor system. Consequently, the values for these parameters are seen as uncontrollable factors when the test floor is simulated and are not accounted for in the neural network architecture.

With the activation of the menu function which allows for neural network analysis, programs would be executed which produce files that are readable by the XERION software. These files would contain information to be used as blueprints in the construction of an appropriate neural network. Additionally, this menu function would start an iterative program which would generate random inputs and execute the test floor
simulation. The inputs and outputs of these simulation runs would then be used as training pairs for the neural network. Once an adequate level of learning is reached by the neural network, the analyst is asked for levels of the output which he or she wishes to obtain. The trained neural network would then suggest appropriate levels of decision inputs to achieve those output levels. These suggested inputs would then be fed back into the test floor simulation model and the simulation program would be re-executed. After execution, the output obtained from the simulation run would be compared against the desired output. An error calculation would be done (similar to the one presented in chapter five) and would be presented to the user. If the error was within an acceptable limit (e.g. less than 5), the system would present the findings to the user. If the error did not lie within the acceptable limit then the user would be notified that the desired output which was entered is not feasible for the test floor system, and, thus, the neural network can offer no assistance.

In the manner described above, artificial neural network technology could be used as an analysis tool. It is important to stress again that neural networks are only a means of learning simulation input-output relationships and are in and of themselves not a means of optimizing a given system. However, the use of this technology in a decision support system does show some definite potential. The guidelines and suggestions for implementing this type of technology (as outlined in this section) should serve as a means of properly using artificial neural network technology in the decision support system which has been developed.
In summary, a basic decision support system has been constructed. At the heart of this system is a simulation model. Through the use of a provided menu, the analyst, or user of this system, is able to test the effect of various decisions on the performance of the test floor at AT&T. The integration of neural networks into this decision support system has not been completed as of the writing of this thesis. However, preliminary plans and guidelines have been presented which should allow this task to be carried out. It should be noted, though, that further experimentation with neural networks must be done in order to get a better "feel" for the use of this technology as applied to this problem. Specifically, robust methods for determining the parameters needed for constructing and training neural networks must be devised. The application of robust methods would allow the analysis performed by the neural network to be applicable in a wide range of simulation scenarios which a neural network would be required to lean.
CHAPTER SEVEN

CONCLUSIONS FROM THE RESEARCH

With the conclusion of the research presented here some important aspects concerning
decision support systems and the use or artificial neural network technology have been
illuminated. The potential for using this form of artificial intelligence in a real world
application has been shown to have potential.

A simulation decision support system for AT&T Microelectronics has been developed.
This system with its use of simulation is able to capture the random nature and complexity
involved with semiconductor test operations. A neural network is then able to offer
assistance in determining ways for the decision maker to configure the system such that
company goals can be achieved.

As an analysis tool, neural networks offer the decision maker a means for improving
the system being simulated. As stated in earlier chapters, neural networks are not tools for
optimization. They simply offer the decision maker some guidance in being able to meet
specific goals. It should be noted that using neural networks does not offer the same type
of assistance found with response surface methodology. With response surface
methodology, the analyst is able to use experimental data (simulation input and output) to
formulate an equation which relates input and outputs. The defining of the response
surface allows the user to determine or predict the outcome (response) that should be
experienced with a given set of inputs. However, the use of response surface methodology requires the analyst to run several experiments for each simulation output. Once the defining of the response surface for each of the outputs has been completed, the analyst must incorporate additional techniques to make sure overall goals are met. One means of doing this is goal programming. Although these methods do allow the analyst to reach a decision with regard to meeting objectives, they require a lot of effort. Not only must the analyst define the response surface for each simulation, which requires a large number of simulation runs, but the careful formulation and execution of a proper goal program must also be done. Additionally, with an increase in the number of inputs and/or outputs, the required effort on the part of the analyst is magnified.

The use of neural networks, however, as presented in this thesis, allows the user to specify certain levels of many responses, and appropriate input is suggested in order to achieve these responses.

If the goal of the decision maker is to seek improvement in a system, and not specifically to optimize some single response which is often identified with the success of the system, then neural network technology could be the answer. The advantages of this approach of analysis lies in its stand-alone ability. By having a tool which learns the reverse relationship of a simulation model, the analyst is able to set up a program to run on its own. This program would be able to execute the number of simulations needed to train a neural network, train the neural network, and then offer suggestions for achieving goals of the decision maker. Since the neural network is able to learn a relationship, there is no
need to explicitly model the reverse process of the simulation. No logic must be defined, no additional abstractions must be performed, and no assumptions must be made concerning the simulated system [as far as the development of the neural network tool is concerned]. An additional advantage gained with neural networks is seen when this technology is combined with simulation. Unlike many other applications of neural networks which rely on historical data for a training base, the application presented here allows scenarios other than those experienced in real life to be used in training the network. The simulation model allows the collection of data from systems which have not yet been implemented and the neural network is able to learn these new simulated systems. Furthermore, the simulation can be used for verification of the output from the neural network. In this manner, no blind risk is assumed in employing the abilities of this form of artificial intelligence.

Along with the advantages listed above, there are also some disadvantages which exist. One prominent disadvantage is seen when trying to create simulation data with which the neural network can be trained. With any moderate to large system involved there are a lot of inputs and a lot of outputs with which the analyst may be concerned. In order to train the neural network, it is required that an adequate representation of all possible values be presented to the network so that it can learn to a sufficient level. Additionally, the time required to train the network may become quite large. Another, not so apparent disadvantage with using neural networks, is inherent with this form of technology. Often, neural networks are seen as a "black box" and consequently their innerworkings are not
well understood. For this reason the progress of the use of simulation combined with neural networks may be impeded.

One of the most important areas to explore with this new technology involves trying to define the areas in which neural nets can be applied and how robust this approach to simulation analysis actually is. For these reasons, progress must be made in order to find -- for a given system -- what range of parameter values, product mixes, etc. can be accounted for without having to retrain a neural network. Also, in any comprehensive decision support system, ways of limiting desired output must be addressed so that a trained neural network will be able to offer assistance to the analyst. With infeasible simulation outputs neural nets will perform little better than random guesswork. Perhaps the addition of some checking routine to look at output feasibility can be employed. The generalization abilities of any trained network must also be maintained. This means that the network has the ability not only to learn the relationship between the training pairs in the training set, but it also is able to generalize or produce a set of simulation inputs from a set outputs which were not present in the training set.

From a theoretical stand point, comparisons between response surface methodology and neural network analysis need to be performed. In doing this, the researcher may gain a better understanding of the innerworkings of neural nets. Additionally, a base for objectively assessing this new technology could be better defined. Even though there are inherent differences which exist between these two approaches, a formal comparison could shed some light on possible future directions.
Overall, the research presented here has offered a new perspective on simulation analysis. A new tool for doing this analysis, artificial neural networks, has been explored. The setting for the introduction of this technology is semiconductor test operations. Specifically, the use of neural networks with a simulation based decision support system has been explored. The application of neural nets as a means of guiding the analyst to arrive at appropriate decisions has been the focus of the study. Although this approach shows definite potential, some areas for further investigation have been mentioned. With the advance of this approach to simulation analysis, practical and theoretical advances should be realized in the area of decision making sciences.
APPENDICES
APPENDIX 1

TEST FLOOR PROCESSING STEPS
This appendix describes each of the processing steps included during test floor operations.

<table>
<thead>
<tr>
<th>Processing Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply Tape</td>
<td>This step is required to prevent damaging of the product during the backgrind operation</td>
</tr>
<tr>
<td>BackGrind</td>
<td>This step is required for ensuring that the electronic chips will fit into the housing for the integrated circuits made by AT &amp; T's customers</td>
</tr>
<tr>
<td>Short cycle Rinser/Dryer</td>
<td>This step is used to remove potentially contaminating Rinser / Dryer material from the product</td>
</tr>
<tr>
<td>DeTape</td>
<td>This step is required to remove the adhesive tape applied to the product in the &quot;Apply Tape&quot; step</td>
</tr>
<tr>
<td>Long cycle Rinser / Dryer</td>
<td>This step removes any contaminants which may have have fallen on the product during the &quot;Detape&quot; step</td>
</tr>
<tr>
<td>Step</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TVS</td>
<td>This is a required diagnostic test which ensures that the level of sodium in the product is not above the maximum allowed level.</td>
</tr>
<tr>
<td>IV</td>
<td>This is a diagnostic test which checks current (I) and voltage (V) parameters of the product.</td>
</tr>
<tr>
<td>PreTest</td>
<td>This is a diagnostic test which further tests the electrical performance of the product.</td>
</tr>
<tr>
<td>Probe (optical test)</td>
<td>This step performs visual inspection of all components in each of the dies located on every wafer of a given lot.</td>
</tr>
<tr>
<td>Off-Line Ink</td>
<td>This step is used for those products which are sent to customers who require that all bad dies be visually marked.</td>
</tr>
<tr>
<td>Bake Ink</td>
<td>This step is required to dry the ink applied in the &quot;Off-Line Ink&quot; step</td>
</tr>
<tr>
<td><strong>Cool</strong></td>
<td>This step is required to allow the product to cool after it has been baked.</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Bake area Rinser / Dryer</strong></td>
<td>Required to remove contaminants that may have contacted the product during the &quot;Bake Ink&quot; step</td>
</tr>
<tr>
<td><strong>In-Process Inspect</strong></td>
<td>This step is required to ensure that defects which can be visually seen and were not caught during the &quot;Probe&quot; step are caught before the product is sent to finished goods inventory.</td>
</tr>
</tbody>
</table>
APPENDIX 2

SIMULATION MODEL DETAILS
The purpose of this appendix is to detail all the relevant assumptions, information, and data that were used in constructing a simulation model of the Wafer Probe area. The data used and the assumptions made in constructing the simulation model stem from conversations with test floor personnel, from direct observation of operations, and from analysis of historical data, whenever it was available.

The information in this appendix is presented as follows:

1. Product mix and arrival of lots to the test area
2. Routing of products through the test area
3. Scheduling Strategies which may be employed
4. Maintenance Issues
5. Assumptions about operational metrics predicted
6. Probe Personnel as they relate to the flow of lots
7. Detailed description of lot flow through Test Area

(1) **Product Mix and Arrival of Lots**

To use the tool being developed, a schedule (or program) is loaded into the simulation model. The following assumptions and/or decisions were made concerning a schedule under study.

I. The study period is 12 weeks in length (roughly a business quarter). This length was chosen because of the availability of information regarding
projected demand and product mix from the forecasting department at AT&T.

II. The schedule loaded into the system, allows the simulation model to know the codes, (type products) quantity of chips, and approximate time the chips are needed by the customer. Look at Figure 2 (in chapter three) for a sample schedule.

III. The arrival of lots is assumed to have an exponential distribution. Historical data was obtained from the company database, a histogram was constructed, and the underlying distribution appeared to follow an exponential distribution rather closely. The plotting of interarrival data is shown in Figures 20, 21, and 22. Each of these plots show different spans of time for the data represented by the histogram.

IV. The mean of the exponential distribution for the arrival of lots in minutes computed as follows:

\[
\text{mean} = \frac{\text{lots needed for week}}{10080 \text{ minutes per week}}
\]

... this gives a mean interarrival time in minutes and is may be different for each of the 12 weeks in the study period

\[
\text{Lots needed for week} = \text{sum of all lots needed for each code}
\]

\[
\text{Lots needed for each code} = \text{INTEGER}[(\text{Demand(chips)}) / (\text{Sites on wafer*PlanYield*48.5 wafers})]
\]
Figure 20 -- InterArrival Data for 999 Consecutive Lots

Figure 21 -- InterArrival Data for 500 Consecutive Lots

Figure 22 -- InterArrival Data for 100 Consecutive Lots
In order to obtain lot size, an empirical distribution was formed for two
types of lots. These lot types are regular, production lots and engineering
lots. All lots start with 50 wafers when fabrication is begun. However, by
the time that a lot arrives to the test area, some wafers may have been
scraped. Taking this into consideration, data was obtained from the
company database, and empirical distributions were constructed for both
engineering and production lots. The histograms of the two lot types are
shown in figures 23 and 24. Treatment of lots as two distinct types
with respect to number of wafers in a lot was deemed to be necessary after
initial analysis was done and it was noticed that engineering lots
tended to have a lot fewer wafers per lot than did production lots.

(2) **Routing of Lots Through the Test Area**

I. Sixteen (16) different lot routings are realized by the simulation model.
There are 8 routings for memory lots and there are 8 routings for ASIC
lots. These routings are based on the following process steps (as shown in
Figure 1 of the text):

1) Apply Tape 5) Long Cycle Rinser Dryer
2) BackGrind 6) TVS
3) Short cycle rinser/dryer 7) IV
4) DeTape 8) PreTest
Figure 23 -- Histogram of Production Lotsizes

Figure 24 -- Histogram of Engineering Lotsizes
9) Probe (Optical Test)  13) Long Cycle Rinser / Dryer
10) Off-Line Ink  14) In-Process Inspect
11) Bake Ink  15) Die Bank (inventory)
12) Cool

Variations on this routing include the routing of memory lots versus ASIC lots and the option of omitting one of the following from the above routing:

TVS

Probe (for products that don't require optical testing)

In-process Inspect (omitted only by engineering lots)

Off-Line Ink (steps 11 through 13)

Refer to Appendix 1 for a physical description of the process steps.

II. Treatment of rework steps will be detailed in section (7) of this appendix.

(3) **Scheduling Strategies**

I. All scheduling strategies allowed by the simulation model are based on the ordering of lots in queue at each of the processing steps on the test floor (i.e. the queuing strategy).

II. The following are allowed orderings for lots in a queue:

1) First In / First Out

2) Last In / First Out
(3) Lots with the lowest expected testing time (at Probe) are first

\[
\text{Expected Testing Time} = \text{Good Die Test Time} \times \text{Good Dies} + \text{Bad Die Test Time} \times \text{Bad Dies} + \text{Total dies} \times 0.40\text{sec (die index & test start)} + \text{Wafers} \times 45\text{sec}
\]

(4) Lots with the highest expected testing time (at Probe [optical test]) are first

(5) Codes which have the highest demand for lots are first

(6) Codes which have the lowest demand for lots are first

(7) Codes which need the largest amount of test time to fill demand, where test time to fill demand is given by

\[
\text{Test Time to fill Demand} = \text{Lots needed} \times \text{Expected Lot Test Time}
\]

(8) Codes which need the least amount of test time to fill demand

(9) Codes which have the smallest slack time value, where slack time is defined by

\[
\text{Slack Time} = \text{Due Date} - \text{Test Time needed to fill Demand}
\]

and

Due Date for week is 6:00 P.M. Sunday Morning of each week

(10) Codes which have the highest slack time value

(11) Lots which have the smallest slack time value where Lot Slack Time is given by

\[
\text{Lot Slack Time} = \text{Due Date} - \text{Expected Lot Test Time}
\]

(12) Lots which have the largest slack time value
III. All Queuing strategies which require expected test times are considered to be planned for testing on an A-Tester unless they specifically require the B-Tester, or C-Tester. This assumption is made to account for the fact that most codes are tested on type A-testers (A-1, A-2, or A-3). This also accounts for the different speeds with which different testers operate.

IV. The amount of lots needed takes into account the amount of dies that are currently in process in the test area and the dies which are residing in the Die Bank (finished goods inventory), and the current weeks demand. The demand for each week is equal to that week's scheduled demand plus any demand left over from the previous week.

V. The calculations that are required for the above queuing strategies are done when the lot of wafers arrives to the test area. In other words, these values are calculated once, upon entry of the lot to the test area, and these values are used to rank the lots in whatever queue they may reside.

Furthermore the ranking of queues only applies at the following processing steps:

(1) Apply Tape

(6) TVS

(7) IV

and

(9) Probe
All other processing steps have a First In / First Out ranking for their queues. This assumption considers that the steps which occur between the steps listed or after Probe have comparably short processing times and thus never build up a queue of lots.

(4) **Maintenance Issues**

I. All Downtime captured by the model incorporates unscheduled and scheduled maintenance.

II. Scheduled maintenance only includes those activities with a frequency of once a month or less. This assumption is valid considering that all scheduled maintenance activities not listed below were negligible when considering the overall effect on availability of resources. The following maintenance activities, however, were included in the model.

(1) **BackGrinders**

   weekly 3 hours

(2) **Rinser Dryers**

   monthly 2.5 hours

(3) **Memory Testers**

   weekly 2.5 hours
   monthly .25 hours

(4) **Type B Optical Testers**
III. All scheduled maintenance activities are performed during one machine downtime. This means that daily, weekly, or monthly preventive maintenance activities which are scheduled are taken care of during one machine downtime if the day of occurrence of these activities falls on the same day.

IV. Scheduled maintenance is performed on a staggered basis such that no two machines are scheduled to be down at the same time.

V. Scheduled maintenance and unscheduled maintenance are performed by personnel other than probe process analysts, BackGrind Operators, Inspectors, and Probe Operators. (i.e. Maintenance does not require personnel which are used in moving lots of products through the test area.)

VI. Unscheduled maintenance involves the optical testers only. This assumption was believed to be valid after discussing the issue with
maintenance engineers. Furthermore, it was understood that the frequency of breakdowns, the short downtime durations, and the fact that the optical tester processing step (probe) was indeed the bottleneck operation served as further validation for this assumption.

VII. The time that elapses between unscheduled maintenance activities is assumed to follow a Gamma distribution and the time taken to perform an unscheduled maintenance activity also follows a Gamma distribution. These distributions were determined as follows:

Estimates for mean machine downtime and efficiency of machinery were obtained from Probe and maintenance personnel. These estimates were seen to be adequate considering the time required and complexity involved with extracting relevant information from historical data. Furthermore, efficiency is defined as follows:


Efficiency = e = the long run proportion of potential processing time

This value was estimated at 0.9 and mean downtime was estimated at approximately 3 hours. The simulation model assumes a Gamma distribution which is used quite often for time between failures and machine repair time. The parameters needed for distribution are a shape and a scale parameter. The shape parameters for downtime and time between failures are 1.4 and 0.7 respectively. These numbers are
consistent with most historical data collected during downtime studies.

The scale parameters that needed to be determined were:

**Distribution for Time between failures**

scale parameter \( B = \frac{\text{efficiency} \times \text{mean(downtime)}}{(0.7 \times (1-e))} \)

The resulting distribution is Gamma\((2270, 75)\) in minutes

**Distribution for Repair Times**

scale parameter \( B = \frac{\text{mean downtime}}{1.4} \)

The resulting distribution is Gamma\((134, 1.35)\) in minutes

Figures 25 and 26 show the distributions used to model time between breakdowns and time to repair.

VIII. Machine Breakdown cannot occur while an unscheduled maintenance activity is being performed. In other words, a breakdown can occur only when a machine is idle or active. It cannot occur while an unscheduled maintenance activity is occurring.

5) **Operational Metrics Being Predicted**

I. Utilization of equipment

The values of these numbers represent the proportion of time that the machine was dedicated to a lot of wafers.
REAL data Data pts = 500 Intervals = 22 Range: 0 to 1.09 e+04
Mean = 1.71 e+03 StdDev = 1.88 e+03 Min = 0.107 Max = 1.09 e+04

Gamma DISTRIBUTION: Gam(2.37 e+03, 0.754)
Sq Error = 0.00154 Chi Sq: p = 0.448 KS: p > 0.15

Figure 25 -- Distribution of Time Between Machine Failures

REAL data Data pts = 500 Intervals = 22 Range: 1 to 1.1 e+03
Mean = 1.82 StdDev = 1.62 Min = 1.04 Max = 1.1 e+03

Gamma DISTRIBUTION: 1 + Gam(134, 1.35)
Sq Error = 0.00133 Chi Sq: p = 0.458 KS: p > 0.15

Figure 26 -- Distribution of Time to Repair
II. Cycle Time

This number represents the total time spent from entering the test floor at Apply Tape through in-process inspect. Times are collected for each individual code, lots that are reworked, lots that are not reworked, and for individual codes on the basis of rework or no rework.

IV. Late Jobs

This number is determined by checking the inventory level at 6:00 P.M. on Sunday evening of each week (shipping time). If the inventory level is negative (i.e. dies are put on back order), then that code is considered to be late.

V. Throughput

This number represents the average number of lots completely processed each week during the simulation study period.

VI. Lots in test area longer than 48 hours

The quantity of lots whose cycle time exceeded 48 hours. This metric gives management an idea of the turn around time capability in the test area.

VII. Work in Process Inventory

The quantity of lots or wafers present within the backgrind to in-process inspect processing steps
IX. All Metrics have an average value and a sample standard deviation provided by the simulation run. Appendix three provides details of the output data analysis which ensures the precision of the predicted metrics.

(6) **Probe Personnel as They Relate to the Flow of Lots**

I. BackGrind Operators and Probe Operators are required for transporting lots, setting up equipment, performing spc activities, and downloading lots from equipment. These are the only activities with which the operators are associated.

II. Operator work shifts are 10 or 12 hours long and start at the following times:

- **A shift**  --  5:00 a.m. or 7:00 a.m. Sun, Mon, Tues, and Wed
- **B shift**  --  5:00 p.m. or 7:00 p.m. Sun, Mon, Tues, and Wed
- **C shift**  --  5:00 a.m. or 7:00 a.m. Wed, Thurs, Fri, and Sat
- **D shift**  --  5:00 p.m. or 7:00 p.m. Wed, Thurs, Fri, and Sat

Wednesday is a day which is alternated between A and C shifts and B and D shifts.

II. Three (3) BackGrind Operators are present on A and C shift, and two (2) BackGrind Operators are present on B and D shift.

III. BackGrind Operators process a lot from Apply Tape through PreTest.

IV. Four (4) Probe Operators are present A, B, C, and D shift. Probe operators
process lots of wafers from Probe (optical test) up to but not including in-process inspect.

V. To Load most equipment with a lot requires from 1 to 2 minutes with a mostly likely value of 1.5 minutes \(\rightarrow\) Triangular Distribution \((1,1.5,2)\)

VI. BackGrind Setup times depend on the required thickness of the wafers in a lot and the Circumference of the wafer (5 or 6 in.). Times are determined as follows:

For the following setup times, TRI\((a,b,c)\) refers to a triangular distribution with a lowest value of "a" minutes, a most likely value of "b" minutes, and a highest value of "c" minutes. Figure 27 shows a typical triangular distribution. If a lot has 5" diameter wafers and the previous lot had 4" diameter wafers, then an additional time of TRI\((20,25,30)\) minutes is required to perform the setup. In order to setup a backgrind machine for processing a lot, the thickness of the current lot and previous lot must also be considered. When changing from one thickness to another, the following setup times are found to be appropriate:

- No spindle change \(\rightarrow\) TRI\((1,1.5,2)\)
- Medium to Thick spindle change \(\rightarrow\) TRI\((11,13,15)\)
- Thin to Thick spindle change \(\rightarrow\) TRI\((11,13,15)\)
- Thin to Med spindle change \(\rightarrow\) TRI\((22,26,30)\)
- Thick to Med spindle change \(\rightarrow\) TRI\((22,26,30)\)
Figure 27 -- Typical Triangular Distribution
Thick to Thin spindle change  $\mapsto$ TRI(33,41,49)
Medium to Thin spindle change  $\mapsto$ TRI(33,41,49)
Change for different diameter  $\mapsto$ TRI(20,25,30)

VII. TVS setup time follows a TRI(3,5,5,8) distribution which takes into account warm up time for the TVS machine.

VIII. IV and PreTest each follow a TRI(8,10,12) distribution which takes into account the time required to load the appropriate program into the tester.

IX. Memory and Probe setup time is calculated as follows:
Setup Time = TRI(8,12,15) [test program load and lot load]

\[ + \text{Time to test 1 correlation wafer} + \]

\[ [\text{Time to run a 2nd correlation wafer (with 10% prob.)}] \]

\[ + \text{TRI}(1,2,10) \text{(additional set up time)} \]

where a correlation wafer is used to make sure the tester has the proper setup. If a tester is not setup properly done, a second correlation wafer is ran and proper adjustments are made.

X. Off-Line Ink setup follows a TRI(3,4,5) distribution.

XI. In-process Inspect setup time is lumped in with the processing time for this task.

XII. Three (3) in-process inspectors are present on A and C shift and one (1) in-process inspector is present on B and D shift.

XIII. All other setups require a TRI(1,1.5,2) distribution.

XIII. SPC work after the long cycle dryer in the backgrind area requires TRI(3,5,7) minutes.
(7) **Detailed Description of Lot Flow Through the Test Area**

A detailed description of each process step is offered in appendix one. However, this section helps to give a better understanding of the actual flow of lots through the test area. Each process step in this appendix helps explain the product flow shown in Figure 2 of the text (chapter three). Additionally, the time consumed in performing each process step is included.

I. **Apply Tape**

   (1) lots enter the test area at this step with an exponential distribution as detailed in section (1) of this appendix

   (2) this step takes .6167 min. or 37 sec. per wafer with total time for a lot requiring \([\text{Number of Wafers} \times 37 \text{ sec.}]\)

II. **BackGrind**

   (1) lots enter this step after Apply Tape with a FIFO queuing strategy

   (2) BackGrind Time = 14.75 min. + (Number of Wafers - 1) \(\times .8375 \text{ min.} \)

   \((\text{i.e. It takes } 14.75 \text{ minutes for the first wafer to finish and each additional wafer is finished every } .8375 \text{ minutes.})\)

III. **Short Cycle Rinser / Dryer**

   Requires a set amount of time to run the cycle = 3.833 minutes

IV. **DeTape**

   Requires .4417 minutes for each wafer in a lot, with an entire lot taking \([.4417 \text{ minutes} \times \text{Number of Wafers in lot}] \) to process
V. Long Cycle Rinser / Dryer

(1) Requires a set amount of time = 10 min.

(2) S.P.C. measurements and quick inspect are performed by the backgrind operator upon completion of this rinser/dryer cycle. This requires TRI(3,5,7) minutes to complete.

VI. TVS

(1) When a schedule is input to the simulation model, the user states whether or not a particular code will require this step.

(2) Requires TRI(10,15,20) to complete test

VII. IV and Pretest

(1) IV Requires Time to test a 5 wafer sample with each wafer taking TRI(1,3,5,4) minutes

(2) PreTest requires time to test 5 wafers with each taking TRI(2,7,8) minutes

(3) Possible rework at this step is given with a probability defined by the user/analyst of the simulation. Rework percentages are defined for each of the following:

IV required for entire lot (not just a 5 wafer sample)

PreTest required for entire lot (not just a 5 wafer sample)

IV and PreTest required for entire lot (not just a 5 wafer sample)
(4) Rework requires an additional setup for IV and an additional setup for PreTest

(5) Lots can only go through rework once. After this they are scrapped or are sent on to the next processing step.

VIII. Probe (Optical Testing for memory circuits and logic circuits)

(1) When entering a schedule to the simulation model, the user enters:

(a) the good die test time for a code
(b) the bad die test time for a code
(c) the planned yield for a code
(d) the required tester for the code

Die test times are provided by a test engineer. The Bad Die test time provided accounts for 90% of all failures which can occur with a die. Thus all failures are put in a single bad die category and the time to complete this test is constant for all failures that can occur for a certain product type. Good die test time is also treated as a deterministic value which is constant for a given product type.

(2) All lots entering this process, check first to see if a tester of the required type is available. The order for checking testers is:

(1) Type C tester
(2) Type B tester
Type A-1 tester

Type A-2 tester

Type A-3 tester

This ordering ensures that the fastest testers will be chosen first, that a dual head tester will be chosen after a single head tester, and that the A-3 will be chosen last. Choosing the A-3 tester last increases the likelihood of this type of tester being available when a code requires this specific tester. (Only a few codes require this particular tester type).

When multiple testers of a single type are available, the incoming lots are tested on these testers in a cyclical fashion to ensure equal utilization.

In single head mode (CPU runs one test head) the lot test time is calculated as follows:

\[
\text{Lot Test Time} = \text{Good Dies in lot} \times \text{Good Die Test Time (GDTT)} + \text{Bad Dies in lot} \times \text{Bad Die Test Time (BDTT)} + \text{Total Dies} \times .40 \text{ (test start and index time)} + \text{Number of Wafers} \times .75 \text{ minutes (transfer time)}
\]

where ...

Good Dies is chosen from a Normal distribution which approximates the binomial distribution for good and bad dies present in a lot. This distribution is

\[
\text{Normal (nplan,nplanq)}
\]

and

\[
\text{Good Dies} = \text{Integer [Normal (nplan,nplanq) ]}
\]

where...
\[ n_{plan} = \text{number of wafers in lot} \times \text{total sites per wafer} \times \text{planned yield} \]

and

\[ n_{planq} = n_{plan} \times (1 - \text{planned yield}) \]

Planned yield is a number between 0 and 1 that represents the most current ratio of good dies to total dies for any wafer of a certain type product type.

Bad Dies present in a lot are determined by...

\[ \text{Bad Dies} = (\text{Number of total dies in lot} - \text{number of Good Dies in lot}) \]

GDTT (good die test time) and BDTT (bad die test times) are inputs by the user of the system. This information is obtained from test engineers who keep this information current.

(5) Dual head test time is calculated as follows

\[ \text{GDTT}(1) = \text{Good Die Test Time of head 1 code in single head mode} \]

\[ \text{BDTT}(1) = \text{Bad Die Test Time of head 1 code in single head mode} \]

\[ \text{GDTT}(2) = \text{Good Die Test Time of head 2 code in single head mode} \]

\[ \text{BDTT}(2) = \text{Bad Die Test Time of head 2 code in single head mode} \]

\[ P_{g_g} = \text{probability of having a good die on test head 1 followed by a good die on test head 2} \]

\[ P_{g_b} = \text{probability of having a good die on test head 1 followed by a bad die on test head 2} \]

\[ P_{b_b} = \text{probability of having a bad die on test head 1 followed by a bad die on test head 2} \]
\( Pb_g = \) probability of having a bad die on test head 1 followed by a good die on test head 2

The probabilities listed above are calculated as shown with the following example:

\[
P_g_b = \frac{\text{GoodDies(\text{head 1})}}{\text{Total Dies(\text{head 1})}} \times \frac{\text{BadDies(\text{head 2})}}{\text{Total Dies(\text{head 2})}}
\]

The other probabilities are computed in a similar manner.

The test time associated with each pairing of dies assumes that the index time for moving from die to die on a wafer (a single wafer on a test head) is shorter than the Good Die test time duration and is computed as shown below.

- **g_g** test time = \( \text{GDTT(\text{head 1})} + .15 \) (test start) + \( \text{GDTT(\text{head 2})} + .15 \)
- **g_b** test time = \( \text{GDTT(\text{head 1})} + .15 \) (test start) + \( \max(.25, \text{BDTT(\text{head 2})}) + .15 \)
- **b_b** test time = \( \max(.25, \text{BDTT(\text{head 1})}) + .15 + \max(.25, \text{BDTT(\text{head 2})}) + .15 \)
- **b_g** test time = \( \max(.25, \text{BDTT(\text{head 1})}) + .15 + \text{GDTT(\text{head 2})} + .15 \)

where index time = .25 seconds as shown in the formulas above and test start time = .15 seconds as shown in the formulas above.

Lot Test time is computed as follows:

\[
\text{Lot Test Time(1)} = (P_g_g \times \text{g_g test time} + P_g_b \times \text{g_b test time} + Pb_b \times \text{b_b test time} + Pb_g \times \text{b_g test time}) \times \text{Lot(head 1) total dies + Number of Wafer(Lot(head1))} \times .75 \text{ min.}
\]

\[
\text{Lot Test Time(2)} = (P_g_g \times \text{g_g test time} + P_g_b \times \text{g_b test time} + Pb_b \times \text{b_b test time} + Pb_g \times \text{b_g test time}) \times \text{Lot(head 2) total dies +}
\]
Number of Wafers(Lot(head 2))* .75 min.

where .75 minutes is the time required to unload one wafer and load the next wafer.

It is assumed that a dual head tester can test in either dual or single head mode. If a lot is being tested on a dual head tester in single mode (i.e. it is the only lot being tested on that tester), and a second lot arrives, then the tester switches to dual head mode once the second lot has been setup for testing. In order to keep track of the amount of testing completed on a lot, an attribute of the lot known as percentage_complete is calculated each time regular dual or single head testing is interrupted by the entrance or exiting of another lot. This ensures that a lot will not stay on a tester after all dies have been tested for that lot.

(6) When a lot completes testing, the probe queue is queried to find a lot of a code which can be tested on the tester which is now available.

(7) Rework at Probe occurs after testing with a probability provided by the user of the simulation. The time in minutes required to perform rework (time that the lot is held before being released to Off Line Ink) is determined by sampling from a Weibull distribution which was found to be the best fit for historical data. This distribution is...

Rework Time = [126 + WEIBULL(6110, .707)] min.
Lots may also be reprobed after rework with a probability defined by the use of the simulation. In cases where a lot is reprobed, it is assumed that the entire lot must be reprobed. A Lot can only go through rework once, and a lot can only be reprobed once. Figure 28 show the Weibull distribution used for probe rework time.

IX. Off Line Ink

\[ \text{time required} = 0.6667 \text{ minutes} \times \text{Number of Wafers in lot} + \text{quantity of Bad Dies} \times 0.0092 \]

where

0.6667 minutes is the time required to unload one wafer and load the next wafer and

0.0092 minutes is the average time required to index (move) between dies and ink a bad die.

Bad Dies are determined to be bad by the probe (optical test) process step.

This is the process step just prior to the Off Line Ink step.

X. Bake Ink and Cool and Long Cycle Rinser/Dryer at Bake Ink

1. Baking requires 30 minutes
2. Cooling requires 30 minutes
3. Rinser / Dryer 10 minutes
Figure 28 -- Histogram of Probe Rework Time

Figure 29 -- Histogram of Inspection Time
XI. In-Process Inspect

The time required to inspect a lot is given by the distribution:

Inspect Time = (1140 sec. + Erlang(2970,2))/60sec./min.

This distribution was fit by obtaining historical data from process control and using the "Best Fit" Option of the SIMAN input processor. It is shown in Figure 29.
APPENDIX 3

STATISTICAL VALIDATION OF SIMULATION OUTPUT
The analysis presented in this appendix was performed in order to ensure that operational metrics being predicted by the simulation model were accurate. Specifically, the prediction of output metrics within a +/- 10% range of the predicted mean was the goal. For example, the analyst should note that a predicted cycle time of 1650 time units is sure to be within the range 1650 +/- 165 with a confidence level of 95%. To ensure levels of accuracy such as this, a warm-up period and simulation runlength had to be determined. The data used for analysis comes from the simulation of the test floor with its current operating conditions.

In doing the initial analysis, it was noted that a simulation run of 12 weeks, which is the time period covered by the schedule, did not allow for an adequate warm up period or simulation run length. To combat this problem, a long run of the test floor simulation was performed. This run encompassed 10 back to back 12 week schedules. Each of these 12 week schedules were identical and represented the reoccurrence of the product mix contained in the schedule. Defining the test floor simulation as a non-terminating system (one which does not have a definite stopping or starting state) allowed this assumption to be employed. With the long simulation run, an adequate warm up and an adequate number of sample outputs were generated such that the required accuracy and confidence level could be obtained.

In determining the required warm up period and run length, it was decided to look at the work-in-process inventory. Although other metrics of interest could have been used, this metric seemed to be the most stable and predictable. If other metrics were used,
such as cycle times, large variations might be seen due to the variation in test times for different products. Consequently, the determination of a warm up period could become quite difficult if not impossible. However, using lot WIP as the output of interest allowed a warm up period of 250,000 time units to be determined. As can be seen in figure 30, the moving average plot for this metric smooths out around the 250,000 mark. this is the point at which the transient effects of the start up period are no longer seen in the system.

To determine the appropriate run length, a correllologram was constructed using the SIMAN output processor. This graph is shown in figure 31. As seen by this graph, the correlation which exist between observations diminishes around 50,000 time units. To ensure independence between observations of lot WIP, it was decided to use twice the lag length (50,000 time units), or 100,000 time units as the batch size. The batch size is the length of time used for the recording of one observation of lot WIP. For example, with a runlength of 1,200,000 time units, the following number of independent observations can be made:

\[
\text{# of obs.} = \left( 1,200,000 \ [\text{runlength}] - 250,000 \ [\text{warm up period}] \right) / 100,000 \ [\text{batchsize}]
\]

Using these observations, a 95 % confidence can be constructed such as the one shown in figure 32. This plot show that the half-width of the confidence interval is approximately 1.0 units. The desired half-width, however, is 13 * .10, or 1.3, which represents a +/- 10% range about the predicted mean lot WIP of 13.0. Since our actual half-width is less than the desired half-width, we have already achieved the desired level of accuracy. Thus,
Figure 30 -- Moving Average Plot for Lot WIP

Figure 31 -- Correllologram for Lot WIP
Figure 32 -- 95% Confidence Interval for Lot WIP

Figure 33 -- 95% Confidence Interval for Cycle Time
our run length of 1,200,000 is adequate for the purposes of our analysis. Although lot
WIP was decided upon as the output for determining the warm up period and runlength,
other outputs were also of interest. For example, a similar analysis was done for cycle
time. The resulting confidence interval for this metric is shown in figure 33. As can be
seen by this graph, the desired level of accuracy of +/- 10% is again obtained. Other
metrics were treated in a similar fashion, whenever output analysis of the type presented
here was possible.

The reader may wonder how such levels of accuracy can be guaranteed when
inputs to the model are changed for every simulation scenario. For the purposes of the
study performed, a certain runlength was assumed which was believed to allow for this
guarantee. However, the author does admit that this aspect of analysis does deserve
further attention.
APPENDIX 4

EXPERIMENTAL AND MODEL FILE LISTINGS FOR

SIMAN MODEL OF TEST FLOOR
MODEL FILE LISTING
ASSIGN: vdata:
NumCodes,
NumTrills,
NumCreds,
NumAd128_1s,
NumAd128_2s,
Num256s,
NumMems,
InterArrival,
PerIV,
PerPT,
PerIVPT,
PerReProbed,
PerTVS;

QUEUE,
dataQ:DETACH;
CREATE:
EXPO(InterArrival,5):MARK(LotTimeIn);
COUNT:
Wk = 1;
ASSIGN:
LotType = DP(Wk+9,3);  
NumInSystem = NumInSystem+1:  
DueDate = 19800:
M=ENTRANCE:
Wk = Wk+1:
ASSIGN:
NumWafers=DP(CodeInfo(16,LotType),2); is it an eng. lot
WIPP,"(1X,E14.8,1X,E14.8)":TNOW,NumInSystem;
WRITE,
WRITE,
ASSIGN:
WippWAF,"(1X,E14.8,1X,E14.8)":TNOW,NumInSystem;

NumWafers = WipinWafers + NumWafers:
Num128s = NumAd128_1s + NumAd128_2s:
NumProbers = NumTrills+Numcreds+NumAd128_1s+

NumAd128_2s+Num256s:

ET = NumTrills:
EC = NumTrills + NumCreds:
E2 = NumTrills + NumCreds + NumAd128_1s:
E8 = NumTrills + NumCreds + Num128s:
E6 = NumProbers;
ASSIGN:
Lots_Needed = AINT((Demand(Wk,LotType) -

CodeInfo(14,LotType))/(CodeInfo(1,LotType)*

CodeInfo(10,LotType)*48))+1;

BRANCH,1:
IF, (CodeInfo(15,LotType).eq.1),checkprobelot:
else,ProcUnprobedLot;
probelot BRANCH,1:
IF, (CodeInfo(9,LotType).eq.3).or.
(CodeInfo(9,LotType).eq.4).or.
(CodeInfo(9,LotType).gt.5),AdtestTime:
IF, (CodeInfo(9,LotType).eq.1).or.
(CodeInfo(9,LotType).eq.5),TrillTestTime:
else,CredTestTime;
NextTime ASSIGN: TesterType = 5:NEXT(GetPropAss);
TestTime ASSIGN: TesterType = 1:NEXT(GetPropAss);
TestTime ASSIGN: TesterType = 3:NEXT(GetPropAss);

PropAss
ASSIGN: Lot_EPT = NumWafers*(CodeInfo(1,LotType)*

CodeInfo(10,LotType)*CodeInfo(TesterType+1,LotType)+

(1-CodeInfo(10,LotType))*CodeInfo(1,LotType)*

CodeInfo(TesterType+2,LotType)):

NEXT(ProcessLot);

UnprobedLot ASSIGN: PTime_Needed = 300*Lots_Needed:
Lot_EPT = 300:NEXT(ProcLot);

TestLot ASSIGN: PTime_Needed = (Lot_EPT+300)*Lots_Needed;
Lot ASSIGN:  LotSlackTime = DueDate-Lot EPT:
   CodeInfo(14,LotType) = CodeInfo(14,LotType)+
   NumWafers*CodeInfo(10,LotType)*CodeInfo(1,LotType);
   TVSopt=DISCRETE(PerTVS,8,1,0,2);
 ASSIGN:  NS=CodeInfo(11,LotType)+TVSopt:
 ASSIGN:  SlackTime = DueDate-PTime_Needed:
 ASSIGN:  NEXT(BGOperQ);

Lot QUEUE,  BGOperQ;
 REQUEST:  BGOper(SDS,Tvar);
 BRANCH,1:  IF,M.eq.6,CheckLot:
            ELSE,RegUnload;
 Lot DELAY:  TRIANGULAR(3,5,7);    !Time to do SPC and inspect
 Load DELAY:  TRIANGULAR(1,1.5,2);  !UnLoadTime
 TRANSPORT:  ;

Load QUEUE,  ProbeOperQ;
 REQUEST:  ProbeOper(SDS,PVar);
 DELAY:  TRIANGULAR(1,1.5,2);    !UnLoadTime
 TRANSPORT:  ;

STATION,  ApplyTape;
 FREE:  BGOper(Tvar):MARK(ApTapeIn);
 QUEUE,  APTapeQ;
 INCLUDE:  "ApTape"
 QUEUE,  BGOperATQ;
 REQUEST:  BGOper(SDS,Tvar);
 DELAY:  TRIANGULAR(1,1.5,2);    !SetUp time for Tape Machine
 FREE:  BGOper(Tvar);
 DELAY:  NumWafers*.61667;
 TALLY:  ApTapeCyc,INT(ApTapeIn);
 TALLY:  Thru AT Cyc,INT(LotTimeIn);
 RELEASE:  ApTape(AP):NEXT(BGOperQ);

STATION,  BackGrind;
 FREE:  BGOper(Tvar):MARK(BkGTimeIn);
 QUEUE,  BackGrindQ;
 INCLUDE:  "BkGrind"
 BRANCH,1:  IF,BGSETUP(bg+10).eq.CodeInfo(12,LotType),BGSetCk2:
            else,ChgSpindles;
            ChgSp = 1;
 CHck2 BRANCH,1:  IF,BGSETUP(bg).EQ.CodeInfo(8,LotType),NoBGSetUp:
            if,CodeInfo(8,LotType).eq.1.and.BGSETUP(bg).eq.2,
               MedToThick:
               if,CodeInfo(8,LotType).eq.1.and.BGSETUP(bg).eq.3,
                  ThinToThick:
                  if,CodeInfo(8,LotType).eq.2.and.BGSETUP(bg).eq.1,
                     ThickToMed:
                     if,CodeInfo(8,LotType).eq.2.and.BGSETUP(bg).eq.3,
                        ThinToMed:
                        if,CodeInfo(8,LotType).eq.3.and.BGSETUP(bg).eq.1,
                           ThickToThin:
                           if,CodeInfo(8,LotType).eq.3.and.BGSETUP(bg).eq.2,
                              MedToThin:
                              else, NoBgSetUp;
 BGSetUpTime=TRIANGULAR(1,1.5,2)+
            TRIANGULAR(20,25,30)*ChgSp:NEXT(GrindIt);

Chck ASSIGN:  BGSetUpTime=TRIANGULAR(11,13,15)+
            TRIANGULAR(20,25,30)*ChgSp:BGSetUp(BG)=1:NEXT(GrindIt);

Chck ASSIGN:  BGSetUpTime=TRIANGULAR(11,13,15)+
            TRIANGULAR(20,25,30)*ChgSp:BGSetUp(BG)=2:NEXT(GrindIt);

Chck ASSIGN:  BGSetUpTime=TRIANGULAR(22,26,30)+
            TRIANGULAR(20,25,30)*ChgSp:BGSetUp(BG)=3:NEXT(GrindIt);

Med ASSIGN:  BGSetUpTime=TRIANGULAR(22,26,30)+

TRIANGULAR(20,25,30)*ChgSp:BGSetUp(BG)=2:next(GrindIt);
BGSetUpTime=TRIANGULAR(33,41,49) +
TRIANGULAR(20,25,30)*ChgSp:BGSetUp(BG)=3:next(GrindIt);

ASSIGN: BGSetUp(bg+10)=CodeInfo(12,LotType):
ChgSp = 0;
BGOperBGQ;
REQUEST: BGOper(SDS,Tvar);
DELAY: BGSetUpTime; SetUp time for BG Machine
FREE: BGOper(Tvar);
DELAY: 14.75 + (NumWafers -1)*.8375;
TALLY: BkGrindCyc,INT(BkGTimeIn);
RELEASE: BGMach(BG):
NEXT(BGOperQ);

STATE, FREE:
FREE: BGOper(Tvar);
REQUEST: sRinDry;
DELAY: TRIANGULAR(1,1.5,2); SetUp time for BG Machine
FREE: BGOper(Tvar);
DELAY: 3.83333;
RELEASE: sRinDry:
NEXT(BGOperQ);

STATE, FREE:
FREE: BGOper(Tvar);MARK(DeTapeIn);
REQUEST: DTape;
DELAY: TRIANGULAR(1,1.5,2); SetUp time for BG Machine
FREE: BGOper(Tvar);
DELAY: .4417*NumWafers;
TALLY: DeTapeCyc,INT(DeTapeIn);
RELEASE: DeTape:
NEXT(BGOperQ);

STATE, FREE:
FREE: BGOper(Tvar);
REQUEST: LRinDry;
DELAY: TRIANGULAR(1,1.5,2); SetUp time for BG Machine
FREE: BGOper(Tvar);
DELAY: 10;
TALLY: BGandAT_Cyc,INT(LotTimeIn);
RELEASE: LRinDry:
NEXT(BGOperQ);

STATE, FREE:
FREE: BGOper(Tvar);MARK(TVSTimeIn);
REQUEST: TVSStat;
DELAY: TRIANGULAR(3,5.5,8); SetUp time for TVS Machine
FREE: BGOper(Tvar);
DELAY: TRIANGULAR(10,15,20);
TALLY: TVS_Cyc,INT(TVSTimeIn);
TALLY: Thru_TVS_Cyc,INT(LotTimeIn);
TVSMach(TVS):
NEXT(BGOperQ);

STATION, FREE:
BGOper(Tvar):MARK(IVTimeIn);
QUEUE, INCLUDE:
"IV"
REQUEST:
BGOper(SDS,Tvar);
DELAY:
TRIANGULAR(8,10,12);
FREE:
BGOper(Tvar);
DELAY:
TRIANGULAR(1,3.5,4)*5;
TALLY:
IV_Cyc,INT(IVTimeIn):NEXT(BGOperQ);

STATION, DELAY:
TRIANGULAR(8,10,12):MARK(PTTimeIn); SetUp time for IV Machine
FREE:
BGOper(Tvar);
REQUEST:
BGOper(SDS,Tvar);
ASSIGN:
RelIV=1;
LEVEL:
TRIANGULAR(8,10,12); SetUp time for IV Machine
FREE:
BGOper(Tvar);
REQUEST:
BGOper(SDS,Tvar);
ASSIGN:
IV_Cyc,INT(PHTimeIn);
BRANCH,1:
WITH,PerIV,FullIV:
else,IVGood;
REQUEST:
BGOperQ1;
FREE:
BGOper(SDS,Tvar);
ASSIGN:
FullIV=1;
LEVEL:
TRIANGULAR(8,10,12); SetUp time for IV Machine
FREE:
BGOper(Tvar);
REQUEST:
BGOper(SDS,Tvar);
ASSIGN:
TriV=1;
LEVEL:
TRIANGULAR(8,10,12); SetUp time for IV Machine
FREE:
BGOper(Tvar);
REQUEST:
BGOper(SDS,Tvar);
ASSIGN:
IV_Cyc,INT(IVTimeIn):
TALLY:
IVMach(IV);
TALLY:
Thru IV_Cyc,INT(LotTimeIn):
NEXT(ProbeOperQ);

STATION, COUNT:
NumMemLots;
TALLY:
Pre_Mem_Cyc,INT(LotTimeIn);
FREE:
ProbeOper(Pvar):MARK(MemTimeIn);
REQUEST:
ProbeOperMemQ;
ASSIGN:
ProbeOper(SDS,Pvar);
TALLY:
Setup_Mem = TRIANGULAR(8,12,15):
Corr_Waf_Time = CodeInfo(2,LotType):
Check = DP(checkcorr,3):
GDs = CodeInfo(10,LotType)*CodeInfo(1,LotType):
SetUp_Mem + Corr_Waf_Time + Check* (Corr_Waf_Time + Triangular(1,2,10));
FREE:
DELAY:
NumWafers*Corr_Waf_Time;
TALLY:
Mem_Cyc,INT(MemTimeIn);
TALLY:
Thru_MEM_Cyc,INT(LotTimeIn);
RELEASE:
MemTester(Mem):
NEXT(ProbeOperQ);

STATION, TALLY:
Pre_Probe_Cyc,INT(LotTimeIn);
WRITE,PQ:
NQ(ProbeQ);
FREE:
ProbeOper(Pvar):MARK(ProbeTimeIn);
ASSIGN:
U = 1;
TALLY:
TTrip = 0;CTrip = 0;128Trip = 0;256Trip = 0;
BRANCH,1:
0.and.1.le.ET).and.((MR(Prober(U))-NR(Prober(U))).gt.0)
) (Interface(LotType)).lt.2),CheckTJob:
U = U + 1;
if, U.ge.NumProbers, ProberQ:
    else, CheckMore;

T Trip = 1;

if, P(TrillJobs, T Trip).eq.CodeInfo(9, LotType),
    TrillQ:
    else, T NextTrip;
    T Trip = T Trip + 1;

if, C Trip.ge.8, NextUnit:
    else, CheckNextTJob;

C Trip = 1;

if, P(CredJobs, C Trip).eq.CodeInfo(9, LotType), CredQ:
    else, C NextTrip;
    C Trip = C Trip + 1;

if, C Trip.ge.8, NextUnit:
    else, CheckNextCJob;

128Trip = 1;

if, P(128Jobs, 128Trip).eq.CodeInfo(9, LotType), Ad_128Q:
    else, 128NextTrip;

if, 128Trip.ge.8, NextUnit:
    else, CheckNext128Job;

256Trip = 1;

if, P(256Jobs, 256Trip).eq.CodeInfo(9, LotType), Ad_256Q:
    else, 256NextTrip;

if, 256Trip.ge.8, NextUnit:
    else, CheckNext256Job;

ProbeQ:
    DETACH;

TesterType = 1;
    TrillQ;
    "TrillProbe";

TesterType = 3;
    CredQ;
    "CredProbe";

TesterType = 5;
    Ad_256Q;
    "Ad_256Probe";

TesterType = 5;
INCLUDE: "Ad_128Probe";

eLot QUEUE, REQUEST: ProbeOperPQ; PROBEOperPQ);
ASSIGN: ProbeOper(SDS,PVAR);
SetUp Prove = TRIANGULAR(8, 12, 15):
Corr_Waf_Time = Codeinfo(1, LotType)*
Codeinfo(TesterType+1, LotType)*Codeinfo(10, LotType) +
Codeinfo(1, LotType)*Codeinfo(TesterType+2, LotType)*
(1-Codeinfo(10, LotType));
DELAY: Check = DP(checkcorr, 3);
FREE: ProveOper(PVar);
ASSIGN: Jobs(Mach) = Jobs(Mach) + 1:
nplan = NumWafers*Codeinfo(1, LotType)*Codeinfo(10, LotType)
PerLeft = 1.0;
nplanq = (1-Codeinfo(10, LotType))*nplan;
ASSIGN: GDS = ANINT(NORM(nplan, nplanq, 5));
ASSIGN: BDs = NumWafers*Codeinfo(1, LotType)-GDS;
ASSIGN: PTimeSNG = GDS*Codeinfo(TesterType+1, LotType)+
BDs*Codeinfo(TesterType+2, LotType)+
(.00667*Codeinfo(1, LotType)+.75)*NumWafers;
BRANCH, 1:
if, Jobs(Mach) == 1, TestSNG:
else, TestDual;
SNGL QUEUE, PREEMPT:
ASSIGN: SNGTime(Mach) = PTimeSNG * PerLeft:
TestTimeSNG(Mach) = PTimeSNG:
MARK(TimeIn);
ASSIGN: VTimeIn1(Mach) = TimeIn;
Mach + 100;
QUEUE:
ASSIGN: Probe_CPU(Mach);
DELAY:
ASSIGN: SNGLTime(Mach);
RELEASE:
ASSIGN: Probe_CPU(Mach):
NEXT(LeaveStation);
Assume QUEUE,
PREEMPT:
ASSIGN: Probe_CPU(Mach), RemProcTime,
ImagQ:
MARK(TimeIn);
ASSIGN: .001;
ASSIGN: VPerLeft(Mach) = AQUE(Mach + 300, 1, 21) -
(TNOW - VTimeIn1(Mach))/AQUE(Mach + 300, 1, 20):
PreemptLType = AQUE(Mach + 300, 1, 27);
ASSIGN: VTimeIn2(Mach) = TimeIn:
Pg_b = GDS/(NumWafers*Codeinfo(1, LotType))*
AQUE(Mach+300, 1, 25)/(AQUE(Mach+300, 1, 9)*
Codeinfo(1, PreemptLType)):
Pb_b = BDs/(NumWafers*Codeinfo(1, LotType))*
AQUE(Mach+300, 1, 25)/(AQUE(Mach+300, 1, 9)*
Codeinfo(1, PreemptLType)):
Pb_g = BDs/(NumWafers*Codeinfo(1, LotType))*
AQUE(Mach+300, 1, 24)/(AQUE(Mach+300, 1, 9)*
Codeinfo(1, PreemptLType)):
Pq_g = GDS/(NumWafers*Codeinfo(1, LotType))*
AQUE(Mach+300, 1, 24)/(AQUE(Mach+300, 1, 9)*
Codeinfo(1, PreemptLType)):
BDTT1 = Codeinfo(TesterType+2, LotType):
BDTT2 = Codeinfo(TesterType+2, PreemptLType):
GDTT1 = Codeinfo(TesterType+1, LotType):
GDTT2 = Codeinfo(TesterType+1, LotType):
ASSIGN: good_goodtime=.0025+GDTT1+.0025+GDTT2:
good_badtime=.0025+GDTT1+MX(.004167,BDTT1)+.0025:
bad_Badtime=.0025+MX(.004167,BDTT1)+.0025+MX(.25,BDTT2):
bad_goodtime=.0025+MX(.004167,BDTT1)+.0025+MX(.004167,BDTT2):
ASSIGN: PTmDual = Pb_b*bad_goodtime + Pg_b*good_goodtime +
...
Pg g*good_goodtime + Pb g*bad_goodtime:
Lot2TestTime(Mach) = (PTmDual*
CodeInfo(1, LotType) + .75*NumWafers)*PerLeft:
Lot1TestTime(Mach) = VPerLeft(Mach)*(PTmDual*
CodeInfo(1, PremptLType) + AQUE(Mach+300, 1, 9) * .75);

DELAY: Min(Lot1TestTime(Mach), Lot2TestTime(Mach));
RELEASE: Probe_CPU(Mach);
BRANCH,1: if, Lot1TestTime(Mach) <= Lot2TestTime(Mach),
SendOld:
elself, GetOld;

SEARCH, 1
REMOVE: Mach+300:min(PTimeIn);
ASSIGN: PerLeft = PerLeft - ((TNOW - TimeIn)/PTmDual):
NEXT(TestSNG);

REMOVE:
status ASSIGN: 1, Mach + 300, GetStatus:NEXT(LeaveStation); PerLeft = PerLeft -
((TNOW - VTimeIn2(Mach))/PTmDual):
NEXT(TestSNG);

QUEUE,
Status ASSIGN: Mach + 300, 1: MARK(TimeIn):
DETACH;
station ASSIGN: Jobs(Mach) = Jobs(Mach) - 1;
Release: Prober(Mach), 1: Interface(LotType), 1;
COUNT: Mach+11;
COUNT: NumJobsProbed;
BRANCH,1: if, NQ(ProbeQ) > 0. AND. Mach <= NumTrills, GetTrillLot:
NQ(ProbeQ) > 0. AND. Mach <= (NumTrills + NumCreds), GetCredLot:
NQ(ProbeQ) > 0. AND. Mach <= (NumTrills + NumCreds + Num128s), GetAd_128Lot:
NQ(ProbeQ) > 0. AND. Mach <= NumProbers, GetAd_256Lot:
GoToNextSTAT;

TrillLot SEARCH,
ProbeQ: TTyp.eq.1.or.TTyp.eq.5.or.TTyp.eq.6.or.TTyp
.eq.7.or.TTyp.eq.11.or.TTyp.eq.12.or.TTyp.eq.
14.or.TTyp.eq.15.and. (NR(Interface(LotType)).lt.
2);
BRANCH,1: if, J.gt.0, TRemove:
elself, GoToNextSTAT;
J, ProbeQ, TrillQ:NEXT(GoToNextSTAT);

CredLot SEARCH,
ProbeQ: TTyp.eq.2.or.TTyp.eq.5.or.TTyp.eq.8.or.TTyp
.eq.9.or.TTyp.eq.11.or.TTyp.eq.12.or.TTyp.eq.
13.or.TTyp.eq.15.and. 
(NR(Interface(LotType)).lt.2);
BRANCH,1: if, J.gt.0, CRemove:
elself, GoToNextSTAT;
J, ProbeQ, CredQ:NEXT(GoToNextSTAT);

Ad_256Lot SEARCH,
ProbeQ: TTyp.eq.3.or.TTyp.eq.6.or.TTyp.eq.8.or.TTyp
.eq.10.or.TTyp.eq.11.or.TTyp.eq.13.or.TTyp.eq.
14.or.TTyp.eq.15.and. 
(NR(Interface(LotType)).lt.2);
BRANCH,1: if, J.gt.0, 256Remove:
elself, GoToNextSTAT;
J, ProbeQ, Ad_256Q:NEXT(GoToNextSTAT);

Ad_128Lot SEARCH,
ProbeQ: TTyp.eq.4.or.TTyp.eq.7.or.TTyp.eq.9.or.TTyp
.eq.10.or.TTyp.eq.12.or.TTyp.eq.13.or.TTyp.eq.
14.or.TTyp.eq.15.and. 
(NR(Interface(LotType)).lt.2);
BRANCH,1: if, J.gt.0, 128Remove:
elself, GoToNextSTAT;
J,ProbeQ,Ad_128Q:NEXT(GoToNextSTAT);

NextSTAT TALLY:
  QUEUE,
  REQUEST: ProbeOper(SDS,PVar);
  DELAY: TRIANGULAR(5,10,20);
  FREE: ProbeOper(PVar);
  BRANCH,1: if,(ProbeLoops<1).and.(UNIFORM(0,1).LT.0.095),
             ProbeReWork:
             else,GetGone;
             ReworkTime = (6090 + WEIB(4.37E05,.65))/60:
             ProbeLoops = 1;
             COUNT: ProbeReWorkLots;
             DELAY: ReworkTime;
             BRANCH,1: with,PerReProbed,ReProbedLot:
             else,CheckIV;
             ReProbedLots:NEXT(SecondProbe);
TALLY:
  COUNT:
  DELAY: SetUp time for IV Machine
  FREE: BGOper(Tvar);
  DELAY: TRIANGULAR(2,7,8)*NumWafers:NEXT(GetGone);

TALLY:
  QUEUE, BGOperQ2;
  REQUEST: BGOper(SDS,Tvar);
  DELAY: TRIANGULAR(8.5,10.5,12.5); SetUp time for IV Machine
  FREE: BGOper(Tvar);
  DELAY: TRIANGULAR(2,7,8)*NumWafers:NEXT(GetGone);

TALLY:
  QUEUE, BGOperQ3;
  REQUEST: BGOper(SDS,Tvar);
  DELAY: TRIANGULAR(8.5,10.5,12.5); SetUp time for IV Machine
  FREE: BGOper(Tvar);
  DELAY: TRIANGULAR(1,3.5,4)*NumWafers;
  QUEUE, BGOperQ4;
  REQUEST: BGOper(SDS,Tvar);
  DELAY: TRIANGULAR(8,10,12); SetUp time for IV Machine
  FREE: BGOper(Tvar);
  DELAY: TRIANGULAR(2,7,8)*NumWafers:NEXT(GetGone);

TALLY:
  ProbeLoops+20,INT(ProbeTimeIn);
  TALLY: AllProbe_Cyc,INT(ProbeTimeIn);
  TALLY: LotType*2+ProbeLoops+28,INT(ProbeTimeIn):
          NEXT(ProbeOperQ);
  TALLY: LotType+30,INT(ProbeTimeIn);

STATION, OffLineInk;
  FREE: ProbeOper(PVar):MARK(InkTimeIn);
  QUEUE, OffLineInkQ;
  INCLUDE: "OffLineInk";
  QUEUE, ProbeOperQ1;
  REQUEST: ProbeOper(SDS,PVar);
  DELAY: TRIANGULAR(3,4,5);
  FREE: ProbeOper(PVar);
  DELAY: .6667*NumWafers+BDs*.0092;
  DELAY: NORM(7400,2950)/60;
  TALLY: InK_Cyc,INT(InkTimeIn);
  TALLY: Thru_Ink_Cyc,INT(LotTimeIn);
  RELEASE: Inker(INK):NEXT(ProbeOperQ);
STATION, BakeInk;
  FREE: ProbeOper(PVar):MARK(BakeTimeIn);
  QUEUE, BakeInkQ;
  INCLUDE: "BakeInk";
QUEUE, ProbeOperQ2;
REQUEST: ProbeOper(SDS,PVar);
DELAY: TRIANGULAR(1,1.5,2);
FREE: ProbeOper(PVar);
30;
RELEASE: Baker(Bk):NEXT(ProbeOperQ);

STATION, BakeCool;
FREE: ProbeOper(PVar);
DELAY: 30:NEXT(ProbeOperQ);

STATION, BakeRD;
FREE: ProbeOper(PVar);
QUEUE, INCLUDE: "BakeRD";
REQUEST: ProbeOper(SDS,PVar);
DELAY: TRIANGULAR(1,1.5,2);
FREE: ProbeOper(PVar);
DELAY: 10;
TALLY: Bake_Cyc,INT(BakeTimeIn);
TALLY: Thru_Bake_Cyc,INT(LotTimeIn);
RELEASE: BakeRD(BRD):NEXT(ProbeOperQ);

STATION, IPInspect;
FREE: ProbeOper(PVar);
BRANCH,1:
if,Inloop.gt.0,LookLot:
else,GetTimeIn;

Inloop ASSIGN: Inloop = Inloop + 1:MARK(InspectTimeIn);
Lot QUEUE, InspectorQ;
INCLUDE: "IPInspect";
It DELAY: (1140 + ERLA(2970,2))/60;
RELEASE: Inspector(IN);
BRANCH,1: IF,DISCRETE(.78,0,1.0,1).gt.0.and.
Inloop.eq.1,ReBake:
else, GoToDieBank;
M = 11:
Inloop = Inloop + 1:NEXT(ProbeOperQ);
DieBank TALLY: 24+Inloop,INT(InspectTimeIn);
TALLY: All_Inspect_Cyc,INT(LotTimeIn):NEXT(ProbeOperQ);

STATION, Last;
FREE: ProbeOper(PVar);
COUNT: NumLots;
ASSIGN: NumInSystem = NumInSystem - 1:
WipInWafers = WipInWafers - NumWafers:
CycTime = TNOW - LotTimeIn;
WRITE, WIPP,"(1X,E14.8,1X,E14.8)";TNOW,NumInSystem;
WRITE, LOTCYC,TIME,"(1X,E14.8,1X,E14.8)";TNOW,CycTime;
WRITE, WIPPWAF,"(1X,E14.8,1X,E14.8)";TNOW,NumInSystem;
COUNT: JobsOut,1;
COUNT: WafsOut,NumWafers;
COUNT: 11+NumProbers+LotType,NumWafers;
TALLY: LotFlowTime,INT(LotTimeIn);
TALLY: 330+Codeinfo(15,LotType),INT(LotTimeIn);
BRANCH,1: IF,(TNOW-LotTimeIn).gt.2880,CountLong:
else,NoLong;
COUNT: Jobs_LT_48hrs;
TALLY: 229+LotType,INT(LotTimeIn);
ASSIGN: Codeinfo(13,LotType) = Codeinfo(13,LotType)+GDs;
TALLY: LotExitInt,BETWEEN:
DISPOSE;
INCLUDE: "ProberMaint";
INCLUDE: "Work_Schedule";
Create,,19800: "1;
Week ASSIGN:  
nextCode BRANCH, 1:  
tractInv ASSIGN:  
BRANCH, 1:  
Late COUNT:  
NextTrip ASSIGN:  
NextWeek ASSIGN:  
WkStart DELAY:  
NextWeek DELAY:  
RegWeek ASSIGN:  

ITR = 1;  
if, ITR.le. NumCodes, SubtractInv:  
else, GoToNextWeek;  
CodeInfo(13, ITR) = CodeInfo(13, ITR) - Demand(Wk, ITR);  
CodeInfo(14, ITR) = CodeInfo(14, ITR) - Demand(Wk, ITR);  
if, CodeInfo(14, ITR).lt. 0, CountLate:  
else, NextITrip;  
LateJobs, 1;  
ITR = ITR + 1: NEXT(DoNextCode);  
DueDate = DueDate + 10080;  
IF, Wk.eq. 11, ResetWkStart:  
ELSE, HitNextWeek;  
20160: NEXT(DoRegWeek);  
10080;  
Wk = Wk + 1: NEXT(NextWeek);
EXPERIMENTAL FILE LISTING
k55BG;
"Resources";
"Tallies";
ProbesQ:
TrillQ:
CredQ:
Ad_256Q:
Ad_128Q:
MemTestQ:
101,SngQ1:
102,SngQ2:
103,SngQ3:
104,SngQ4:
105,SngQ5:
106,SngQ6:
107,SngQ7:
108,SngQ8:
109,SngQ9:
110,SngQ10:
111,SngQ11:
112,SngQ12:
113,SngQ13:
114,SngQ14:
115,SngQ15:
201,DualQ1:
202,DualQ2:
203,DualQ3:
204,DualQ4:
205,DualQ5:
206,DualQ6:
207,DualQ7:
208,DualQ8:
209,DualQ9:
210,DualQ10:
211,DualQ11:
212,DualQ12:
213,DualQ13:
214,DualQ14:
215,DualQ15:
301,ImagQ1:
302,ImagQ2:
303,ImagQ3:
304,ImagQ4:
305,ImagQ5:
306,ImagQ6:
307,ImagQ7:
308,ImagQ8:
309,ImagQ9:
310,ImagQ10:
311,ImagQ11:
312,ImagQ12:
313,ImagQ13:
314,ImagQ14:
315,ImagQ15:
BGOperQ:DeTapeQ:sBGRinDryQ:LBGRinDryQ:
ApTapeQ:BGOperQ2:IVQ:PreTestQ:TVSQ:BGOperQ1:
dataQ:BGOperQ3:BGOperQ4:BGOperATQ:BGOperBGQ:
BGOperQ:BGOperLRDQ:BGOperDTQ:BGOperTVSQ:
BGOperIVQ:BackGrindQ:BakeInkQ:
BakeRDQ:InspectorQ:OffLineInkQ:
ProbeOperQ1:ProbeOperQ2:ProbeOperQ3:ProbeOperQ:
ProbeOperPQ:ProbeOperP2Q:ProbeOperMemQ:
ProbeOperRewQ:
Ad128.1MaintQ:
| 1 | 1*20000,0*150,1*20170: | 2 | 1*10000,0*150,1*30170: | 3 | 1*4000,0*180,1*5900: | 4 | 1*5500,0*150,1*34670: | 5 | 0*5,3*180,0*30,3*210,0*45,3*245,0*10,1*180,0*30,1*210,0*45,1*245,0*5: | 6 | 0*65,3*175,0*30,3*150,0*45,3*105,0*30,3*85,0*10,1*175,0*30,1*150,0*45,1*105,0*30,1*85,0*125: | 7 | 0*725,1*205,0*30,1*210,0*45,1*220,0*5; |
|   | block(GetData), 0.0,1: |   | block(GetData), 120960,1: |   | block(GetData), 241920,1: |   | block(GetData), 362880,1: |   | block(GetData), 483840,1: |   | block(GetData), 604800,1: |   | block(GetData), 725760,1: |   | block(GetData), 846720,1:
block(GetData), 967680, 1:
block(GetData), 1088640, 1:
block(GetData), 1209600, 1:
BGOper, 3, 1, 300:
ProbeOper, 4, 1, 300;
"dstats.dat";
"QRanks";
"Counters";
1, vdata, "simdat", , free, REW:
2, PQ, "ProbeQgrowth", , free:
3, WIPP, "WIP.asc", , "(1X, E14.8, 1X, E14.8)";
4, WIPPWAF, "WIP.WAF.asc", , "(1X, E14.8, 1X, E14.8)";
5, LOTCYCTIME, "cyctime.asc", , "(1X, E14.8, 1X, E14.8)";
, , 655200, yes, yes, 241920;
APPENDIX 5

COMPARISONS BETWEEN NEURAL NETWORKS
AND OTHER METHODS
Comparisons such as those found in Chapter Five are presented in this appendix for the reader. Information shown in this appendix details the output obtained from the simulation model when...

1. the neural network was used to suggest inputs
2. best guesses were used to suggest inputs
3. random guesses were used to suggest inputs

In all cases an error was computed by comparing the actual simulation output and the desired simulation output. In all, there were 20 sets of desired outputs which were used during experimentation. For each set the neural network's performance was compared to three best guesses and 5 random guesses. For the cases presented, the neural network performed the best in eleven. (Refer to Chapter Five for a description of the experiments performed.) The results of the 20 test cases are on the following pages.
<table>
<thead>
<tr>
<th></th>
<th>WIP</th>
<th>Cycle Time</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target</strong></td>
<td>850</td>
<td>1650</td>
<td>80</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>753</td>
<td>1728</td>
<td>71</td>
<td>30</td>
<td>27</td>
<td>3.05</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>738</td>
<td>1679</td>
<td>54</td>
<td>23</td>
<td>0</td>
<td>3.86</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>771</td>
<td>1752</td>
<td>57</td>
<td>23</td>
<td>0</td>
<td>4.11</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>3.91</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>5.02</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>4.015</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>3.91</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>2.93</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>6.07</td>
</tr>
<tr>
<td><strong>Target</strong></td>
<td>650</td>
<td>1550</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>715</td>
<td>1617</td>
<td>64</td>
<td>27</td>
<td>18</td>
<td>4.27</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>764</td>
<td>1744</td>
<td>55</td>
<td>23</td>
<td>5</td>
<td>6.43</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>760</td>
<td>1723</td>
<td>53</td>
<td>22</td>
<td>0</td>
<td>6.38</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>7.12</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>7.12</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>4.735</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.23</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>5.33</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>9.19</td>
</tr>
<tr>
<td>Target</td>
<td>WIP</td>
<td>CycleTime</td>
<td>T-1 Util.</td>
<td>T-2 Util.</td>
<td>T-3 Util.</td>
<td>Error</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>Target</td>
<td>650</td>
<td>1550</td>
<td>20</td>
<td>80</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>715</td>
<td>1617</td>
<td>64</td>
<td>27</td>
<td>18</td>
<td>6.27</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>697</td>
<td>1625</td>
<td>61</td>
<td>26</td>
<td>0</td>
<td>6.97</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>10.1</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>860</td>
<td>1989</td>
<td>61</td>
<td>27</td>
<td>0</td>
<td>12.19</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>9.12</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>6.735</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>9.23</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>7.43</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>11.09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>1150</td>
<td>2050</td>
<td>80</td>
<td>80</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>692</td>
<td>1603</td>
<td>71</td>
<td>30</td>
<td>30</td>
<td>12.5</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>819</td>
<td>1848</td>
<td>75</td>
<td>37</td>
<td>0</td>
<td>9.73</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>748</td>
<td>1706</td>
<td>73</td>
<td>37</td>
<td>0</td>
<td>11.96</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>795</td>
<td>1800</td>
<td>73</td>
<td>37</td>
<td>0</td>
<td>10.55</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>12.68</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>14.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>11.77</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>11.77</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>8.81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>950</td>
<td>1750</td>
<td>60</td>
<td>60</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>753</td>
<td>1728</td>
<td>71</td>
<td>30</td>
<td>27</td>
<td>4.59</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>761</td>
<td>1712</td>
<td>67</td>
<td>31</td>
<td>0</td>
<td>5.07</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>4.71</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>788</td>
<td>1747</td>
<td>69</td>
<td>31</td>
<td>0</td>
<td>4.55</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>5.82</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>6.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>4.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.57</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>5.17</td>
</tr>
<tr>
<td>Target</td>
<td>WIP</td>
<td>Cycle Time</td>
<td>T-1 Util.</td>
<td>T-2 Util.</td>
<td>T-3 Util.</td>
<td>Error</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>850</td>
<td>1650</td>
<td>60</td>
<td>60</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>805</td>
<td>1744</td>
<td>62</td>
<td>59</td>
<td>22</td>
<td>2.44</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>726</td>
<td>1723</td>
<td>67</td>
<td>31</td>
<td>6</td>
<td>5.77</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>792</td>
<td>1820</td>
<td>63</td>
<td>27</td>
<td>0</td>
<td>6.08</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>737</td>
<td>1636</td>
<td>59</td>
<td>26</td>
<td>8</td>
<td>4.62</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>6.82</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>5.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>5.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.73</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>6.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>Cycle Time</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1050</td>
<td>1650</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>753</td>
<td>1728</td>
<td>71</td>
<td>30</td>
<td>27</td>
<td>6.45</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>697</td>
<td>1625</td>
<td>61</td>
<td>26</td>
<td>0</td>
<td>7.53</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>788</td>
<td>1819</td>
<td>56</td>
<td>25</td>
<td>0</td>
<td>7.86</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>738</td>
<td>1679</td>
<td>54</td>
<td>23</td>
<td>0</td>
<td>6.96</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>8.02</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>6.915</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.73</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>8.27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>Cycle Time</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>950</td>
<td>1550</td>
<td>60</td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>673</td>
<td>1520</td>
<td>55</td>
<td>24</td>
<td>5</td>
<td>6.27</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>784</td>
<td>1780</td>
<td>57</td>
<td>24</td>
<td>0</td>
<td>7.31</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>765</td>
<td>1763</td>
<td>61</td>
<td>26</td>
<td>0</td>
<td>7.33</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.71</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>8.02</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>6.235</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.53</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>9.27</td>
</tr>
<tr>
<td>Target</td>
<td>WIP</td>
<td>CycleTime</td>
<td>T-1 Util.</td>
<td>T-2 Util.</td>
<td>T-3 Util.</td>
<td>Error</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>1950</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>750</td>
<td>1730</td>
<td>71</td>
<td>30</td>
<td>27</td>
<td>7.8</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>8.76</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>9.44</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>7.75</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>9.78</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>11.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>9.29</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>8.77</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>7.89</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>650</td>
<td>1550</td>
<td>60</td>
<td>60</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>715</td>
<td>1617</td>
<td>64</td>
<td>27</td>
<td>18</td>
<td>3.27</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>726</td>
<td>1723</td>
<td>67</td>
<td>31</td>
<td>0</td>
<td>5.29</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>860</td>
<td>1989</td>
<td>61</td>
<td>27</td>
<td>0</td>
<td>9.19</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>788</td>
<td>1747</td>
<td>69</td>
<td>31</td>
<td>0</td>
<td>6.25</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>6.92</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>4.535</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>6.23</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.43</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>8.09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>750</td>
<td>1650</td>
<td>80</td>
<td>80</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>692</td>
<td>1603</td>
<td>71</td>
<td>30</td>
<td>30</td>
<td>4.5</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>748</td>
<td>1706</td>
<td>73</td>
<td>37</td>
<td>0</td>
<td>4.08</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>795</td>
<td>1800</td>
<td>73</td>
<td>37</td>
<td>0</td>
<td>5.45</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>720</td>
<td>1551</td>
<td>77</td>
<td>35</td>
<td>0</td>
<td>4.69</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>6.92</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>5.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>5.43</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.13</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>7.89</td>
</tr>
<tr>
<td>Target</td>
<td>WIP</td>
<td>CycleTime</td>
<td>T-1 Util.</td>
<td>T-2 Util.</td>
<td>T-3 Util.</td>
<td>Error</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>1050</td>
<td>1580</td>
<td>60</td>
<td>80</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>796</td>
<td>1825</td>
<td>65</td>
<td>28</td>
<td>17</td>
<td>7.29</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.82</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>7.86</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>7.55</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>8.82</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>7.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.83</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>8.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1150</td>
<td>1580</td>
<td>60</td>
<td>60</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>798</td>
<td>1832</td>
<td>65</td>
<td>28</td>
<td>17</td>
<td>5.7</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>7.56</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>6.4</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>5.55</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>7.68</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>9.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.57</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>7.57</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>6.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>950</td>
<td>1580</td>
<td>60</td>
<td>60</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>805</td>
<td>1744</td>
<td>62</td>
<td>58</td>
<td>22</td>
<td>3.49</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>5.72</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>6.86</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>6.55</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>7.82</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>6.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>6.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>5.73</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>7.17</td>
</tr>
<tr>
<td>Target</td>
<td>WIP</td>
<td>CycleTime</td>
<td>T-1 Util.</td>
<td>T-2 Util.</td>
<td>T-3 Util.</td>
<td>Error</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>650</td>
<td>1650</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>691</td>
<td>1604</td>
<td>71</td>
<td>30</td>
<td>30</td>
<td>3.42</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.34</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>7.1</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>6.93</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>6.12</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>4.515</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>6.23</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.33</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>8.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>850</td>
<td>1750</td>
<td>80</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>795</td>
<td>1800</td>
<td>73</td>
<td>36.5</td>
<td>0</td>
<td>2.975</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.41</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>3.11</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>4.4</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>3.77</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>5.015</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>2.66</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>4.42</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>4.82</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1150</td>
<td>1650</td>
<td>60</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Neural Network</td>
<td>753</td>
<td>1728</td>
<td>71</td>
<td>30</td>
<td>27</td>
<td>6.45</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.72</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>7.86</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>7.55</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>8.82</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>7.715</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.73</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>8.27</td>
</tr>
<tr>
<td>Target</td>
<td>WIP</td>
<td>CycleTime</td>
<td>T-1 Util.</td>
<td>T-2 Util.</td>
<td>T-3 Util.</td>
<td>Error</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>Neural Network</td>
<td>691</td>
<td>1604</td>
<td>71</td>
<td>30</td>
<td>30</td>
<td>2.32</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>2.74</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>5.9</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>5.33</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>5.92</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>4.315</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>4.43</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>2.73</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>6.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>750</td>
<td>1728</td>
<td>71</td>
<td>30</td>
<td>27</td>
<td>7.18</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.82</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>7.86</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>7.55</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>8.02</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>6.915</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>7.71</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>6.83</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>8.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target</th>
<th>WIP</th>
<th>CycleTime</th>
<th>T-1 Util.</th>
<th>T-2 Util.</th>
<th>T-3 Util.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>798</td>
<td>1832</td>
<td>65</td>
<td>28</td>
<td>18</td>
<td>8.45</td>
</tr>
<tr>
<td>Best Guess 1</td>
<td>731</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>10.26</td>
</tr>
<tr>
<td>Best Guess 2</td>
<td>812</td>
<td>1823</td>
<td>62</td>
<td>27</td>
<td>0</td>
<td>9.1</td>
</tr>
<tr>
<td>Best Guess 3</td>
<td>819</td>
<td>1879</td>
<td>66</td>
<td>28</td>
<td>19</td>
<td>8.35</td>
</tr>
<tr>
<td>Random Guess 1</td>
<td>755</td>
<td>1807</td>
<td>52</td>
<td>22</td>
<td>0</td>
<td>9.08</td>
</tr>
<tr>
<td>Random Guess 2</td>
<td>730</td>
<td>1611</td>
<td>52</td>
<td>23</td>
<td>2.5</td>
<td>11.215</td>
</tr>
<tr>
<td>Random Guess 3</td>
<td>776</td>
<td>1757</td>
<td>68</td>
<td>30</td>
<td>0</td>
<td>10.57</td>
</tr>
<tr>
<td>Random Guess 4</td>
<td>730</td>
<td>1713</td>
<td>67</td>
<td>28</td>
<td>21</td>
<td>10.27</td>
</tr>
<tr>
<td>Random Guess 5</td>
<td>846</td>
<td>1958</td>
<td>62</td>
<td>41</td>
<td>0</td>
<td>10.27</td>
</tr>
</tbody>
</table>
REFERENCES


Caudill, M., "Neural Net Adds Smarts to Spreadsheets, Slowly": BYTE, Jan. 1993, 221-23.


Glasserman, P., Structural Conditions for Perturbation Analysis Derivative Estimation, II:


