SANDIA REPORT
SAND2000–8200
Unlimited Release
Printed October 1999

Final Report on LDRD Project: Simulation/Optimization Tools for System Variability Analysis

R. L. Bierbaum, R. F. Billau, J. E. Campbell, K. D. Marx, R. J. Sikorski, B. M. Thompson, S. D. Wix

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: http://www.ntis.gov/ordering.htm

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A08
Microfiche copy: A01

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND2000-8200
Unlimited Release
October 1999

Final Report on LDRD Project: Simulation/Optimization Tools for System Variability Analysis

R. L. Bierbaum and K. D. Marx
Reliability and Electrical Systems Department

R. F. Billau
Software Integration and Technology Department

J. E. Campbell and B. M. Thompson
Systems Reliability Department

R. J. Sikorski
Engineering and Manufacturing Software Department

S. D. Wix
Component Information and Models Department

Sandia National Laboratories
Livermore, CA
Albuquerque, NM

Abstract:
This work was conducted during FY98 (Proposal Number 98-0036) and FY99 (Proposal Number 99-0818) under the auspices of the Sandia National Laboratories Laboratory-Directed Research and Development (LDRD) program.

Electrical simulation typically treats a single data point in the very large input space of component properties. For electrical simulation to reach its full potential as a design tool, it must be able to address the unavoidable variability and uncertainty in component properties. Component variability is strongly related to the design margin (and reliability) of the end product. During the course of this project, both tools and methodologies were developed to enable analysis of variability in the context of electrical simulation tools. Two avenues to link relevant tools were also developed, and the resultant toolset was applied to a major component.

Acknowledgments

We would like to thank the following for contributions to this work: Mike Deveney, 1734, Carolyn Bogdan, 1734, and Albert Nuñez, 1734 provided semiconductor device models. Wendel Archer, 1733 provided magnetic core models and laboratory data on the firing set transformers.

Simulations carried out on the MPACTR and CPlant computer systems were made possible through the work of Michael Johnson, 8114 and Rob Armstrong, 8980, respectively.

Thanks also go to Elmer Collins, Department 12335, and Tom Brown, Department 12333, for their helpful comments on interim versions of this report.

Table of Contents

Table of Figures

Introduction

This work was conducted during FY98 (Proposal Number 98-0036) and FY99 (Proposal Number 99-0818) under the auspices of the Sandia National Laboratories Laboratory-Directed Research and Development (LDRD) program. In addition this work proved to complement two other efforts, the Model-Based Reliability Analysis (MBRA) Project (funded through Surety Management, Case 0014) and the High Performance Electrical Modeling and Simulation (funded through ASCI, Case 5728). Where possible, an effort has been made to leverage the MBRA and HPEMS work.

Electrical simulation has been used successfully at Sandia over the past decade. Two common applications have been: (1) supporting the electrical system design process, and (2) analyzing anomalies detected in the fielded product. Simulation also has a future role in helping to predict failures when degradation precursors are identified. In short, electrical simulation is a tool that can be used during the entire product life cycle to develop an understanding of the system over the range of expected conditions.

Electrical simulation typically treats a single data point in the very large input space of component properties. For electrical simulation to reach its full potential as a design tool, it must be able to address the unavoidable variability and uncertainty in component properties. Component variability is strongly related to the design margin (and reliability) of the end product. This project addressed a set of problems that hinder effective use of simulation, especially in the context of understanding variability.

Objectives

The first goal of this effort was to create a novel linkage of disparate variability analysis approaches to greatly reduce the overall analysis time. The target tools were SUNS™ (Sensitivity and Uncertainty Analysis) and GO™ (Genetic Optimization), both of which were developed by Department 6411. A software copyright was obtained for both of these codes in FY99, and user's manuals are included in this report as Appendices A and B. The electrical simulation tool selected was SPICE (Simulation Program with Integrated Circuit Emphasis), a general-purpose circuit simulator that serves as the industry standard. There are many variants of SPICE available commercially. For this project, Berkeley SPICE 3f5 was selected because the source code is available. This version will be referred to henceforth as SPICE3, while SPICE will be used to refer to the general simulation tool.

There was also a need for methodologies to effectively apply this toolset to a range of problem types. The second goal was to develop an approach to quantify the reliability impacts of variability, especially for time-dependent issues. Ultimately, the benefits and limitations of the developed tools and methodologies were explored, including a consideration of the uncertainties and underlying assumptions. A specific application was explored, that of the W80 firing set. This will be discussed in detail in a later section. Note that in the area of methodologies, the LDRD and MBRA projects were quite complementary because of the strong reliance in MBRA on the ability to integrate variability analysis with modeling.

An extremely important aspect of this project was that of the user interface. The intent was to develop a tool that could be used by designers and analysts who were not necessarily experts in

the use of either electrical simulation tools or uncertainty analysis tools. As will be discussed in the next section, significant strides were made in this area by leveraging with on-going HPEMS work.

Tool Linkage

There were two generations of tool linkage in this project. These will be described below with some examples of how they were exercised.

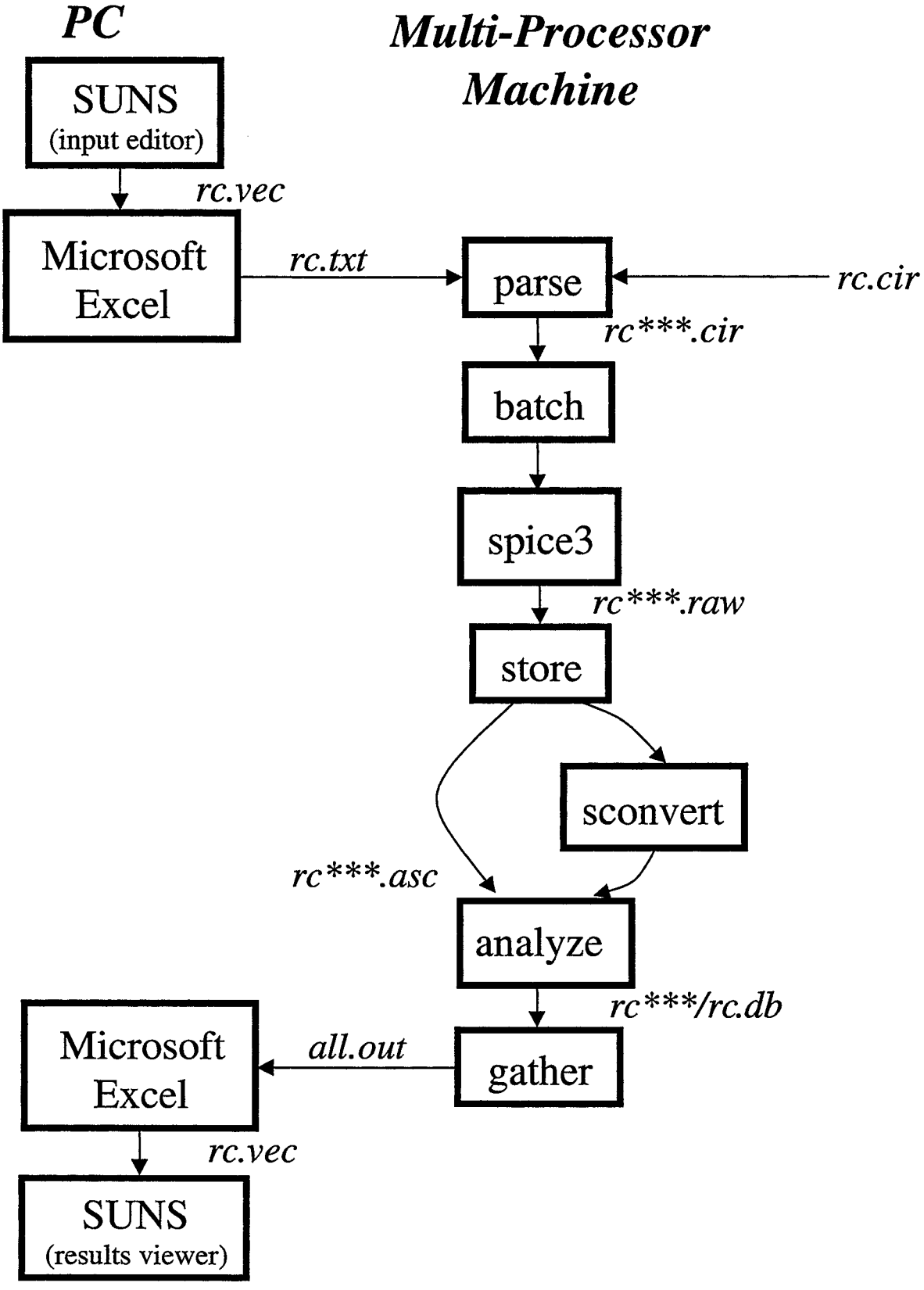
Stage 1 Tool Linkage

The first approach to tool linkage was predicated on the use of MultiSPICE, a process for running multiple simultaneous SPICE3 simulations on multiprocessor computer systems. The general concept was to use the SUNS™ tool to generate input vectors and to develop software to: (1) incorporate these vectors into a family of simulation code input files, (2) manage the execution of these input files, (3) extract relevant response variables from the resultant simulation code output files, and (4) build a SUNS™-compatible file such that the uncertainty analysis could be performed. A block diagram is shown in Figure 1. Each of the blocks is described below. Note that most of the code resides on the multiprocessor machine.

SUNS™ (input editor):
This is the SUNS™ Input Editor program. The user specifies the set of inputs (e.g., the part properties that describe the resistors, diodes, transformers, etc.) for which variability is to be analyzed. Nominal values, distribution types, and distribution parameters are then defined for each of the inputs. Correlations between inputs must also be specified. This is extremely important because failure to identify strong correlations could result in incorrect determination of the variability of the output variables. Given this information, SUNS™ generates a set of input vectors using either Monte Carlo sampling or Latin Hypercube Sampling (LHS). Each individual vector has a value for each of the inputs. The information on inputs is stored in a .sun file, and the input vectors are written to a .vec file.

Microsoft® Excel:
This serves an intermediary role between the .vec file and the parsing code ("Parse") that inserts parameter values into the simulation code input files. The Excel operations could be as simple as stripping off the header rows in the .vec file. However, in some instances more sophisticated operations are necessary. For example, one might want to choose among a set of alphanumeric input options, such as model file names. The SUNS™ input editor does not have this capability directly, so software is needed to map discrete input values to different model file names that can then be passed to the simulation code. This is straightforward to accomplish using one of the SUNS™ "Empirical Discrete" distribution types followed by an Excel operation. This distribution type selects between a set of integers, each of which corresponds to a specific model name; this allows one to "sample" between the various model file options. Note that one could write custom code to handle these scenarios, but it was considered to be a lower priority activity given the ability of Excel to manipulate and remap parameter values.

Figure 1: Stage 1 Tool Linkage



Parse:
This code takes the .txt file (the modified .vec file) plus a template simulation input file (.cir) and inserts the input values, creating a family of simulation input files. The template simulation input file includes all of the necessary information for the simulation code to execute it, other than the values of the inputs that are supplied by SUNS™. The "Parse" code gives each simulation input file a unique name.

Batch:
The "Batch" software submits the family of simulation input files to a multiprocessor machine. The code must be customized for the particular multiprocessor machine being used. MPACTR uses a Distributed Queuing System (DQS) to manage the execution of multiple simultaneous jobs, and thus it was straightforward to send the family of files off to SPICE3. Recently, the Computational Plant (CPlant) machine has been used. This system has a less mature queuing system and requires more user intervention to submit the simulation input files. It is anticipated that the CPlant queuing system will improve with time, requiring less direct management by the user and supporting a more seamless interface.

SPICE3:
The electrical simulation code used was SPICE Version 3f.5, developed at the University of California at Berkeley. SPICE3, while not as user-friendly as commercial versions such as PSpice©, is nonetheless a very useful simulator. It has the capability for modeling many electrical components (including the most common semiconductor devices), a rudimentary but workable graphical postprocessor, and some analog behavioral modeling capability. From the standpoint of the present work, its primary advantage was the fact that the source code had already been acquired by Sandia. Hence, it was not necessary to obtain licenses for the multiple instances of the executable necessary for running several dozen simultaneous simulations on multiprocessor computer systems. Its primary disadvantage was that it does not have the capability for modeling nonlinear magnetic cores, necessitating the in-house development of such models. As noted below, considerable effort is required to obtain realistic models that perform well. It is expected that the use of SPICE3 for this type of simulation will eventually be superseded by the implementation of the CHILESPICE and DSpice codes which are currently under development at Sandia.

Store:
This code retrieves the SPICE3 output files (.raw files) from each of the processors on the multiprocessor machine. As with the "Batch" operation, this is easier to accomplish for the MPACTR machine as compared to the CPlant machine, although the larger number of processors on CPlant (128 vs. 64 on MPACTR) is a significant step forward.

Sconvert:
One can select either binary or ASCII file output from the SPICE3 code. However, the output post-processing code, "Analyze", requires ASCII files as input. Thus when binary files are used, a conversion must be done and this is the function of the "Sconvert" code. The binary files are more compact (by roughly a factor of three) and thus desirable because of the reduced disk space required to store data for the large number of runs. In addition, binary files can be pulled off the processors more quickly. However, they have the undesirable feature of being unusable if the simulation code terminates prematurely due to lack of convergence. During the development phase, files have generally been stored in ASCII format because of convergence problems which

arise when running the simulations. In order to reduce data file size, only selected node signals have been collected rather than the entire set of output that is available. As the firing set model and the simulation code matures, binary format should become the norm.

Analyze:
The output of the electrical simulation code is a set of voltage and current signals as a function of time. In order to analyze variability, key characteristics must be extracted from these waveforms. The characteristics of interest tend to be problem-specific. They may be maximum or minimum values, rise times, ripple levels, etc. Sometimes performance analysis is very complex. For example, proper initiation of a detonator relies upon deposition of adequate energy in the bridgewire within a certain amount of time. This will be discussed in more detail later in this section for a specific application. Data analysis for situations like this is generally not available in a commercial software package. To date, custom modules have been written to extract the outputs of interest. Many of these custom modules will have utility for a wide class of problems. However, the development of a truly general output extraction suite (one of the original goals of this LDRD) has proven to be elusive because of the broad range of electrical simulation analyses that are performed.

Gather:
This code simply collects the set of outputs that were extracted by the "Analyze" code and puts them into a single file.

Microsoft® Excel:
Excel is used to paste the file of extracted outputs into the original .vec file. This file is then passed back to the SUNS™ Results Viewer code. Note that this step could be incorporated into the Gather code if desired.

SUNS™ (results viewer):
The user has numerous options for analyzing and viewing the results: correlations (simple raw, simple rank, partial raw, and partial rank), output and input variable histograms and cumulative distribution functions, scatter plots, and tables of values. Examples of some of these will be shown later in the report, and they are also described in the SUNS™ User's Reference Manual.

MultiSPICE and computing platforms:
Besides the SPICE3 simulator running on the multiprocessor computer, the MultiSPICE system includes the auxiliary elements of software Parse, Batch, Store, Sconvert, Analyze, and Gather. These are discussed in the descriptions above. It should be noted that MultiSPICE performs uncoupled (i.e., serial) simulations on multiprocessor machines (rather than parallel computations).

The first implementation of MultiSPICE used the Massively Parallel Computer Resource (MPACTR) located in California. It consists of an array of from 9 to 14 computers, each of which contained four Pentium 200 processors. From 32 to 52 processors were available on MPACTR for MultiSPICE simulations.

More recently, MultiSPICE has been set up on the Computational Plant (CPlant) computer system. Up to 128 DEC Alpha computers have been available for simulation. As many as 64 simultaneous runs have been executed in the completion of the present work. These simulations

were completed in approximately 24 hours on the CPlant system (each simulation required up to 24 hours of CPU time, but they all ran simultaneously). In the case where results were obtained from 101 simulations (discussed in the "W80 Firing Set Application" section), the entire job was completed by utilizing 42 processors over a weekend. It should be noted that such large numbers of simulations could not have been carried out in any practical manner if it were necessary to do them sequentially on personal computers. The great improvement in turnaround time achieved by the use of multiprocessor systems made the work possible.

Application of Stage 1 Tool Linkage

The initial application to exercise the Stage 1 Tool Linkage approach was a very simple analysis of a Resistor-Capacitor (RC) filtering circuit. The problem was set up such that variability was specified for both the resistor and capacitor. The goal was to use the SUNS™/SPICE3 toolset to determine the variability of the key response variable, the 3 dB cut-off frequency which is a common characteristic of interest for filter circuits. This simple problem (solvable analytically) was intended to provide a vehicle for evaluating the integrated tools. The results were very promising in that a relatively seamless toolset was put together to perform this analysis, although it did require some custom code.

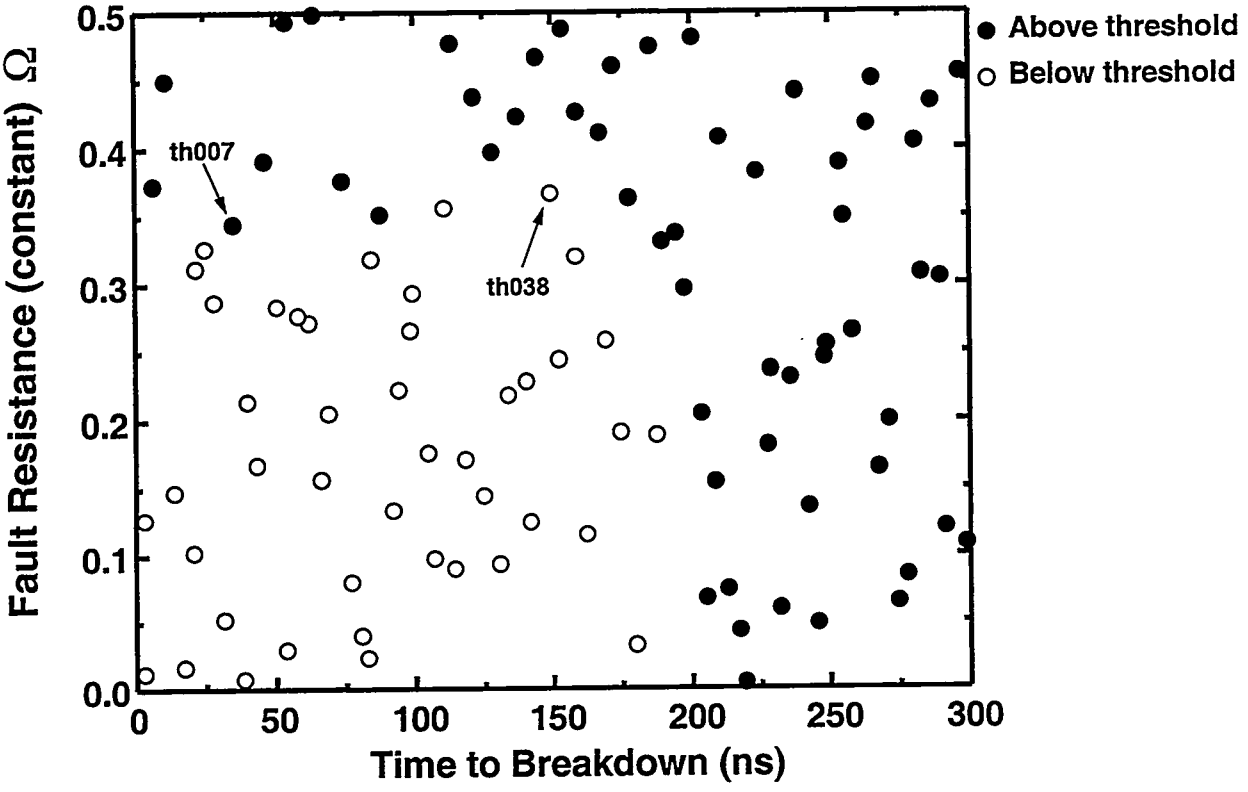
The next problem was of more interest to the stockpile. There have been instances of high voltage breakdown observed in firing sets. Using SPICE3, the characteristics of the breakdowns that are germane to system performance (breakdown time and breakdown resistance) can be modeled. The values of breakdown time and breakdown resistance were derived for actual instances of breakdown during testing. While these instances did not lead to system failure, there was concern that under certain conditions breakdown could cause failure. The SUNS™/SPICE3 toolset was used to develop a performance map as a function of these characteristics.

As mentioned earlier, the issue of detonator initiation is complex. Detonator function is characterized by a threshold current and energy; when a detonator is being initiated, the current must remain above this threshold level until adequate energy has been deposited to initiate it. If energy is deposited too slowly, the detonator will not initiate properly. The threshold current represents the level at which, when adequate energy is deposited, half the detonators would fire properly and half wouldn't. From a modeling perspective, the detonator is treated as a nonlinear resistance as a function of the energy deposited in it. The magnitude of the current flowing through the detonator is squared and integrated using SPICE3. The simulation code monitors the current to see if it remains above the threshold current level until adequate energy has been deposited.

For this problem, SUNS™ was used to generate one hundred input vectors that covered the breakdown time/breakdown resistance state space. The SPICE3 code was then run to see if the threshold current condition was met. Figure 2 summarizes the results of these runs. The approximate regime of concern appears to be breakdowns that have a breakdown time of less than 200 ns or a breakdown resistance of less than 350 mΩ. Of course, this is based upon the assumptions inherent in the model, such as the breakdown inductance (here assumed to be constant across the state space at 1 nH) and the circuit parasitics included in the firing set model. Note that there appears to be a modestly positive slope to the boundary between above and below threshold for breakdown times of less than 200 ns. This is not an intuitive result. We speculate

that "early" breakdowns result in increased current in the circuit and that energy is stored in the parasitic inductances to maintain the current. Later breakdowns allow for less energy storage. The inductances for runs th007 and th038 (indicated in Figure 2) were reduced in subsequent runs, resulting in behavior more consistent with expectations (i.e., a modestly negative slope).

Figure 2: SUNS™/SPICE3 Analysis of Detonator Performance



100 samples taken from uniform distribution
Fault inductance = 1 nH
Low-current model used to better simulate threshold behavior
Runs th007 and th038 studied to investigate non-intuitive results

This example typifies the role that uncertainty analysis plays. It shows performance over a broad range of conditions, it highlights unusual behavior, and it provides focus for follow-on analyses. This example also pointed out one of the challenges of modeling performance in the context of reliability. The threshold current represents a failure probability of fifty percent for the detonators, clearly an unacceptable level for weapons. Trying to ascertain true success/failure criterion for detonator modeling is an enormous task. Instead, testing under realistic and diverse conditions is used to estimate the reliability.

In general, the Stage 1 Tool Linkage worked very well. It allows for significant flexibility and control by the analyst. It is clear that much insight can be gained from this type of analysis approach. The tradeoff is that some customization and user interaction are required to exercise the toolset.

Stage 2 Tool Linkage

The opportunity arose in FY99 to capitalize on the development of a "designer/analyst" graphical user interface (GUI) being developed for ASCI. This interface effectively shields the user from the intricacies of the SPICE electrical simulation tool but provides enough control to the user to perform meaningful analyses. This provided an opportunity to enhance the LDRD work to date in the following areas:

(1) The interface is online, graphical, and easy to use, focusing on changes to key variables. It is a web-based interface, which is much easier to use than the usual SPICE interface.

(2) It allows for quick problem setup and viewing of results.

(3) It offers the potential for a more general interface solution, albeit at a cost of some user flexibility initially.

The SUNS™/SPICE Design Applet Tool provides users with the capability of performing "what if" analyses on a circuit design. Using a SPICE netlist for a power management circuit for an existing weapon system, a user can download an applet using Netscape (4.08 or higher) and change the values and/or models for existing transistors, diodes, resistors, and capacitors, as well as voltage and current sources. Optionally, the user can vary the range of values for a resistor or capacitor between some interval (i.e., high / low values), and submit a SUNS™/SPICE run to analyze the effects of the changes for a given number of trials.

The software generates a netlist for each trial and automatically runs SPICE for each trial run on a UNIX Server using SPICE Server software written in Java. The SPICE Server software on the UNIX workstation communicates with SUNS™ on an NT 4.0 workstation to execute various processes of the SUNS™ software to edit the inputs and analyze the results of the SPICE runs (this software is also written in Java). Upon completion the user, through the design Applet on their workstation, can view plots generated by SPICE and the statistical output graphs generated by SUNS™. Ultimately it is desire to execute the simulation runs on a multiprocessor platform such as CPlant using CHILISPICE.

Screen shots of the user interface are shown in Figures 3 through 5. Figure 3 shows the circuit schematic and the means by which input values are selected. Note that the tolerance value for each input can be selected here. Figure 4 shows the SUNS™ interface. This is where the input distribution is selected. Ultimately, the capability to set input correlations will be added to this screen. Finally, a selected output from the SPICE run is shown in Figure 5.

Figure 3: SPICE User Interface

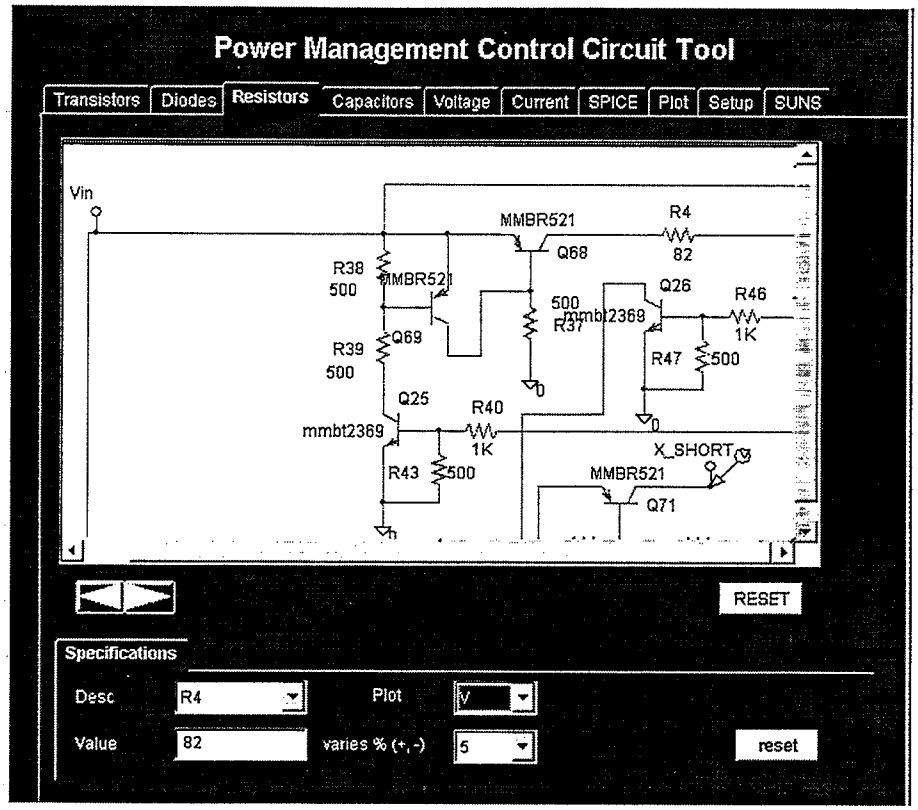


Figure 4: SUNS™ User Interface

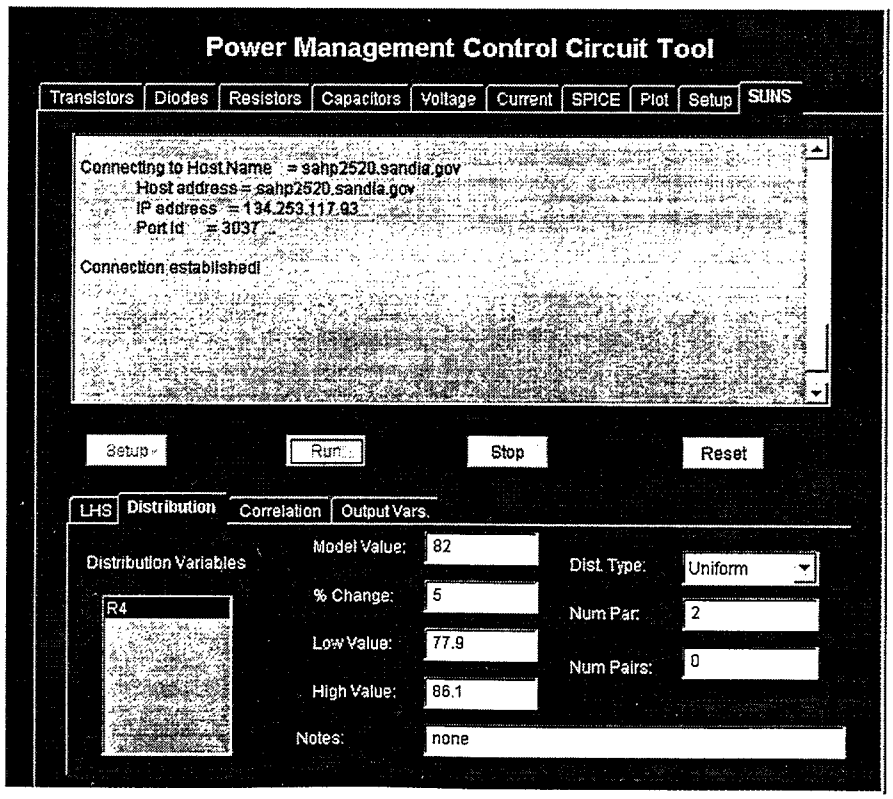
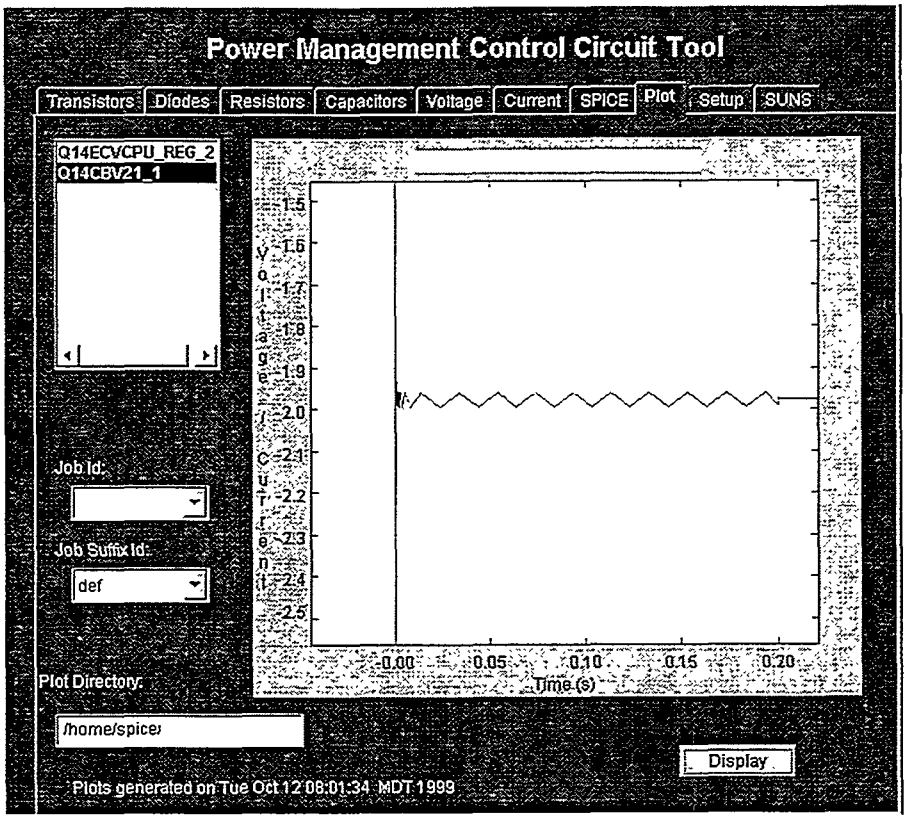


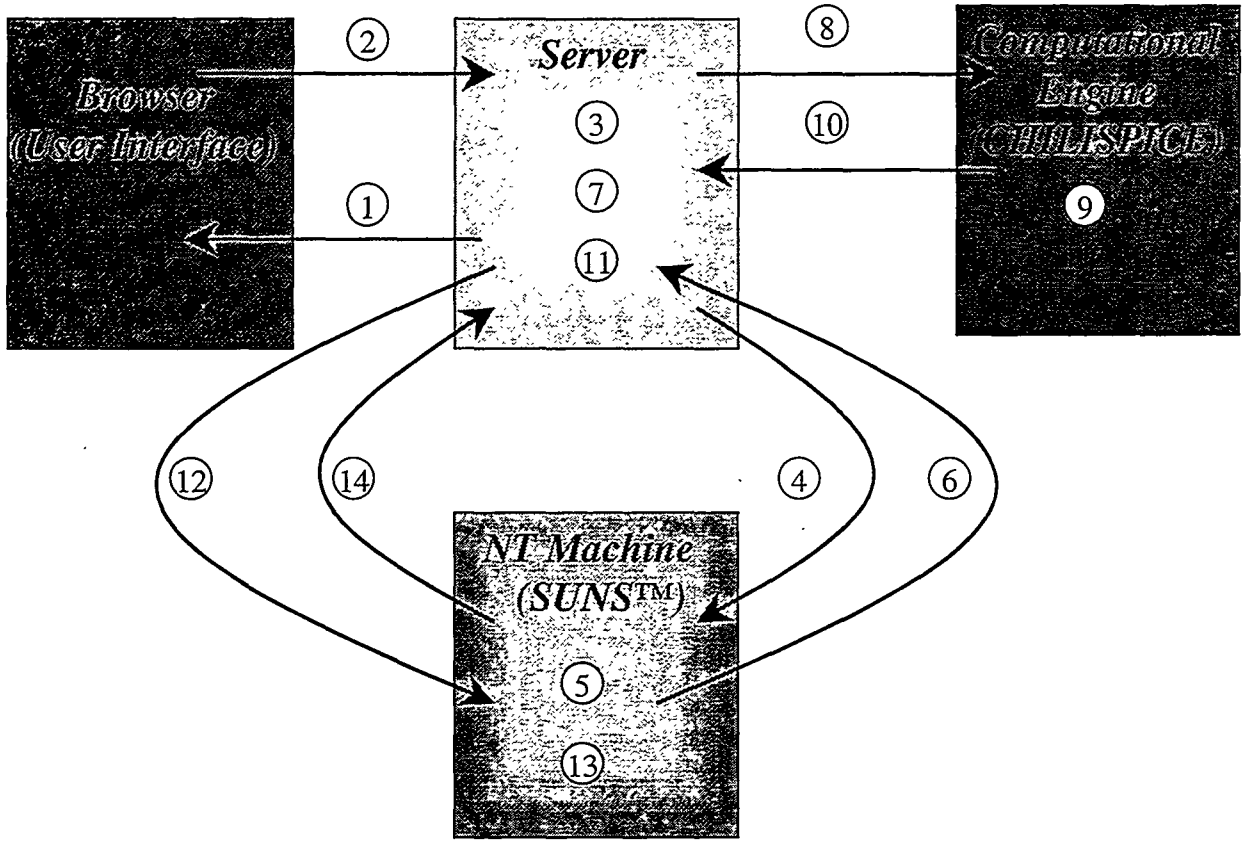
Figure 5: SPICE Output Display



A notional block diagram of the desired process is shown in Figure 6. The data flow is as follows:

(1) Applet downloads from Server; user constructs desired problem
(2) User submits job
(3) Server builds .sun file and initial netlist
(4) .sun file sent to SUNS™ platform
(5) SUNS™ builds .vec file
(6) .vec file passed to Server
(7) Server builds set of netlists from .vec file (the parsing code is included as Appendix C)
(8) Server submits set of netlists to computational engine
(9) Computational engine performs multiple SPICE analyses
(10) Analysis results sent to Server
(11) Server modifies .vec file to include computational results
(12) Server sends .vec file to SUNS™ platform
(13) SUNS™ platform generates standard set of results; notifies user via email results are ready
(14) SUNS™ platform publishes results to web site on Server for user analysis

Figure 6: Stage 2 Tool Linkage Block Diagram



The data flow is very similar to the Stage 1 process. However, the architecture will initially be very restrictive. In SUNS™, only passive components will have variability associated with them. There will be a limited number of component tolerance options. Correlations between input variables will be assumed to be negligible. For SPICE, the circuit topology will be fixed. Limited options for output variable extraction will be available. However, it will be very easy to use and the data flow will be transparent to the user. The goal is to continue this effort in FY00 to expand the capabilities available to the user without sacrificing ease of use.

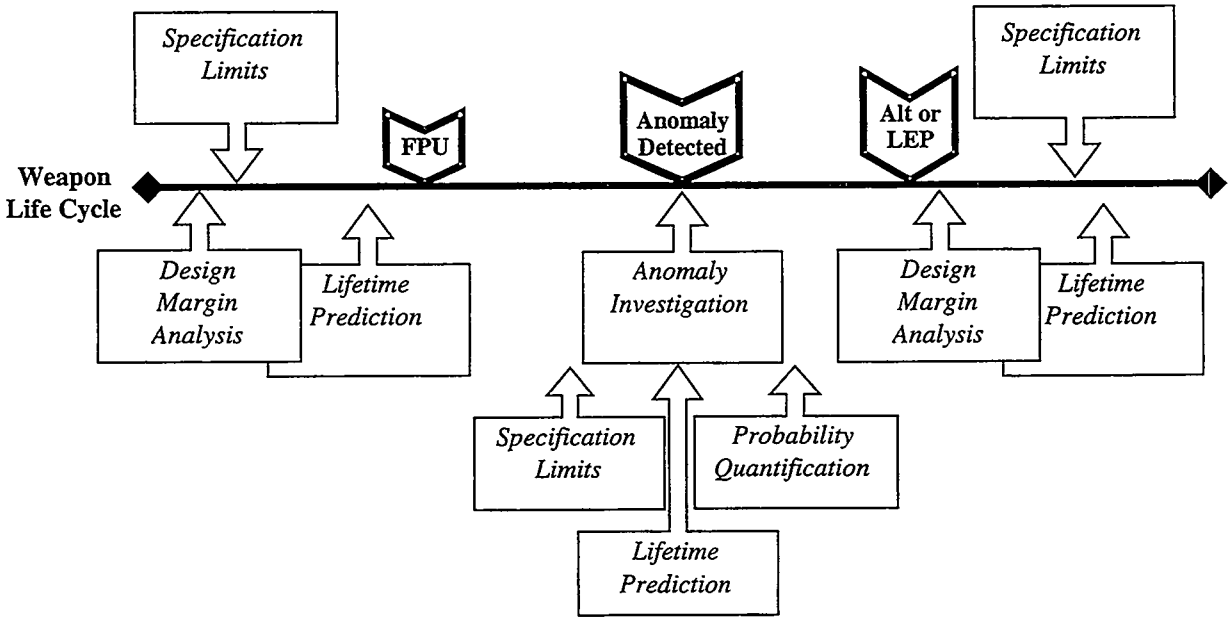
There are many approaches that could be used to expand analysis capabilities while retaining ease of use. One option is to incorporate an "Analysis Advisor" software tool that aids in problem set-up and execution. Successful use of modeling is made difficult by the scope of knowledge required: basic phenomenology, multi-disciplinary modeling, computational capability, analysis methodologies, etc. Incorrect use of the tools can lead to misleading conclusions, a serious issue in the area of surety because of the consequences of error. An Analysis Advisor would aid the surety analyst in using modeling appropriately, both in tool selection and interpretation of results. The Advisor framework will be developed and populated with knowledge in selected modeling areas to permit evaluation. The initial focus will be on advising on rudimentary issues for which written "rules of thumb" would suffice. For example, performing an uncertainty analysis requires the analyst to make critical choices about the sources and nature of variability, variable correlations, what sampling approach to use, and the sample size required. The Analysis Advisor would provide guidance and recommendations on these issues, either on request or in response to user inputs. More sophisticated capabilities such as

selection of appropriate computational resources (vis-à-vis problem class and size) can then be added. In its most advanced state, the Analysis Advisor could extract features from simulation code outputs (using neural networks, for instance) to highlight key characteristics. It is envisioned that projects such as ASCI HPEMS and MBRA would also incorporate the Analysis Advisor.

Methodologies[1]

In this section, several applications for the SUNS™/SPICE will be described. Note that these span much of the nuclear weapon life cycle (Phases 3 through 6, from development through the lifetime of the weapon), as shown below in Figure 7. The bold chevrons depict points in the weapon life cycle where it is anticipated that the tools will be particularly helpful. The rectangles with text in italics show specific processes that are being developed.

Figure 7: Analyses as a Function of Weapon Lifetime Phase



Each of the five processes shown above (Design Margin Analysis, Specification Limits, Lifetime Prediction, Anomaly Investigation, and Probability Quantification) will be discussed in the context of their purpose (as a function of life cycle), current implementation, the proposed new approach, and the benefits of the new approach.

An important point here is that the processes below are all predicated on the development (or existence) of validated models. These models must balance adequate fidelity and level of detail with execution time; this is an important consideration since sensitivity and uncertainty analyses may require tens to hundreds of simulation runs. Depending on the issues being explored, the models could range from piece-part level to system-level in scope. Obviously, developing and validating such models (or tailoring them as needed) is an expensive enterprise requiring keen engineering judgment to ensure that the models are appropriate for a broad range of applications.

[1] Much of this information is also included in the MBRA Interim Report.

However once this investment has been made, the models provide an extremely valuable resource, and their use can be leveraged over the life cycle of the weapon. There is also a significant amount of understanding that is accrued during the course of modeling a subsystem or system. This is extremely beneficial as weapons continue to age and their designers retire.

Design Margin Analysis

The Design Margin Analysis process has perhaps the lowest technical risk of any of the processes but potentially a large payoff. It can be used at almost any stage of the weapon life cycle when there is a desire to either quantify the range of possible performance of a system or subsystem or improve the performance of a system or subsystem. As will be seen in subsequent sections, Design Margin Analysis serves as a foundation for each of the other processes.

Current Process

Design margin has historically been determined (and improved) using an iterative process. A design is generated and development hardware is built and characterized. The amount of design margin is of course dictated by the requirements, but generally these continue to be negotiated during the design and development phase as the actual capabilities of the hardware become clearer. The requirements for one subsystem are driven by the interface requirements of adjacent subsystems, and thus the process of finalizing them can be prolonged as the design evolves. As the time for First Production Unit (FPU) nears, product acceptance specifications are documented and implemented. Product performance data are analyzed and the production processes are revised. Generally the drivers for these changes are too much rework, processes that are time-inefficient, and lower-than-desired yield. In addition, failures and anomalies are evaluated to identify causal factors – these may also result in changes to the production process or part selection process.

The drawbacks to the current process are as follows:

(1) The process can be costly, since it may require several iterations of development hardware until the desired performance is achieved.

(2) Variability is very difficult to determine because of the limited quantities of hardware that can be built and characterized.

(3) The on-going modifications to the production processes following FPU lead to non-homogeneous product; this complicates the job of surveillance and stockpile evaluation.

(4) Specifications may be set too stringently for a variety of reasons. For example, the observed variability is small and therefore those limits are deemed to be easily obtained from the product; thus the limits may be set tighter than necessary.

Proposed New Approach

It is proposed to use SUNS™/SPICE to both quantify and improve design margin. The key difference in the new approach is that modeling is used to explore the state space in a systematic

or impossible to control. In addition, sometimes it is impossible to make the necessary measurements to fully characterize a subsystem without affecting the subsystem's performance.

The proposed process is described below. Note that the approach would be similar for other modeling disciplines and could be extended to multi-disciplinary modeling when technically and computationally feasible; this is one of the goals of the MBRA project. The process is as follows:

(1) Develop and validate a model of the electrical subsystem using an electrical simulation tool.

(2) Identify the circuit parameters that are suspected to contribute to the output and its variability. This involves engineering judgment in conjunction with limited experimentation, although the process itself will allow identification of the most important among these. Determine the anticipated variability in each of the selected parameters. Note that this might include variability due to different environments, such as temperature and total dose radiation, in addition to variability from sources such as manufacturing.

(3) Use SUNS™ to generate a set of input vectors that will explore the state space thoroughly. There are many nuances to this, which must be documented such that the user does not generate meaningless or misleading results.

(4) Execute SPICE using the values in the input vectors. Perform data post-processing to extract the relevant outputs.

(5) Using SUNS™, identify the key contributors to the outputs and the variability of the outputs.

These results can be used in a number of ways. First of all, they can be used to identify meaningful specifications for the subsystem being analyzed. This will be described in more detail in the next section on Specification Limits. Secondly, they can be used to target improvements. Parameters that affect the output variability most can perhaps be screened to minimize the variability they contribute; alternatively, other parts can be selected that demonstrate more stable performance. And finally, this analysis may set the stage for an optimization study once the key contributors to the outputs have been identified.

The expected benefits are as follows:

(1) Development time and required test assets will be reduced. In addition, there will be fewer design iterations necessary.

(2) Because of the increased understanding obtainable by modeling, it is expected that the yield will be better at FPU that it would have been without the use of the toolset.

(3) Fewer modifications in the production processes will be needed, leading to a more homogeneous product.

(4) These tools can be used to evaluate the impact of production and part changes if they do become necessary.

(5) This process can provide focus for resource allocation, aiding in decisions about what parts to upgrade or screen. It also allows for intelligent selection of production acceptance specifications to optimize yield.

(6) Design margin will be improved. Although this may not be a quantifiable reliability improvement, it is obviously desirable and makes the subsystem more resilient to external changes and changes over time.

Note that there are some risks to the use of this process (listed below), although they are outweighed by the benefits of increased understanding:

(1) Not all parameters that influence output and output variability may be included in the analysis.

(2) The parameters that are included may not have their variability quantified correctly.

(3) Failure to identify correlations between input variables can lead to poor estimates of output variability.

Specification Limits

As discussed in the previous section, one of the variants of the Design Margin Analysis process is that of setting specification limits. This is an activity that could have significant payoff – the cost of overly conservative specification limits can be very high due to excess scrap and rework, unneeded inspection and screening, and selection of more expensive parts. Specifications that are not tight enough can lead to unexpected failure; even worse, these failures may not be easily detected in the surveillance program.

Current Process

Currently, specification limits for subsystems are negotiated using a combination of knowledge gained through development hardware characterization and the bounds dictated by the interface requirements. As already pointed out, this is an evolutionary process and can lead to overly stringent requirements because of the limited data available. Drawbacks of the current process are as follows:

(1) The current evolutionary process is costly because of the number of hardware development iterations that are necessary.

(2) Overly stringent specifications lead to numerous exceptions (SXRs) that must be evaluated.

(3) There is an important reliability implication to setting specifications. Because it is impossible to conduct full-up tests, the surveillance program must infer system success or failure by comparing subsystem performance to the specifications. There have been numerous Significant Finding Investigations where the focus of the investigation was understanding how realistic the specifications were and whether they really related to overall system success/failure. The subsystem may have failed the specification limit but

would not have precluded the system from working. This can be a very costly and time-consuming process.

Proposed New Approach

The Specification Limits process builds on the Design Margin process. The first step in establishing specifications is quantifying the design margin of the subsystem as described in the previous section. Using the information derived from that exercise, specification limits can be defined. First of all, the acceptable output level must be determined. This is based on both the interface and operational requirements. Then specification limits can be placed on the parameters, focusing primarily on those that contribute most to the output and the output variability. During this process, it must be determined if variability needs to be reduced (and if it can be reduced) or if additional margin is needed. If variability can be reduced, it may be possible to "give back" margin to the system (i.e., allow another subsystem to have greater variability).

The advantages of the new process are the same as mentioned in the previous section. In addition:

(1) The number of SXRs will be reduced. Those that remain will be more meaningful in the sense that they reflect a failure to meet a specification that is more closely tied to overall system success/failure.

(2) The number of Significant Finding Investigations where the subsystem failed the specification but would have been successful at the system level will be reduced.

Both of these have significant potential for reducing the cost of production and surveillance.

Note that there has been one example of a specification limit application for the stockpile. For Alt 339 for the B61, the MET (MCCS Encryption Translator) is being added to the units to allow for encrypted PAL operations. A study using modeling in conjunction with a Monte Carlo analysis was performed to determine if the existing specifications for input voltage could be met when the MET was added. The study examined both part parameter variability as well as variability due to temperature. The results indicated that the MET could be added without jeopardizing performance over the range of expected conditions. More importantly, a realistic set of specifications was developed for the B61 PAL subsystem during the course of this study[2].

Anomaly Investigation

Modeling is a natural choice for investigation of anomalies, and a variety of modeling tools have already been used for this application. It is a cost-effective way to perform root cause analysis, to examine in a general sense the effect of an anomaly, and to identify fixes.

[2] This study is documented in an unclassified memo, R.C. Elder, 2671, and R.I. Eastin, 8116, to Distribution, "Final Report for the MCCS Low Voltage Qualification", dated August 28, 1995.

Current Process

Although there is a formal process for investigating anomalies (Significant Finding Investigations, or SFIs), the specific elements of each investigation tend to be customized to address the problem at hand. A combination of testing and analysis is generally done, and resources across the lab are brought to bear, including modeling. For example, electrical simulation was used extensively on several W87 SFIs to examine firing set high voltage breakdown. While the modeling capabilities did not allow for true root cause analysis (i.e., identifying the actual source of the breakdown and the factors influencing it), it did allow for an examination of the effects of the breakdown and the impact of breakdowns with slightly different characteristics (location and timing). As discussed earlier, SUNS™/SPICE could be used to thoroughly explore the performance state space for several variables.

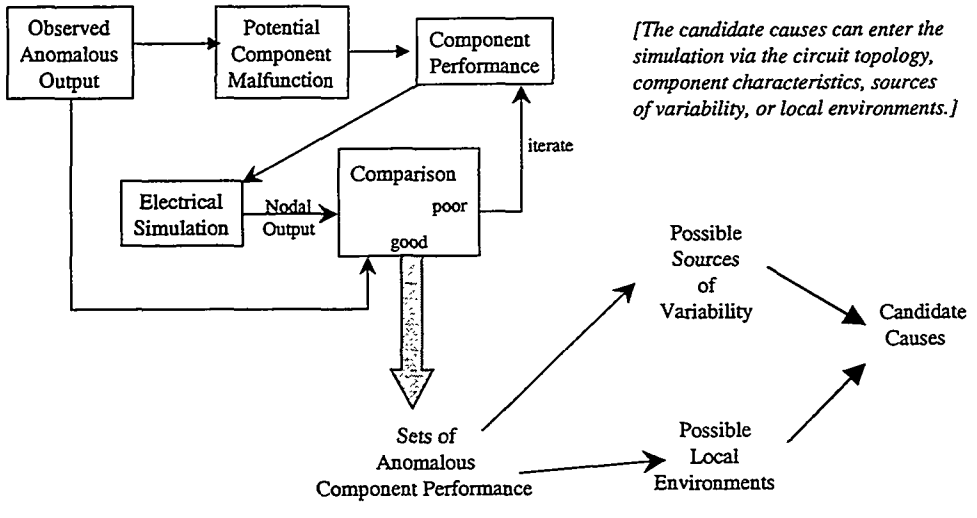
Proposed New Approach

The toolset offers a more formalized approach to modeling in the context of anomaly investigation, as well as integrated tools to perform the analyses. Figure 8 shows the proposed process. It consists of four steps:

(1) Develop a model with adequate detail (which may already exist) to emulate the anomalous results; this requires engineering judgment accompanied by hardware validation and is often an iterative process. Note that the optimization tool may accelerate model development by automating the process of varying parameters to match model outcomes to observed results from hardware testing.

(2) Postulate root causes and inject them into the model; compare with the observed anomaly and add to the list of possible sets of anomalous behavior if they are similar.

Figure 8: Anomaly Investigation Process



(3) After generating a set of anomalous behaviors, it is then necessary to determine their impact. It is important to recall that specific anomalies themselves may not lead to system failure. However, they may be indicators that other units similarly afflicted could fail due to

different environments or even different interface conditions. Thus a critical role of modeling is taking each anomalous behavior and developing a generalized view of its impact. As shown below, each type of anomalous behavior is explored vis-à-vis the sources of variability, the different environments that can be expected, and the interface conditions that might be experienced. Additionally, any time dependence must be examined; this will be discussed further in the next section. Finally, the relevance of this anomaly to other parts or technologies must be examined to identify any other impacts.

(4) The final step is quantifying the impact. Initially it is expected that the primary contribution to the anomaly investigation process will be qualitative. Issues surrounding probability quantification using SUNS™/SPICE will be discussed in more detail later in this report.

There are important advantages to this new approach:

(1) Modeling is a much less expensive way to gain insight than testing. Parameters are easily controlled and monitoring can be done unobtrusively. The ability to inject faults without otherwise affecting performance is particularly critical. Many iterations can be done quickly with a model while varying parameters and conditions based on a small number of validation tests. To gain the same understanding with testing alone could be very costly and time-consuming.

(2) Typically it has been challenging in the past to determine the generalized impact of an anomaly (i.e., how the observed anomalous behavior may manifest itself in other components, under different environments, and for other weapon capabilities). Use of a modeling framework allows one to easily inject the behavior under these different conditions and observe the consequences. This would be prohibitively expensive to do with hardware in most cases. It addition, it is difficult or impossible to test at some environmental extremes (for instance, shock and vibration at higher levels of assembly).

Lifetime Prediction

As shown in Figure 7, the need for a lifetime prediction capability spans the weapon life cycle. Lifetime is a key decision-making and resource allocation metric, and improved means of predicting it are critical as the Nuclear Weapons Complex continues to shrink.

Current Process

There are three important times when lifetime prediction is necessary. The first is during design and development. The most obvious issue is that of the need for Limited Life Component Exchanges (LLCEs) for those components that are known to have a limited life. A combination of testing and analysis has been done in the past to determine if LLCE was needed and when it would be needed. A secondary issue was that of identifying and performing an adequate qualification program such that there was some assurance that the weapon would meet its specified lifetime goal (often 20 years). This was done via accelerated aging tests, IC burn-in, part dissection to look for changes, and environmental cycling. This was not accompanied by a formal lifetime prediction, although the tests and conditions were selected to try to emulate a long life.

The second time when lifetime prediction is used is during investigation of an anomaly. It is necessary to determine if the anomaly is time-dependent. If it is, the expected failure probability as a function of time must be estimated such that a decision regarding corrective action can be made. Again, testing and analysis has been relied upon to answer these questions. Historically there have been instances where time dependent degradation was observed, but the time constant was so slow that it was deemed to not be a concern during the expected lifetime of the weapon. On the other hand, some anomalies have shown a degradation rate sufficiently large to motivate a retrofit. Because such unanticipated retrofits are costly and difficult, credible and accurate lifetime prediction is essential – and becoming more so as resources decrease.

The third instance where lifetime prediction is important is during planned retrofits; for example, those conducted to incorporate a new capability into the stockpile. The role of lifetime prediction is very similar to that done during initial design and development.

Proposed New Approach

The new approach capitalizes on the modeling of aging behaviors being done as a part of the Enhanced Surveillance Program. The approach will be hierarchical, building a bridge between the material models and the piece-part models in order to examine the effects of aging in a subsystem- or system-level context. This provides a framework to evaluate pervasive aging throughout the system (e.g., if multiple piece-parts are changing with time).

The steps in the process are as follows:

(1) Analyze the subsystem or system for key contributors using the Design Margin Analysis process. Both parameters prone to variation and parameters that are suspected to affect system output and variability should be included.

(2) Develop a model of behavior as a function of age. This may be physics-based or empirical. Furthermore, although it will often be at the piece-part level, it could be at any level of assembly (piece-part, printed wiring board, Major Component, Major Assembly, or system). One of the key assumptions in using SUNS™/SPICE to perform these analyses is that the aging models can be cast into the context of electrical parameter changes.

(3) Incorporate the models developed in step (2) for all critical age-related variables as identified in step (1) into an electrical subsystem or system model.

(4) Exercise the model as a function of time and examine the outputs to see if specifications are not met at some point in time. This will give a prediction of lifetime. Note that this could then be extended to examine reliability as a function of time.

The advantages of the new approach are as follows:

(1) There can be more informed choices during design of materials and parts, based on the modeling results. Critical parameters can be identified for on-going monitoring.

(2) There is a much better understanding of design margin as a function of age.

(3) The process can help identify if an age-related problem is a gradual degradation or a cliff.

(4) Parametric lifetime studies can be done with the Lifetime Prediction process.

(5) Cost/lifetime trade-off studies can be performed.

Probability Quantification

This is often considered to be the ultimate goal of modeling and simulation. It is the culminating step, but it is also carries with it the highest technical risk. It is important to realize that failure to achieve a generalized probability quantification approach using modeling does not preclude the major benefits accrued by the other processes.

Current Process

During design, a reliability prediction is performed. Since relevant performance data is typically not available, the prediction tends to be based on a comparison to similar piece-parts, Major Components, or systems that operate under similar environments. Design of Experiments might be used with development hardware to examine different options vis-à-vis margin and reliability.

The reliability prediction is used in a variety of ways. Early in the design, trade-off studies can be performed. If the overall reliability requirement is not met, the prediction provides a basis on which to allocate resources to improve it. As FPU approaches, the reliability prediction serves as an initial estimate of reliability for the system as it enters the stockpile and until sufficient data are available to make a data-based assessment.

Probability quantification is an important part of the anomaly investigation process. It lays the foundation for deciding whether or not to take corrective action. The current process is as follows:

(1) Following detection of the anomaly, analysis is performed to identify the root cause.

(2) A determination is made as to what test data are available to assess the impact. This requires a significant amount of engineering judgment, since each type of test (and associated test condition) must be evaluated to see if it would be capable of detecting the anomaly.

(3) If there is insufficient data to assess the impact, additional testing or analysis may be performed to supplement the information available to quantify the impact.

(4) Once sufficient data are available, they are used to quantify the impact (probability of failure equals the number of failures divided by the number of relevant tests).

There have been instances where this approach has been modified slightly. In some cases, it has been recognized that it is more meaningful to think of performance in terms of distributions rather than in terms of failure to meet a specification. The analysis is customized based on the specific anomaly, but the general approach is that probability of failure is determined by examining the overlap of distributions.

Proposed New Approach

Probability quantification builds on the other processes, most importantly the Design Margin Analysis. It is more demanding, though, in the sense that one can tolerate uncertainty in design margin analysis because the goal is typically understanding relative (rather than absolute) margin. For probability quantification, it is essential to minimize uncertainty in the results and to ensure that all salient factors are included. The results must be highly credible when major allocation of resources and potential disruption to the Nuclear Weapons Complex are at stake. It is anticipated that initially the process for probability quantification will be used sparingly or in conjunction with other methods until its credibility is validated. Of course there will be instances where, because of cost or urgency, it is the only option. SUNS™/SPICE is limited in that it only examines electrical performance, which is only one aspect of system success (and hence reliability).

The process in general is described below:

(1) A model of the relevant subsystem or system is developed. Note that it is likely that this will need to be a multi-disciplinary model with aging effects incorporated and must be capable of emulating combined environments. It also needs to be a synergistic model – that is, the different simulation regimes cannot operate independently but must exchange information dynamically (e.g., electrical parameters in the electrical simulation tool must respond dynamically to temperature changes derived by the thermal modeling tool).

(2) The models are exercised under the range of expected conditions (operational and environmental) to determine the distribution of performance values.

(3) These output distributions are compared to specified success/failure criteria to determine reliability. Alternatively, sets of distributions are evaluated to determine the frequency of unacceptable outcomes.

There are numerous benefits to the Probability Quantification process:

(1) After the initial investment to develop the requisite models, the use of modeling is much less expensive than testing. In addition it does not rely on obtaining hardware assets – assets which may or may not be controlled by DOE.

(2) Modeling allows for examination of a much wider multi-disciplinary parameter space.

(3) Direct cost/reliability trade-off studies can be performed.

The risks are as follows:

(1) The initial investment for the models could be substantial.

(2) The need for high credibility has already been discussed. Understanding the sources and magnitude of the uncertainty is an enormous technical challenge. The cost to validate the model results could be substantial.

(3) There is always a risk that the model has not captured all of the key factors. This could lead to misleading results and improper allocation of resources.

Despite the technical challenges noted above, it is strongly recommended that this process be explored thoroughly. Quantifying reliability is the ultimate goal and culmination of the work being done under the auspices of ASCI and ESP. The investment in these projects cannot be fully recouped unless a methodology for more complete use of information obtained from modeling is developed. On the other hand, there is a serious danger that modeling information can be misused; a careful approach with thorough consideration of uncertainty is essential.

W80 Firing Set Application

Scope

The W80 firing set was selected as the prototype application for the SUNS™/SPICE3 tool. The issue to be explored was that of variability of the firing set charging circuitry. In this case, the evaluation variability included that due to random part variability as well as the variability in performance due to temperature.

The W80 firing set charging circuit is a DC-to-DC converter that transforms 28 V DC power to DC voltages at the kilovolt level in order to charge the main CDU and neutron generator capacitors in the firing set. The output voltages reach regulation at 720 ms in the nominal circuit model. The voltage levels at regulation are 4580 volts on the main CDU and 1720 volts on the neutron generator capacitor. The firing set contains an oscillator, power amplifier, and voltage regulator. The circuit model used in the present work consisted of models for those three subcircuits and includes the two capacitors mentioned above as output loads.

The circuit model contains a total of 46 electrical components, as follows: 20 resistors, 7 capacitors, one inductor, 11 diodes, two transistors, an opamp, two transformers, and two 28 V DC voltage sources. The resistors, capacitors, inductor, and voltage sources are modeled with the standard SPICE models. Temperature-tracking models for the semiconductor devices were supplied by Department 1734. Models for the transformers were supplied by Departments 1733 and 8418.

Problem Setup

We used SUNS™ and SPICE3 for this application, with SUNS™ run on a Windows NT machine and SPICE3 run on CPlant. Note that the problem was executed as a set of independent runs on the CPlant processors rather than as distributed runs.

The goals of the analysis were: (1) to determine the major contributors to variability, and (2) to model the variability of the firing set, given some assumptions on random part variability and behavior over temperature. This necessitated two separate SUNS™ cases, referred to below as the uncorrelated case and the correlated case. The first assumed no correlations between the input variables. Obviously this does not reflect reality in that the temperatures of different parts in the weapon cannot be vastly dissimilar at a given time. Any information on output variability for this case would be meaningless; however, it does allow one to identify the major

contributors. The second case induced correlations between the input variables (i.e., for any given run, the temperature was effectively held constant across the circuit) in order to emulate a real case and thus result in meaningful output variability.

As described earlier in the section on Stage 1 Tool Linkage, the SUNS™ Input Editor was used to set up the variations in the 46 electrical components used in the multiple simulations described below. In the case of resistors, capacitors, and inductors, the distributions were simply taken as uniform over the range of product specifications (e.g., a uniform distribution from 0.95 nominal to 1.05 nominal for a 5% resistor, etc.). This was assumed to encompass both random variability as well as the variability due to temperature. For the correlated SUNS™ case, each of the resistors was defined to have a correlation coefficient of 0.3 with temperature. This effectively allowed some degree of correlation between resistance and temperature but did not tie them together completely. The selection of 0.3 as the correlation coefficient was arbitrary, and the sensitivity of the results due to this assumption has not been explored.

Nominal models for the semiconductor devices were used in all cases, with temperature included as an independent variable. We chose to investigate the sensitivity of the circuit performance to variations in these devices by sampling the temperature from a uniform distribution over the range specified by the STS environment. The approach for the uncorrelated case was to sample the temperature in each model from the distribution independently of the temperature in every other model. For the correlated case, the approach was to sample from a single temperature distribution and set the temperature in every device equal to that unique temperature. This mimics what would happen for the real hardware.

When the first simulations were run, it became apparent that allowing the temperature to vary widely in two Zener diode models was leading to unmanageable convergence problems (see the discussion of convergence below). In order to circumvent this difficulty, the temperatures in these two diodes were held at 27 °C. Hence, they did not participate in the investigation of the effects of variability.

The transformers were treated differently. It was not feasible to vary them continuously over the range of their specifications, so nominal, high, and low spec models were developed and used. It is assumed that the specification range encompasses both random variability and the variability due to temperature. The "Empirical Discrete" SUNS™ distribution was used to randomly select one of the three different models for each case; the choice made was to distribute them in the ratio 0.5:0.25:0.25 for nominal, high, and low, respectively.

The charging circuit depends upon voltage outputs (V1 and V2) from power supply circuitry that was not included in the analysis. A range of voltage values for V1 and V2 was provided by the firing set designer, and these ranges were input into SUNS™ with a uniform distribution. No correlations of V1 or V2 with temperature were induced in either of the SUNS™ cases.

Output from the SUNS™ Input Editor was processed using Excel and then sent to the Parse program to produce netlists for the SPICE3 simulations (see Figure 1).

Signal outputs to both the main CDU capacitor (C1) and the neutron generator capacitor (C2) which were used to measure the performance of the charging circuit were as follows: Voltage at

regulation, time to 90% of regulation voltage, dV/dt at 50 ms, voltage at 300 ms, and dV/dt at 300 ms.

Issues

In this section, we discuss a number of issues that arose in simulating the W80 firing set charging circuit.

Circuit Model Development

In developing the circuit model itself, it proved to be useful to start with a PSpice© schematic. The PSpice© package includes a schematic capture graphical user interface which allows the user to draw a schematic in a window on the computer screen. The original charging circuit model was developed in the form of just such a PSpice© schematic. The schematic is then converted to a netlist by the schematic capture program. The netlist is the input file that is read by the simulation code (PSpice© or SPICE3). The netlist refers to device models to completely define the circuit in terms of (e.g.) semiconductor and magnetics models. These models are typically contained in library files for convenience.

The SPICE3 code does not include a schematic capture interface. Input to SPICE3 is accomplished strictly by netlists. A SPICE3 model for the charging circuit was prepared by generating a netlist with the PSpice© schematic capture program and modifying it for compatibility with SPICE3.

In some cases, it was also necessary to modify the semiconductor device models themselves in the model libraries, as the syntax used in SPICE3 is slightly different from that used in PSpice©.

The greatest difficulty encountered in developing device models for the charging circuit that would run under SPICE3 was in providing good models for the nonlinear magnetic core in the MC2929 transformer in the oscillator. The PSpice© code has a nonlinear magnetics model, but SPICE3 has none. Hence, it was necessary to develop behavioral models for the transformer that were SPICE3-compatible. This is briefly described below.

Simulator Convergence

Simulation of nonlinear electrical circuits as complex as the charging circuit are frequently subject to convergence problems. This occurs when the computer code attempts to reduce the time step to an unacceptably small value (typically of the order of femtoseconds for the firing set circuit) in order to solve iteratively for the next solution step while maintaining a specified accuracy. The code then stops the simulation.

The earlier discussion of the convergence failures due to temperature variations in two Zener diodes (see the section on the W80 firing set charging circuit) provides one example of this problem.

We also frequently found nonconvergence of the charging circuit simulations to be associated with nonlinear transformer models. The presence of hysteresis in the transformer cores leads to a

complex model, which has the potential for introducing convergence problems. Furthermore, the hysteresis process itself, involving multivalued functions, is a natural source of difficulties.

The following measures were taken to obtain better convergence: (1) The tolerances on the solution accuracy were relieved (while still maintaining the tolerances at a level such that the results were sufficiently accurate for our purposes), and (2) The order of the appearance of devices in the netlists was altered (which presumably effects favorable changes in the ordering of the solution matrix). The result was that the rate of convergence failures prior to the simulation of the onset of voltage regulation were reduced to approximately 4%. Although the elimination of this 4% from the results presents a risk that effects in some region of sample space are being ignored, it was felt that this risk was statistically small enough to be acceptable.

Nonlinear Transformer Modeling

For reasons discussed previously, the development and application of a model for the MC2929 oscillator transformer has required a great deal of time and care. We currently have models that work well for simulation of the operation of the oscillator and power amplifier subcircuits, but we still have frequent convergence difficulties in the long runs necessary for a full simulation of the charging process.

The parameters which dominate the determination of the firing set oscillator frequency are those which govern the properties of the magnetic core in the MC2929 transformer. Specifically, the frequency is approximately inversely proportional to the difference between the saturation magnetization and the minimum core magnetization.

Hysteresis in the core of the MC2929 is very important in determining the oscillator frequency. The reason is that when the excitation of the core returns to zero, the magnetization does not do so because of hysteresis. Hence, the aforementioned difference between saturation magnetization and minimum core magnetization is less than it would be if hysteresis were not present, and the oscillator frequency is higher as a result.

An MC2929 model which includes hysteresis was developed, but it was found necessary to utilize a model without hysteresis in order to carry out the long simulations involved in the variability investigation without convergence failures. The elimination of hysteresis reduced the oscillator frequency, as noted above. Since hysteresis is important in the operation of the oscillator, it is necessary to justify the use of a model without hysteresis in this work. Since the effect that hysteresis has is to reduce the peak-to-peak change in core magnetization that occurs in an oscillator cycle, the effect of hysteresis can be simulated within a model without hysteresis by reducing the saturation magnetization. Even so, such a reduction was not done for the simulations carried out in the current work, as the oscillator frequency does not have a direct effect on the charging circuit output voltage, and our purpose was to compare the significance of varying parameters and not to obtain extreme accuracy.

Extraction of Simulation Results

One part of the process that required a significant amount of time was the extraction of results from the SPICE3 output files (the Analyze step in the Stage 1 Tool Linkage). As noted above, it was necessary to write new code or to significantly alter existing code for different circuit

configurations. This is simply a result of asking different questions about the behavior of each configuration. This code occasionally became rather intricate because of the need for focusing on the behavior at specific points in a circuit process and ignoring data from the signals at other points.

Input Correlations in SUNS™

Some difficulty was encountered in setting up the correlated SUNS™ case. Initially, we attempted to use SUNS™ to correlate the temperatures for all of the semiconductor devices, with a correlation coefficient set close to 1.0. SUNS™ was unable to generate a satisfactory correlation matrix, perhaps because of the sheer quantity of variables being correlated. This was solved by reducing the set of semiconductor device temperatures to a single temperature variable. This was varied using a uniform distribution. Microsoft® Excel was then used to set this temperature for all of the semiconductor values for that particular run.

Another issue appeared to be that of the number of runs. The uncorrelated SUNS™ case used 60 runs, of which 57 converged when put into SPICE3. Discussions with the SUNS™ team indicated that a minimum of 100 runs are suggested to develop statistically meaningful results. For the correlated SUNS™ case, 105 runs were selected with 101 that actually converged.

Some contradictory results were obtained when comparing results from the uncorrelated and correlated cases. For example, for the case when inputs were uncorrelated, the input C2 was shown to be positively correlated with the output "Time to 90% for C2". This is the expected dependence. However, for the correlated case, C2 was shown to be negatively correlated with "Time to 90% for C2". It both cases, the magnitude of the correlation coefficient was rather small at approximately 0.2.

The cause of this anomalous result is unknown. However, we speculate that it may be due to the limited quantity of samples combined with the inability to force zero correlation coefficients between uncorrelated inputs. The SUNS™ software offers the option to force zero correlation coefficients, but it was unable to generate a correlation matrix for this case (again, perhaps because of the large number of inputs). Thus some of the inputs that were intended to be uncorrelated had relatively high correlation coefficients (up to 0.3 in some cases) with other inputs. If the C2 input described above was inadvertently correlated with other inputs (especially those that were strong contributors), it might appear to affect the output differently because of its accidental convolution with the other input.

Results

Pareto charts of the correlation coefficients between selected outputs and the inputs are shown in Figures 9 through 12. As can be seen from Figures 9 and 10, the variability of V2 has the most influence on the variability of the time to charge up to 90% of the regulated voltage value. Figures 11 and 12 show that R110A has the most influence on the value of the regulated voltage.

Figure 9: Simple Raw Correlations for "Time to 90% for C1"

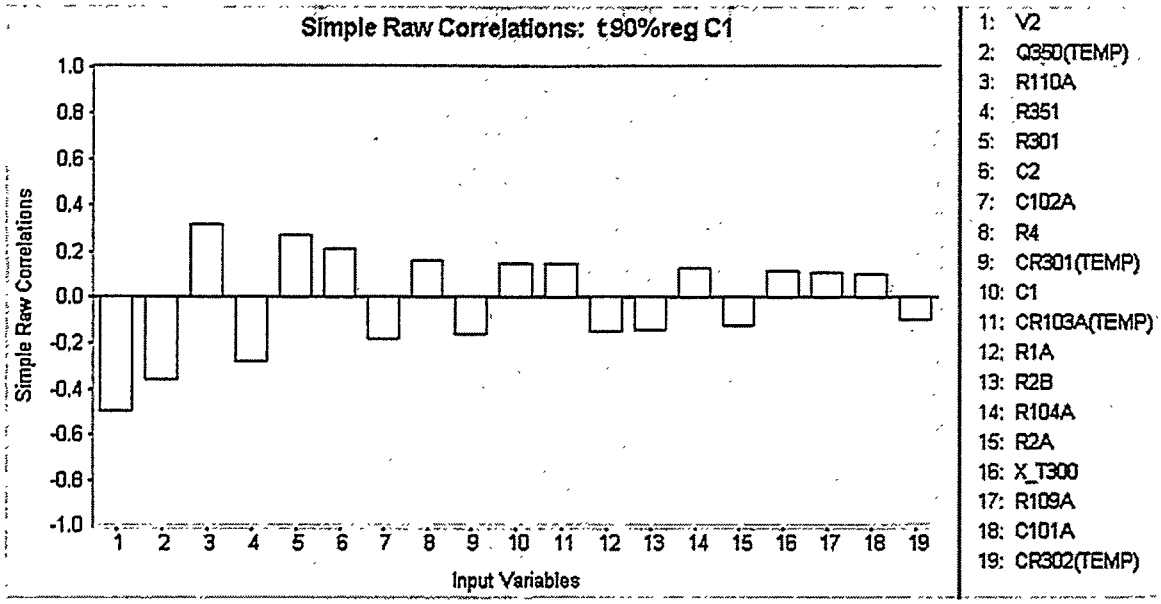


Figure 10: Simple Raw Correlations for "Time to 90% for C2"

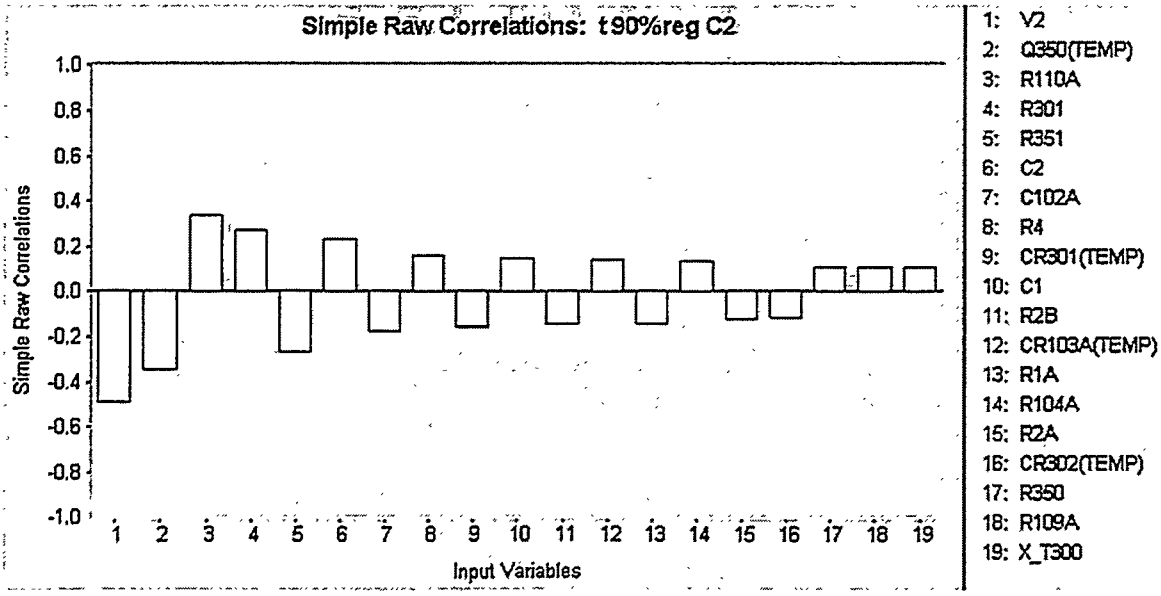


Figure 11: Simple Raw Correlations for "Vreg C1"

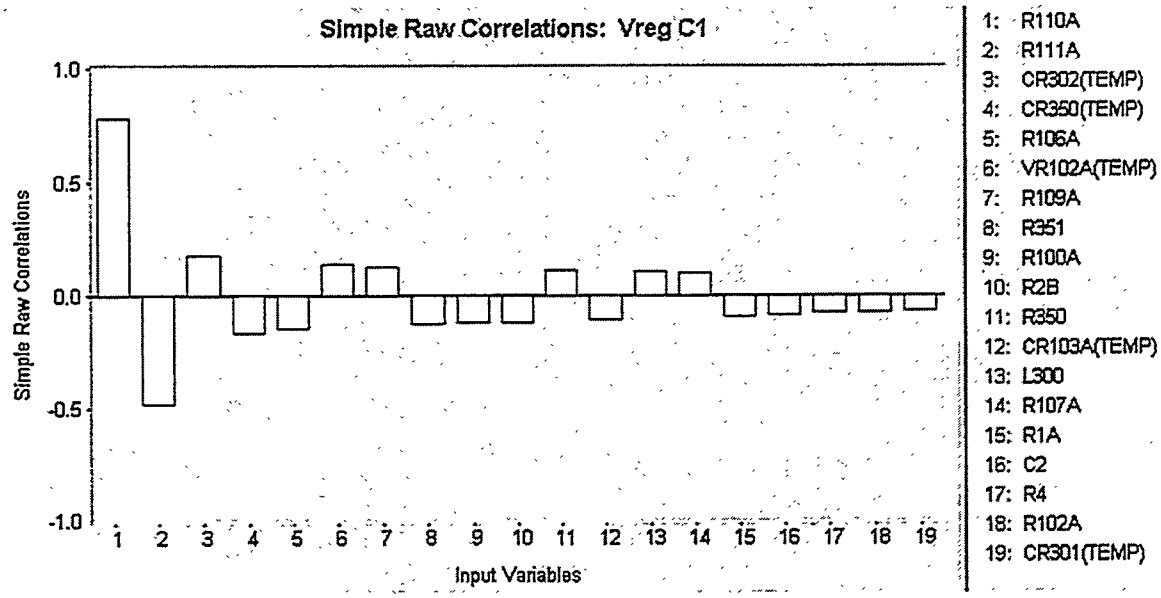
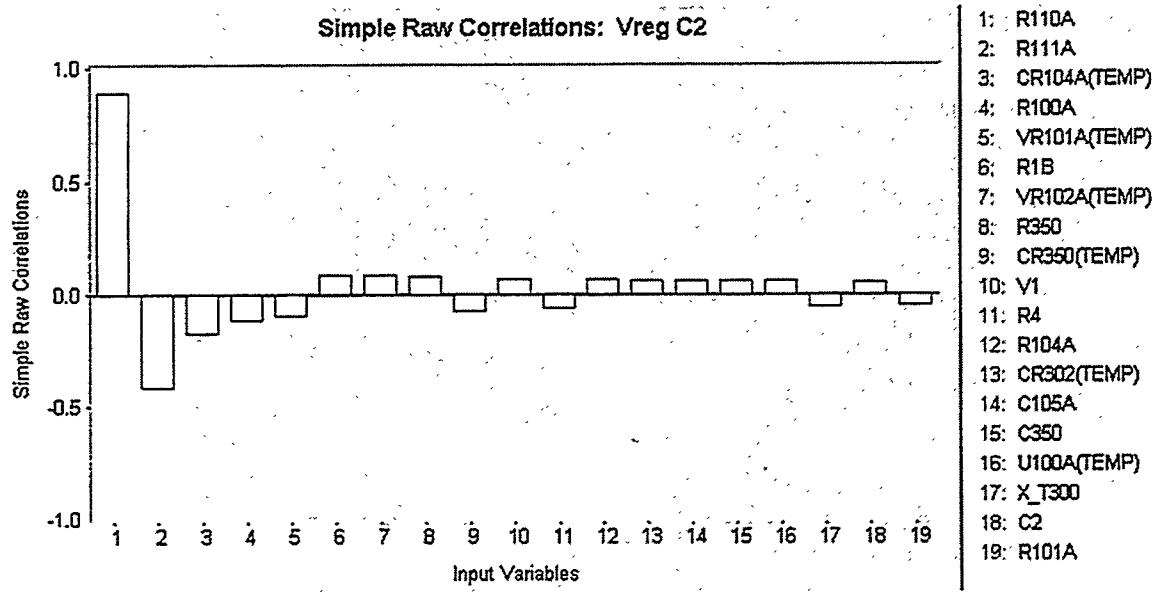


Figure 12: Simple Raw Correlations for "Vreg C2"



Figures 13 through 16 show the expected distributions of the four above outputs, given the assumptions as to the variability of the inputs.

Figure 13: Histogram for "Time to 90% for C1"

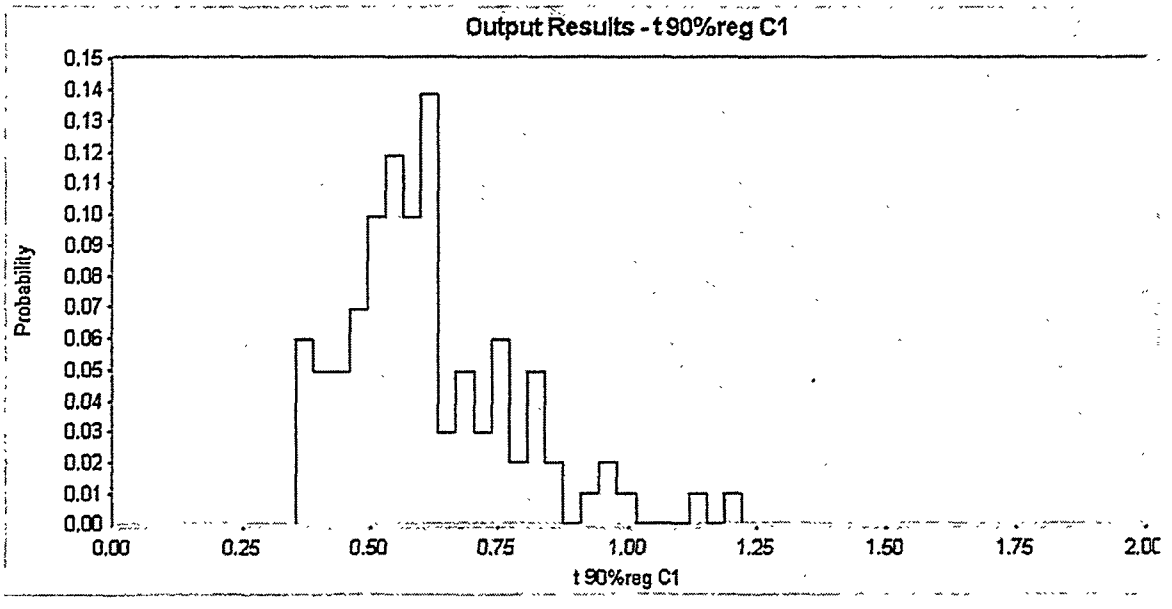


Figure 14: Histogram for "Time to 90% for C2"

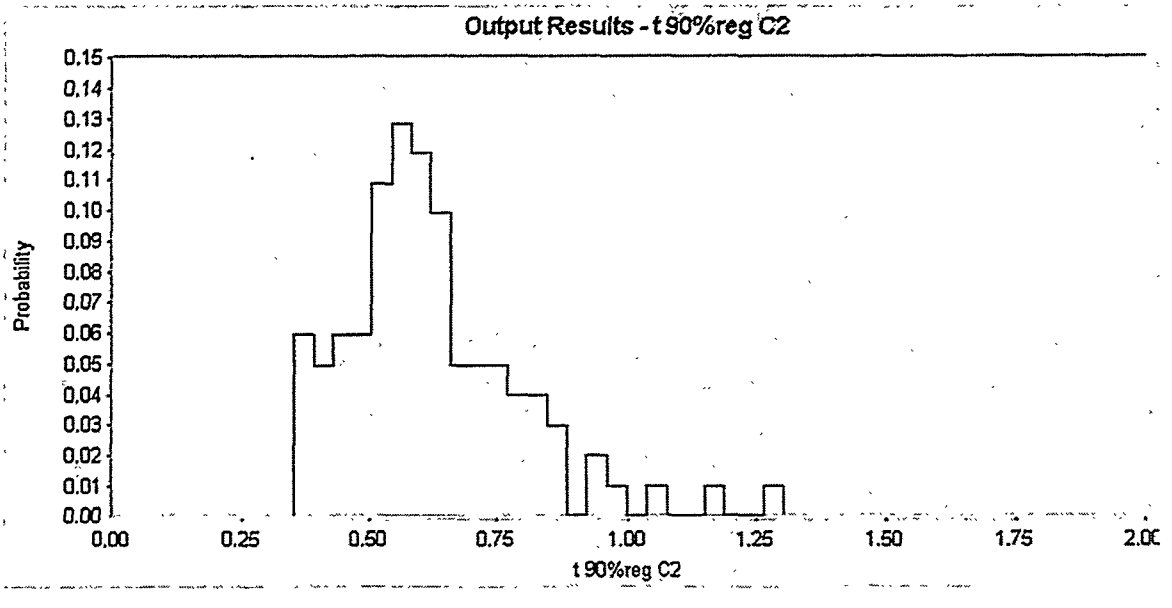


Figure 15: Histogram for "Vreg C1"

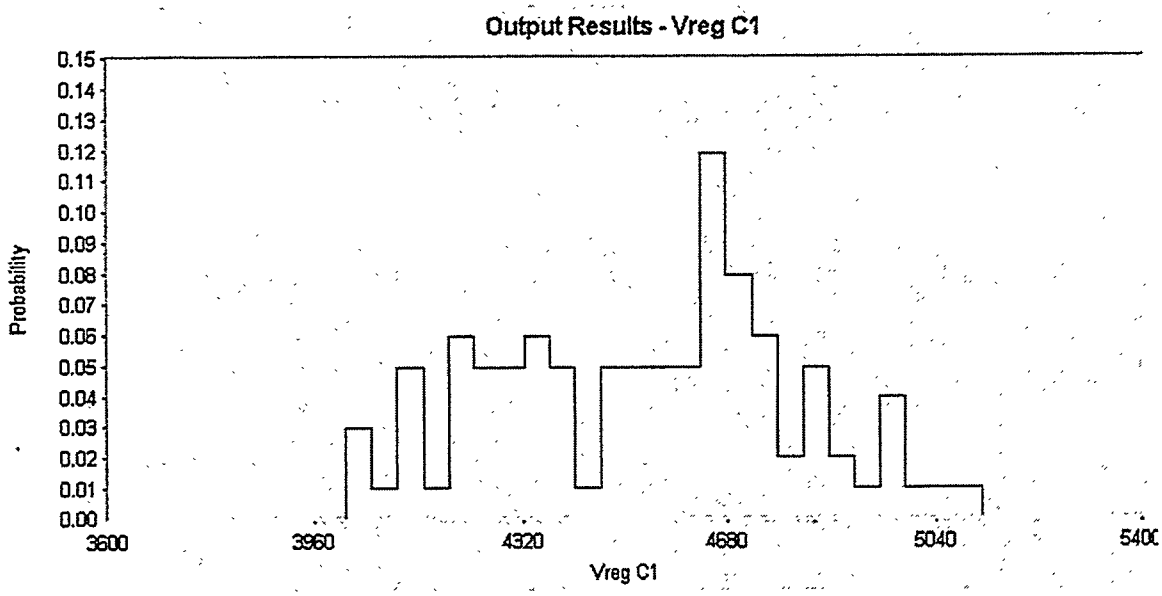


Figure 16: Histogram for "Vreg C2"

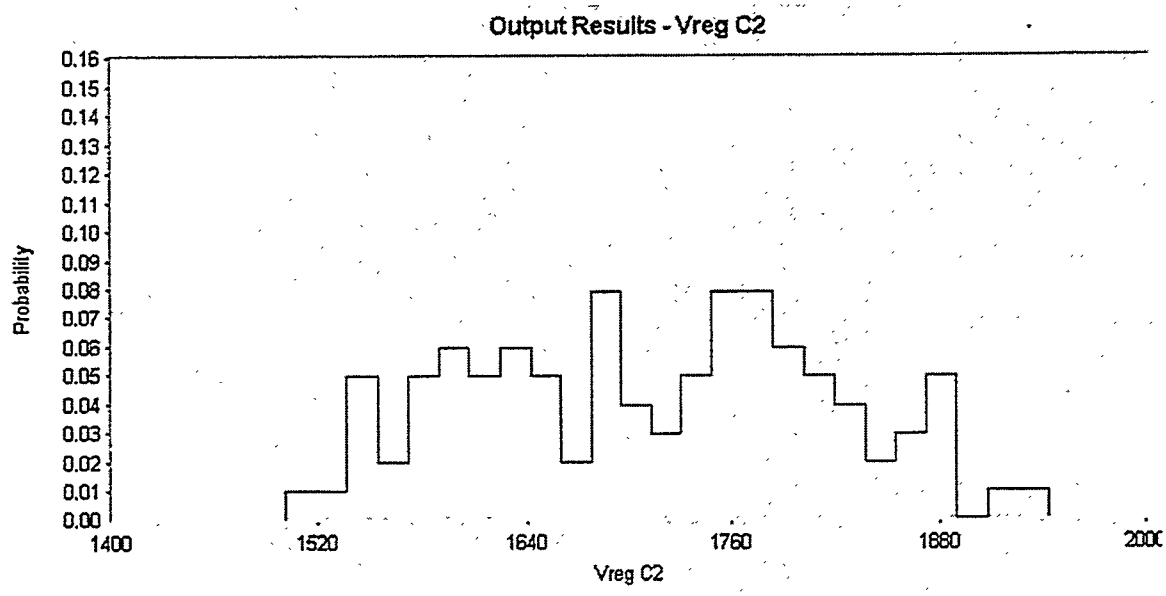


Figure 17 summarizes the means and standard deviations for each of the outputs above.

Figure 17: Output Variability

Variable	Mean	Standard Deviation
Time to 90% for C1 (sec)	0.609	0.165
Time to 90% for C2 (sec)	0.617	0.172
Vreg for C1 (Volts)	4535	259
Vreg for C2 (Volts)	1712	104

Conclusions

For each of the output variables, the variability does not result in failure to meet specification (i.e., for the expected range of inputs, the outputs are always within specification). These are encouraging results, suggesting that the firing set charging circuit design margin is adequate.

Note that if the outputs failed to meet specification for some combinations of inputs, the analysis of key contributors would provide insight into which inputs might need better control. Furthermore, the degree of control could be determined using the variability tool. Also, the expected unreliability due to the variability could be estimated by examining the output histogram vis-à-vis the output specification. The fidelity of this estimate obviously hinges upon the fidelity of the input information, the completeness of the analysis (i.e., have all important factors been included?), and the relationship of the output specification to the actual pass/fail criteria. Despite all of these caveats, the variability analysis will certainly help to identify areas where design margin is questionable or inadequate. In those cases, either further analysis or focused testing can be performed as validation.

Optimization

One of the initial goals of this project was to do a similar tool linkage and methodology development for an optimization code, GO™ (Genetic Optimization). Further work on GO™ was not pursued because of the opportunity to pursue the Stage 2 variability tool linkage.

Although it was not used in the present work, some optimization software has recently been applied to improve the agreement between simulation results and laboratory data in the case of the oscillator circuit in the W80 firing set charging circuit. The software employs a technique referred to as the asynchronous parallel direct search (APDS) optimization method and was implemented on the CPlant computing system.

There is a predecessor to APDS, the standard parallel direct search (PDS) optimization method which can be quite useful for engineering optimization problems characterized by expensive objective function evaluations. However, PDS does not perform well on cluster computational platforms such as CPlant because it is hindered by synchronization penalties and crashes in the

event of a node failure. The APDS technique is an asynchronous fault-tolerant version of PDS that overcomes these limitations[3].

A previous attempt to optimize the oscillator model without using the APDS approach had involved the adjustment of a limited number of parameters in a rather crude procedure. It resulted in a significant improvement in the agreement between simulation results and laboratory data, but it required detailed prior knowledge of the effects of certain parameter variations. When the APDS method was employed, it utilized more circuit parameters and was done as a blind test. That is, the approach was not influenced by prior knowledge of effects. The APDS method performed very well; it resulted in a significant improvement over the crude procedure in the comparison of simulation results with laboratory data.

We feel that such optimization software could play a strong role in future modeling efforts. It is a much more efficient way to close the gap between modeling results and laboratory measurements, an often time-consuming process when high fidelity models are required for analysis.

Follow-On Work

There are three main thrusts for follow-on activities. The first is continued development of the variability tool within the framework of the HPEMS user interface, including the addition of an "Analysis Advisor". This provides a valuable tool for the designer to perform variability analysis during the course of the design, when there is maximum flexibility to make changes to the circuit design. The second thrust is incorporating the optimization capability in the HPEMS user interface. This will allow the designer to identify means by which design margin can be improved. As noted in the previous section, this could also provide an important capability to the modeling community. The final area for exploration is analyzing time-dependent phenomena such as aging or low-dose radiation. There is a need for tools that can examine the impact of a degrading parameter on the overall system, particularly when the effect is a pervasive one throughout the system.

---

[3] See the following references for more information:

J. E. Dennis and V. Torczon, "Direct Search Methods on Parallel Machines", SIAM J. Optimization, 1(1991):448-474.

P. D. Hough, T. G. Kolda, V. Torczon, "Asynchronous Parallel Direct Search Algorithms for Nonlinear Optimization", in preparation (1991).

Appendix A: SUNS™ User's Reference Manual

This page intentionally left blank

Center for System Reliability

SUNS™
User's Reference Manual

Version 1.0

Sandia National Laboratories
Albuquerque, New Mexico

October, 1999

"Exceptional Service in the National Interest"

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Contents

Figures

Tables

This page intentionally left blank

1 Getting Started

1.1 Introduction to SUNS for Windows

SUNS is an acronym for Sensitivity and Uncertainty Analysis Shell. SUNS for Windows is a 32-bit application that runs under Windows 95/98 and Windows NT. It is designed to aid in statistical analyses for a broad range of applications. The class of problems for which SUNS is suitable is generally defined by two requirements:

1. A computer code is developed or acquired that models some process (referred to here as the user application) for which input is uncertain and the user is interested in statistical analysis of the output of that code.
2. The statistical analysis of interest can be accomplished using Monte Carlo analysis.

The implementation then requires that you identify which input to your application should be manipulated for statistical analysis. With this information, changes required to loosely couple SUNS with the application can be completed. Finally, SUNS is used to generate the required samples and your application analyses the sample.

This document provides a brief introduction to the statistical sampling capabilities included in SUNS for Windows. Previous versions of SUNS were DOS applications. For the sake of simplicity, this documentation refers to SUNS for Windows Version 1 as "SUNS."

Note: The version of SUNS documented here is at a beta development stage, meaning that it is currently being tested.

1.2 System Requirements

To install SUNS you will need an IBM-compatible PC with a 486 or Pentium processor. A Pentium-based system is recommended. SUNS is a 32-bit Windows application and will run under Windows 95/98 and Windows NT. The installation requires approximately 5 MB of free hard drive space. The amount of RAM required to run SUNS is dependent on the memory requirements of the application you are going to use with SUNS.

To use SUNS you will need to be familiar with basic Windows functions such as menus, dialogs, files, and mouse operation. If you are not accustomed to using Windows applications, please refer to your Microsoft Windows User's Guide. To link your application with SUNS you will also need a detailed knowledge of your application source code. A detailed description of the file format used to transfer data to and from your application is provided in Chapter 4, *Linking SUNS with Your Application.*

1.3 Installing SUNS

SUNS must be installed on your computer using the provided Setup program. The program files are compressed and cannot be used by copying them directly to your hard drive.

To install SUNS from floppy disks:

1. Start Windows.
2. Insert SUNS Disk 1 into your floppy disk drive (A: or B:).
3. Choose "Run..." from the Start menu.
4. Type A:\Setup (or B:\Setup) in the Command Line text box.
5. Follow the on-screen instructions.

To install SUNS from a CD-ROM:

1. Start Windows.
2. Insert the GO CD-ROM into your CD-ROM drive (typically D:).
3. Choose "Run..." from the Start menu.
4. Type D:\Setup (where D: is the letter for your CD-ROM drive) in the Command Line text box.
5. Follow the on-screen instructions.

When you run the Setup program, you'll see a form like the one shown in Figure 1.1. If you want to install SUNS in the default directory, click the Next button to proceed. If you want to install SUNS in some other location, click the Browse button to select an alternate location. Then proceed with the installation by clicking the Next button.

As part of the installation process, Setup will create a Program item "SUNS" on your Start|Programs menu. In addition to the program files, Setup will also install some example files and an online version of this User's Manual.

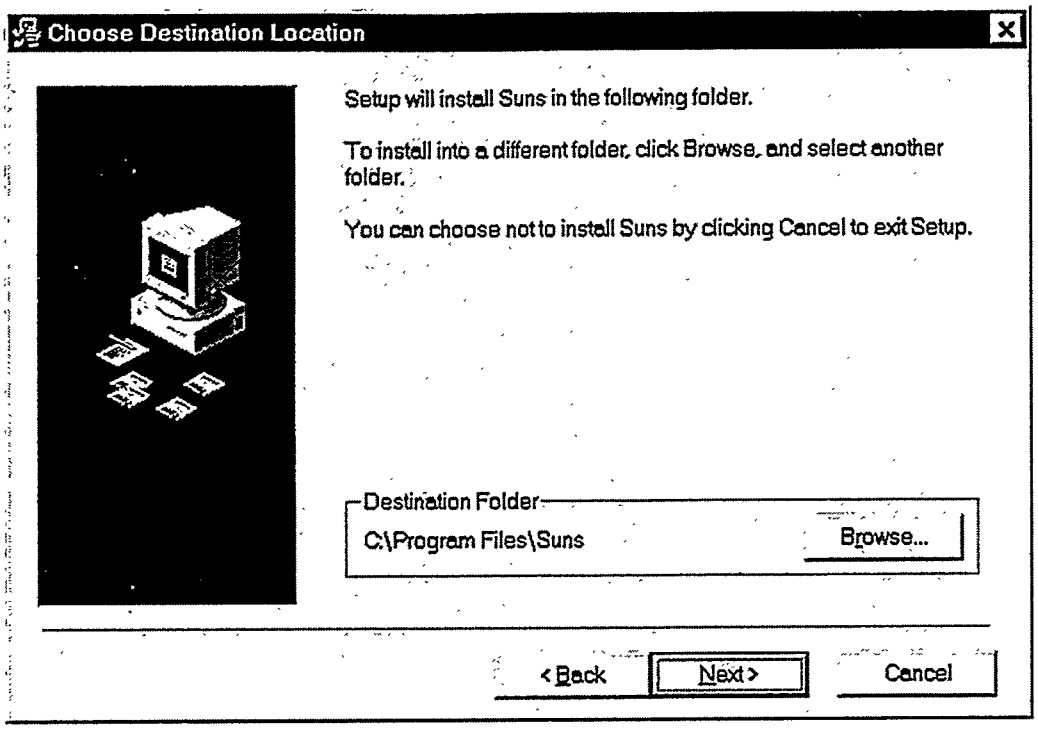


Figure 1.1 SUNS Installation Form

1.4 User's Reference Manual

The SUNS documentation set consists of this User's Reference Manual (paper and on-line versions), a context-sensitive help system, and the Readme.txt file copied to your SUNS program directory during installation. The SUNS User's Reference Manual provides information on the distribution and correlation options available in SUNS as well as information on how to use the SUNS software. Here's what you'll find in each chapter of this manual:

Chapter 1, *Getting Started*, provides system requirements and information on how to install SUNS on your computer.

Chapter 2, *SUNS Design Overview*, briefly describes how SUNS is designed and how it interacts with your application.

Chapter 3, *SUNS Statistical Sampling*, provides a detailed description of the distributions available in SUNS and defines the file format used to transfer sample values to your application.

Chapter 4, *Linking SUNS With Your Application*, describes how to use the SUNS user interface to define variables, assign distributions and establish correlations.

Chapter 5, *SUNS Results Viewer*, shows you how to display the results of an analysis performed using the SUNS sample vectors.

Chapter 6, *SUNS Example*, provides a simple example that uses the SUNS software to address sensitivity and uncertainty questions.

Chapter 7, *References*, provides a list of references for Latin Hypercube Sampling.

1.5 In Closing...

If you run into difficulties and cannot find a solution in the documentation, please consult the Readme.txt file installed in your SUNS program directory. It contains information that has become available since the manual was written and also contains contact information if you need help.

Like most software, SUNS has benefited from the comments and suggestions of others. If you have comments or suggestions for improvements, please share them with the authors. In addition, we would like to hear about your applications of SUNS.

Lastly, please keep in mind that SUNS is copyrighted software. You may make backup copies of the software for your own use but you may not distribute the software or documentation to others without prior consent.

Microsoft, Windows, Windows 95, Windows 98, and Windows NT are trademarks of Microsoft Corporation. Pentium is a trademark of Intel Corporation. SUNS is a trademark of Sandia Corporation.

2 SUNS Design Overview

The previous (DOS) version of SUNS was designed for tight coupling with your application. That is, your computer code had to be modified to fit into the SUNS structure and compiled as an integral part of SUNS. SUNS then provided an input editor, a statistics driver (which was integrated and compiled with your application) and a results viewer. Thus SUNS provided the entire interface including an executive program to control execution of the various SUNS modules. In the current (Windows) design, Windows provides executive functions and there is only a loose coupling of SUNS with your application. By "loose coupling" we mean that your application will not be coupled directly with SUNS by compilation. Instead your code will remain a separate application and communication with SUNS will be performed using data files.

When using SUNS as a statistical driver, analysis proceeds in three phases:

1. The SUNS statistical driver creates an input sample.
2. Your application analyzes all the input vectors in the sample.
3. Results are viewed using the SUNS results viewer.

The structure is shown in Figure 2.1.

Figure 2.1 SUNS Structure for Statistical Analysis

As shown in Figure 2.1, statistical (sensitivity and uncertainty) analysis requires a single pass. For this reason, SUNS does not need any connection to, or control over your application. A file of sampled input vectors can be created and then analyzed at any time the user chooses. Similarly, the SUNS results viewer can open any existing file that contains sampled input and output values.

This page intentionally left blank

3 SUNS Statistical Sampling

The SUNS Statistical Driver provides an interface allowing you to specify input variable distributions and, if desired, correlations among input variables. Once input is complete, SUNS creates the sample and saves it to an ASCII text file.

3.1 Sampling Methodology

The statistical sampling approach uses Latin Hypercube Sampling (LHS) [1, 2] by default although random (Monte Carlo) sampling is also available. All the algorithms required to create the sample and induce correlations are encapsulated into an ActiveX DLL, which is a type of reusable software component.

3.1.1 Distributions

Each distribution is either a theoretical type, which must have associated parameters, or an empirical type, which must have associated data pairs. In addition, a "Fixed" distribution may be specified. In this case, a specified value is assigned to a variable for every sample.

3.1.2 Correlations

Variables may be correlated with each other [3, 4]. A positive correlation between two variables A and B means that when one variable A is toward the upper end of its allowable range, variable B tends to be towards the upper end of is allowable range. Conversely, a negative correlation between two variables A and B means that when variable A is toward the upper end of its allowable range, variable B tends to be towards the lower end of is allowable range. The correlation coefficient must be in the range (-1, 1). A value of +1 indicates perfect positive correlation while a value of –1 indicates a perfect negative correlation. A correlation coefficient of 0 indicates that the variables are independent.

The user can select an option to induce zero correlations. If there are non-fixed variables for which no correlations have been specified, selecting this option causes SUNS to attempt to induce zero correlations between all variable pairs that are not correlated. If there are correlations specified, the user can partition the correlated variables into groups. A group is characterized by a set of variables (none of which appear in any other group) for which correlations have been specified. Note that if you select the option to induce zero correlations, groupings are ignored.

3.1.3 Properties

In addition to distributions for each variable and correlations amongst variables, several options can be set. These include:

- Use of LHS or random Monte Carlo sampling.

- A seed value for the random number generator.
- The number of trials to be performed.
- The option to induce zero correlations between variable pairs for which no correlations have been specified.

3.1.4 SUNS Distribution Types

The primary purpose of the SUNS user interface is to allow the user to specify variable distributions to be sampled. A large number of distributions are available for selection from a drop-down list. The parametric distributions are listed in Table 3.1.

Distribution	Parameter 1	Parameter 2	Parameter 3	Parameter 4
Fixed	Value			
Normal	Mean	StdDev > 0		
Truncated Normal	Mean	StdDev > 0	LowBd ≥ 0	LowBd < **UpBd** ≤1
BoundedNormal(3)	Mean	StdDev > 0	LowBd	LowBd < **UpBd**
End Point Normal	0.1 P'Tile	99.9 P'Tile		
ErrorFactor Lognormal	Median > 0	ErrorFactor > 1		
Lognormal (1)	Mean	StdDev > 0		
TruncatedErrorFactor Lognormal	Median > 0	ErrorFactor > 1	LowBd ≥ 0	LowBd < **UpBd** ≤1
Truncated Lognormal (1)	Mean	StdDev > 0	LowBd ≥ 0	LowBd < **UpBd** ≤1
BoundedErrorFactor Lognormal(2)	Median > 0	ErrorFactor > 1	LowBd	LowBd < **UpBd**
BoundedLognormal (1, 3)	Mean	StdDev > 0	LowBd	LowBd < **UpBd**
End Point Lognormal (4)	0.1 P'Tile > 0	99.9 P'Tile > 0		
Uniform	Min	**Max** > Min		
LogUniform	Min > 0	**Max** > Min		
Exponential	Rate > 0			
Maximum Entropy	Min ≥ 0	**Mean** > Min	**Max** > Mean	
Weibull(5)	Shape > 0	Scale > 0		
Pareto	Shape > 2	Scale > 0		
Gamma	Shape > 0	Scale > 0		
Beta	Min	**Max** > Min	$Shape_p \geq 0.001$	$Shape_q \geq 0.001$
Inverse Gaussian(6)	Mean > 0	Spread > 0		
Triangular (7)	Min	**BestEst** ≥ Min	**Max** ≥ BestEst	
Poisson	$10^6 \geq$ Freq > 0			
Binomial (8)	0 < p(fail) < 1	#Tests > 1		
Negative Binomial (9)	0 < p(succ) < 1	#Tests > 1		
Geometric	0 < p(succ) < 1			
Hypergeometric	#Tests > 1	0 ≤ **#Fail** < #Tests	0<**#Samp**<#Tests	

(1) Mean, StdDev in underlying normal units
(2) Bounds input in lognormal units. In normal units for derived underlying Mean and StdDev, Log(Min) < Mean + 5 StdDev and Log(Max) > Mean - 5 StdDev
(3) Bounds in normal units. Also require Min < Mean + 5 StdDev and Max > Mean - 5 StdDev
(4) Percentiles in lognormal units, so must be positive.
(5) Require $Shape^{-1} < 31.91505 + 0.3597197$ Log(Scale)
(6) Require ratio of spread to mean from 10^{-7} to 80.
(7) Require Min < Max, but BestEst could equal one or the other.
(8) Binomial requires failure probability.
(9) Negative Binomial and Geometric require success probability.

Table 3.1 Parametric Distributions in SUNS

In addition to the parametric distributions listed in Table 3.1, there are 5 user-definable empirical distributions. These are:

Continuous Linear: The user provides N ordered pairs where the first number in the pair is the value of the variable at a particular point and the second is the cumulative probability associated with that value.

Continuous Logarithmic: Same input as Continuous Linear.

Continuous Frequency: The user provides N ordered pairs where the first number is the value of the variable at a particular point and the second number is the relative frequency at that point.

Discrete Cumulative: The user supplies N ordered pairs where the first entry of a pair is a value and the second entry is its cumulative probability. Values must increase monotonically from pair to pair.

Discrete Histogram: The user supplies N ordered pairs where the first entry of a pair is a value and the second entry is its relative frequency. Values must increase monotonically from pair to pair and all frequencies must be positive, but they need not add to one.

All of these distributions (parametric and empirical) are available from a drop-down list. Parameter checking is done at the time of data entry. Empirical distribution data is entered in a grid and is also checked at input time.

To specify correlations among input variables, those variables to be correlated are selected from a list by the user. All possible variable pairs are then presented in a grid for the user to provide correlation values. When all input is complete, the user will be able to create the sample (with or without correlations) and save the sampled values to a file of their choosing.

3.2 SUNS Data Files

In addition to creating a file of sampled values (vec file), SUNS can save a file containing all of the user-supplied variable distributions and correlation data (sun file).

The format of the vec sample file is illustrated in Table 3.2.

NumTrials	NumInputVars		
Variable Name 1	Variable Name 2	Variable Name 3	Variable Name 4
$Sample_{1,1}$	$Sample_{2,1}$	$Sample_{3,1}$	$Sample_{4,1}$
$Sample_{1,2}$	$Sample_{2,2}$	$Sample_{3,2}$	$Sample_{4,2}$

Table 3.2 SUNS Sample File Format

- NumTrials is an integer indicating the number of trials (the sample size).
- NumInputVars is an integer giving the number of input variables sampled.
- Variable Name j is a string giving the user-provided name of the j^{th} input variable.
- $Sample_{i,j}$ is the j^{th} sampled value of the i^{th} variable.

The file is comma-delimited. An example of a file having 3 input variables and a sample size (NumTrials) of 5 is shown in Table 3.3.

```
5, 3
"Resistance R1", "Resistance R2", "Resistance R3"
510, 55.2, 77.3
498.2, 61,4, 70.3
503.7, 58.3, 74.6
495.9, 60.6, 73.5
508.2, 59.7, 74.4
```

Table 3.3 Example SUNS File

3.3 The User Application

While we cannot provide specifications for the user application, it is necessary for the application to be able to read the vec file produced by SUNS. For the user application, the vec file provides NumTrials input vectors to be processed. For example, in Table 3.3, the user application would perform an analysis setting

Resistance R1 = 510

Resistance R2 = 55.2

Resistance R3 = 77.3

Thus the first input vector is 510, 55.2, 77.3. When processing is complete for each input vector, your application must add output results to the end of the input vector. The general form of the user application output file is shown in Table 3.4.

NumTrials	NumInputVars	NumOutputVars		
Variable Name 1	Variable Name 2	Variable Name 3	Output Name 1	Output Name 2
Sample $_{1,1}$	Sample $_{2,1}$	Sample $_{3,1}$	Output $_{1,1}$	Output $_{2,1}$
Sample $_{1,2}$	Sample $_{2,2}$	Sample $_{3,2}$	Output $_{1,2}$	Output $_{2,2}$

Table 3.4 User Application Output File

- NumTrials is an integer indicating the number of trials (the sample size).
- NumInputVars is an integer giving the number of input variables sampled.
- NumOutputVars is an integer giving the number of outputs calculated per simulation.
- Variable Name j is a string giving the user-provided name of the j^{th} input variable.
- Output Name k is a string giving the name of the k^{th} output result.
- $Sample_{i,j}$ is the j^{th} sampled value of the i^{th} variable.
- Output $_{i,j}$ is the value of the i_{th} output resulting from the j_{th} input vector.

For example, suppose there are two outputs of interest - call them Voltage V1 and Voltage V2. Then the output file from the user application might look like that shown in Table 3.5.

5, 3, 2 "Resistance R1", "Resistance R2", "Resistance R3", "Voltage V1", "Voltage V2" 510.0, 55.2, 77.3, 17.9, 5.2 498.2, 61,4, 70.3, 18.1, 5.1 503.7, 58.3, 74.6, 18.0, 5.0 495.9, 60.6, 73.5, 17.8, 4.9 508.2, 59.7, 74.4, 17.9, 5.1

Table 3.5 Example SUNS Sample Output File

3.4 SUNS Results Viewer

The SUNS Results Viewer reads the output file from the user application and displays various results in graphical and tabular form. The SUNS Results Viewer cannot modify the user application's output file.

This page intentionally left blank

4 Linking SUNS With Your Application

This section describes how to use the SUNS interface to enter variables and their associated distributions and correlations. Also included in this section are 1) the steps required to perform sampling once the data entry process is completed, and 2) a brief description of other options that are available within the software. Figure 4.1 shows the main SUNS screen when the program is first started.

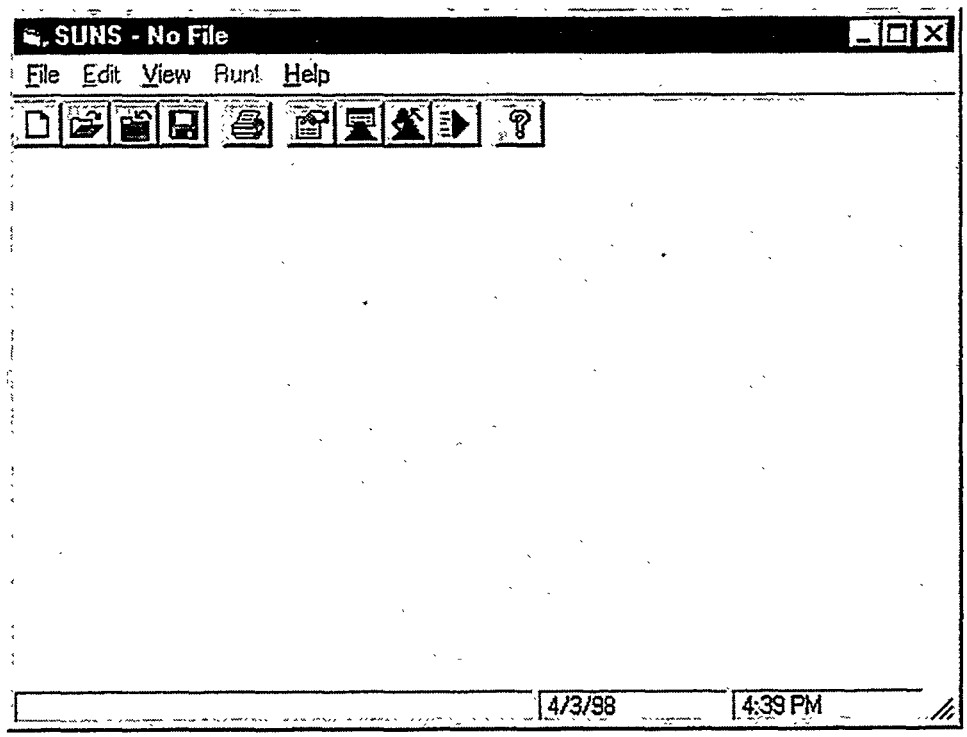


Figure 4.1 SUNS Main Program Screen

4.1 Creating a New File

The definition of distributions and correlations associated with each variable, together with other sampling options, are stored in a file with the sun extension. To create a new file, pull down the File menu and select the New command. You will be prompted to enter a new file name. SUNS then creates the file and opens it so that you can proceed to enter information.

4.2 Saving Your Work

When you have entered information about the distributions and correlations associated with the variables in you application, you will want to save the data. To do so, pull down the File menu and select the Save command. This command will write your current data to the file you are currently editing. Note that SUNS saves this data in a file with the sun extension.

If you want to save your work to a new file, use the Save As command on the File menu. This command will prompt you for a new file name and will then save the current information to the file that you supplied. From that point forward, any changes that you make will be to the *new* file.

4.3 Opening an Existing File

To open an existing SUNS file, pull down the File menu and select the Open command. You will be prompted for the sun file that you want to edit.

4.4 Editing Variable Data

To enter or edit the variables for which you wish to generate sampled values, pull down the Edit menu and selected the Variables command. You'll see a form like the one shown in Figure 4.2. On the left side of the form is an area where you can maintain a list of the variables for which you want sampled values. On the right side of the form is an area where you can edit, for a selected variable, the required distribution type, distribution parameters, and notes.

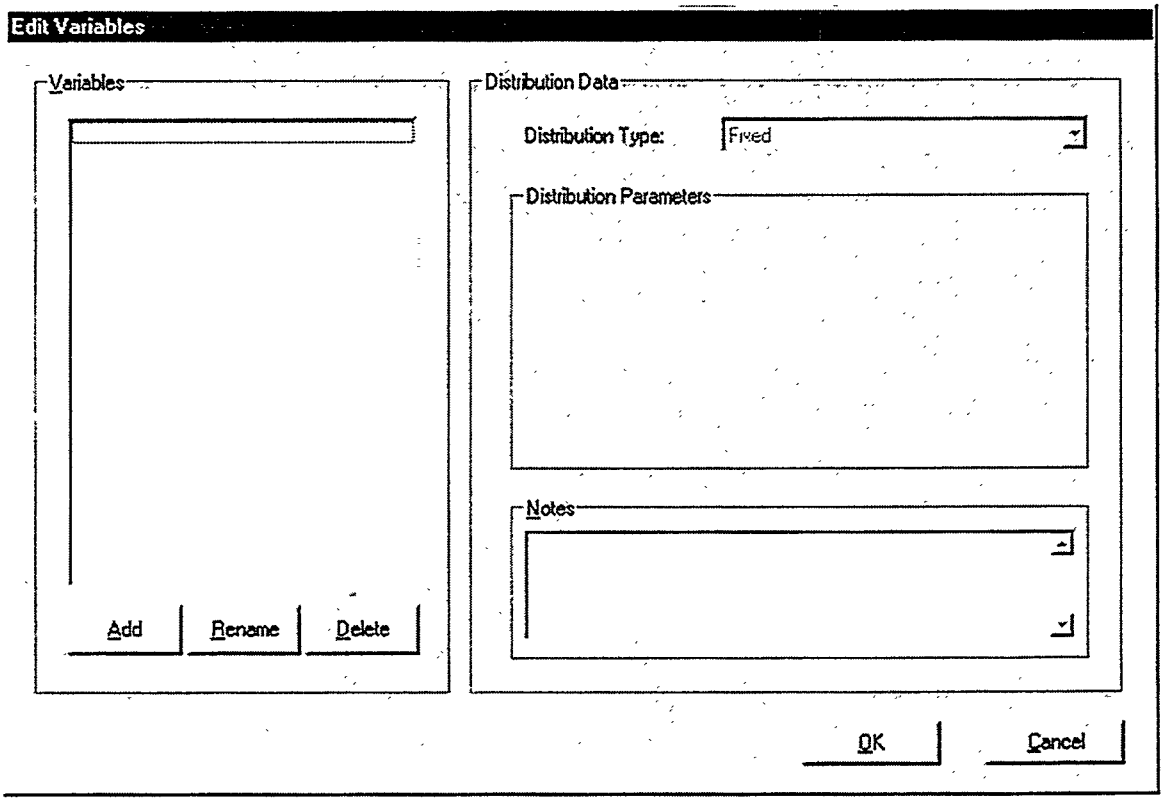


Figure 4.2 Editing Variables

To add a variable to the list, click the Add button. You'll be asked for a variable name. Variable names can be any non-blank text, but they must be unique. The variable name that you enter will be added to the list, which is maintained in alphabetical order. Until you make a different selection, variables that you add to the list will be assigned "Fixed" distributions with a value of zero.

To change the name of a variable already in the list, first select the variable to be renamed and then click the "Rename" button. You'll see the current name and a space to type in the new name. Just as with adding variables, the new name must be unique amongst the list of variables.

To delete a variable from the list, first select the variable to be deleted and then click the Delete button. You'll be asked to confirm the command. If you do confirm the command the currently selected variable will be deleted from the list. In addition, any correlations involving the deleted variable will be removed from the correlation matrix.

4.4.1 Theoretical Distributions

To enter a non-fixed distribution for a variable, first select the variable and then pull down the list of available distributions from the Distribution Types list on the form shown in Figure 4.2. Choose the required distribution and you will see spaces to enter the distribution parameters. For example, Figure 4.3 shows what the screen will look like when you select a Normal distribution for the variable named "Resistance R1." Two spaces are shown so that you can enter the mean and standard deviation for the distribution. If instead of a normal distribution, you choose a bounded normal distribution, four spaces would be visible for mean, standard deviation, upper bound, and lower bound.

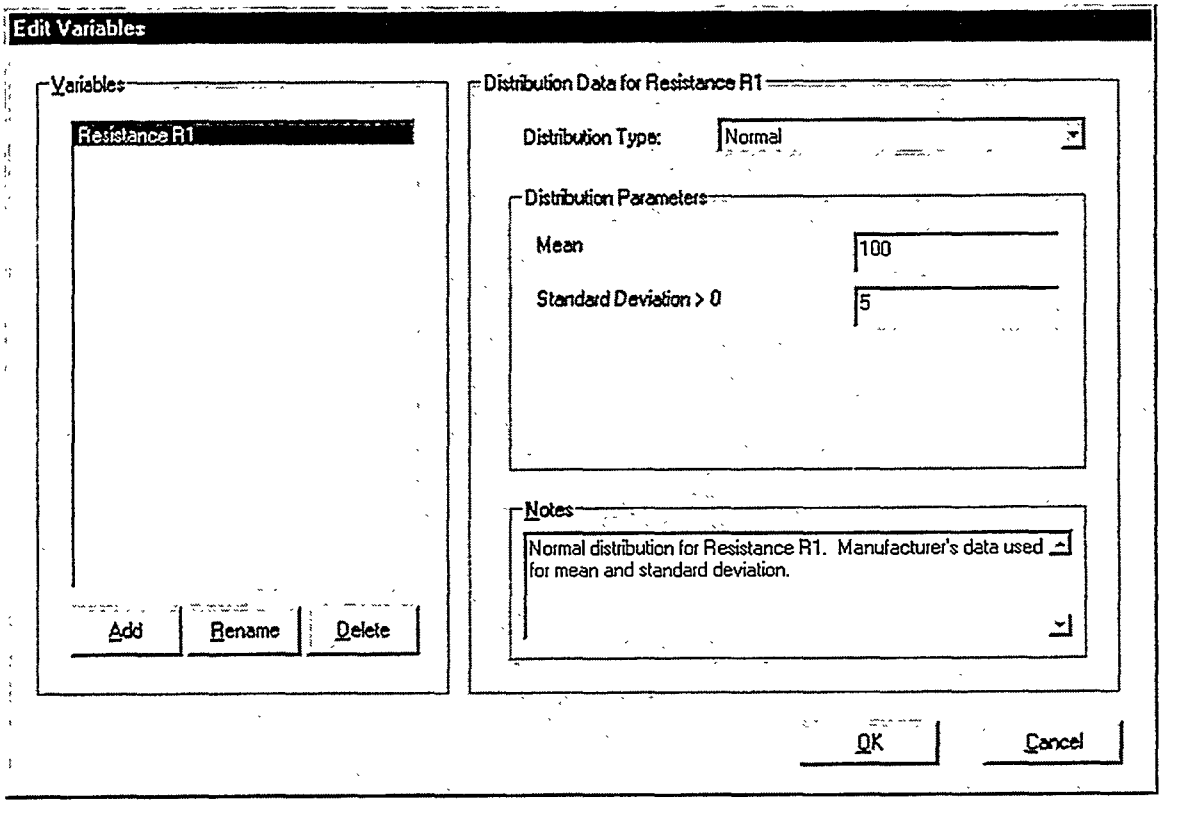


Figure 4.3 Supplying Parameters for a Normal Distribution

If you select a new theoretical distribution type, as many parameter values as possible from the old distribution are retained. This may be useful in some cases but in others cases you'll need to re-enter all the parameter values.

4.4.2 Empirical Distributions

Empirical distributions are defined by pairs of values and are entered into a two-column grid. To see the data entry grid, select a variable and then choose one of the supported empirical distributions. When you do so, a grid like the one shown in Figure 4.4 replaces the 1 to 4 distribution parameter boxes. Column headers are provided to indicate which values are to be entered in which columns. If you need more rows than are displayed, use the scroll bar on the right side of the grid to see more cells.

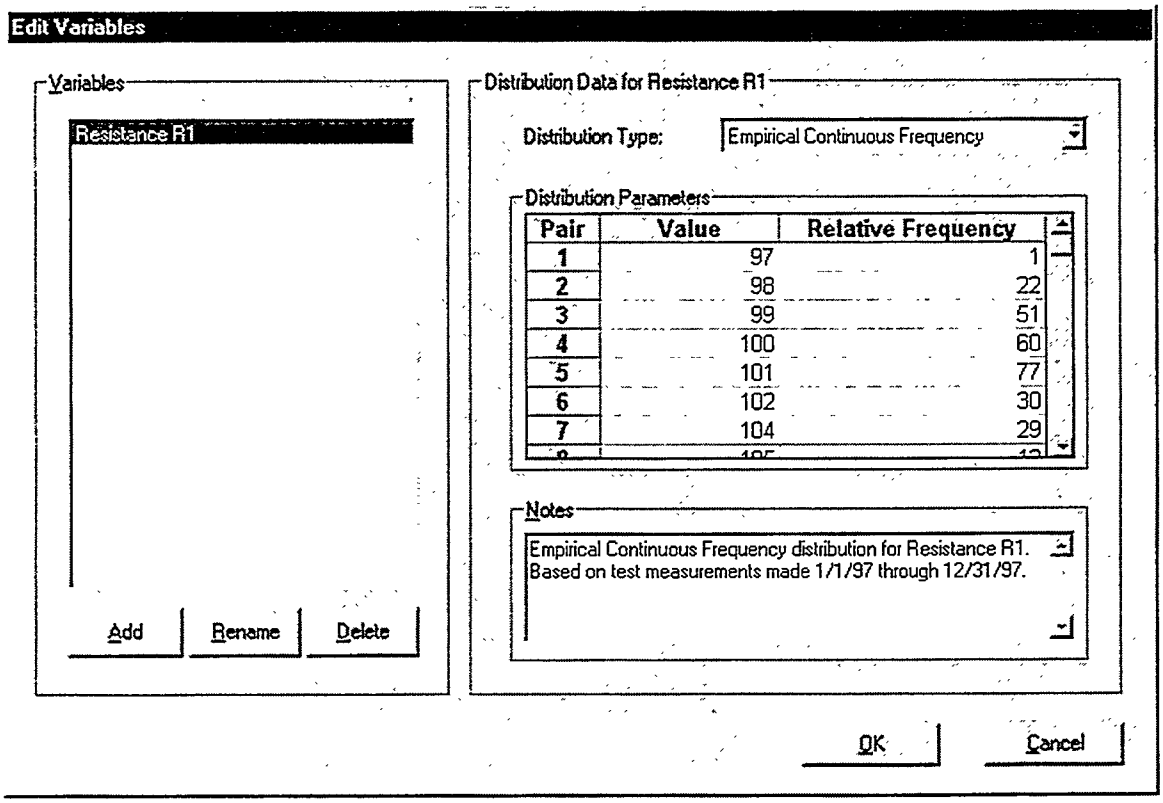


Figure 4.4 Entering Data Pairs for an Empirical Distribution

4.4.3 Notes

Figures 4.3 and 4.4 show an area on the right hand side on the Variables screen for entering notes. Here you can enter an unlimited amount of text for each variable. You may want to use this to clarify the use of a particular variable or to record the reasoning used to select a distribution type or distribution parameters.

4.5 Editing Correlations

Defining correlations between variables requires several steps. First you define one or more groups. Each correlation group is composed of two or more variables that you want to correlate. So, for each correlation group you must select the variables you want to include in the group. Lastly, for each group of correlated variables you provide correlation coefficients.

4.5.1 Correlation Groups

To enter or edit correlations between the variables, pull down the Edit menu and select the Correlations command. You'll see a form like the one shown in Figure 4.5.

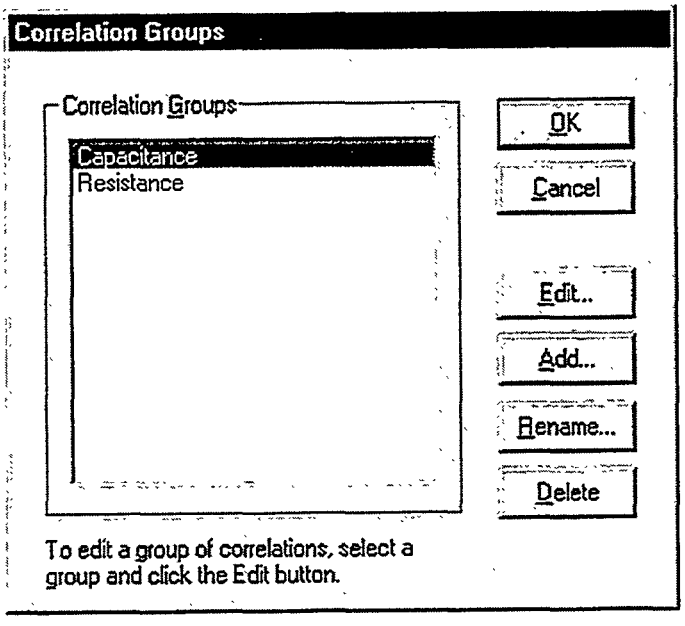


Figure 4.5 Correlation Groups

To add a correlation group to the list, click the Add button. You'll be asked for a group name. Correlation group names can be any non-blank text, but they must be unique. The group name that you enter will be added to the list, which is maintained in alphabetical order.

To change the name of a group already in the list, first select the group to be renamed and then click the "Rename" button. You'll see the current name and a space to type in the new name. Just as with adding groups, the new name must be unique amongst the list of correlation groups.

To delete a correlation group from the list, first select the group to be deleted and then click the Delete button. You'll be asked to confirm the command. If you do confirm the command the currently selected group will be deleted from the list. In addition, any correlations involving the deleted group will be removed from the correlation matrix.

4.5.2 Choosing Correlated Variables

When you have created a correlation group, the next step is to decide which variables to include in the group. To make the selection, select the correlation group from the list and then click on the Edit button (Figure 4.5). A form like the one shown in Figure 4.6 will be displayed. On the left side of this form is a list of variables not already included in other correlation groups. The list includes only variables that have a distribution other than "Fixed" assigned. To add a variable to the current group, select the variable in the list on the left and click the right arrow button. Alternatively, just double-click on the variable to move it. Similarly, to remove a variable from the current group, select the variable in the list on the right and click the left arrow button. Alternatively, just double-click on the variable to move it. Notice that a variable can be in one list or the other but not in both lists.

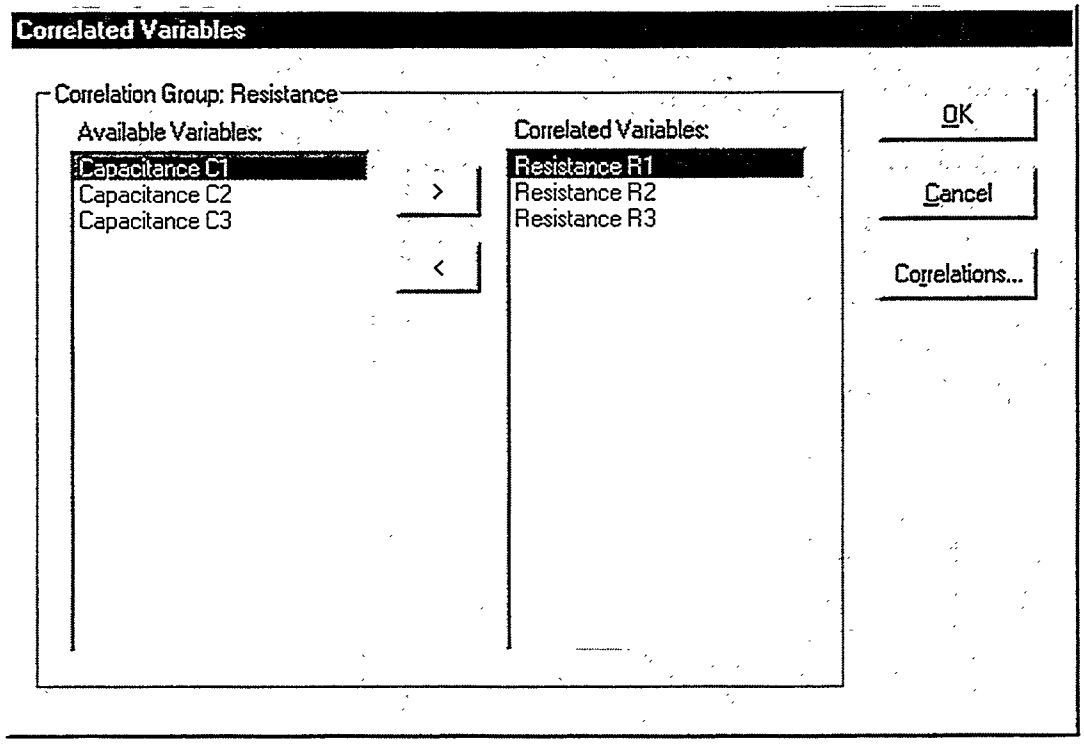


Figure 4.6 Adding Variables to a Correlation Group

4.5.3 Entering Correlation Coefficients

When you have selected the variables that you want to include in a correlation group it is time to enter the correlation coefficients. From the form shown in Figure 4.6 click on the Correlations button. You'll then see a form like the one shown in Figure 4.7. Notice that SUNS has already made up a list of all the possible pairings between the variables that you included in this correlations group. For each pairing, enter the correlation coefficient. Each coefficient must be greater than or equal to –1 and less than or equal to +1.

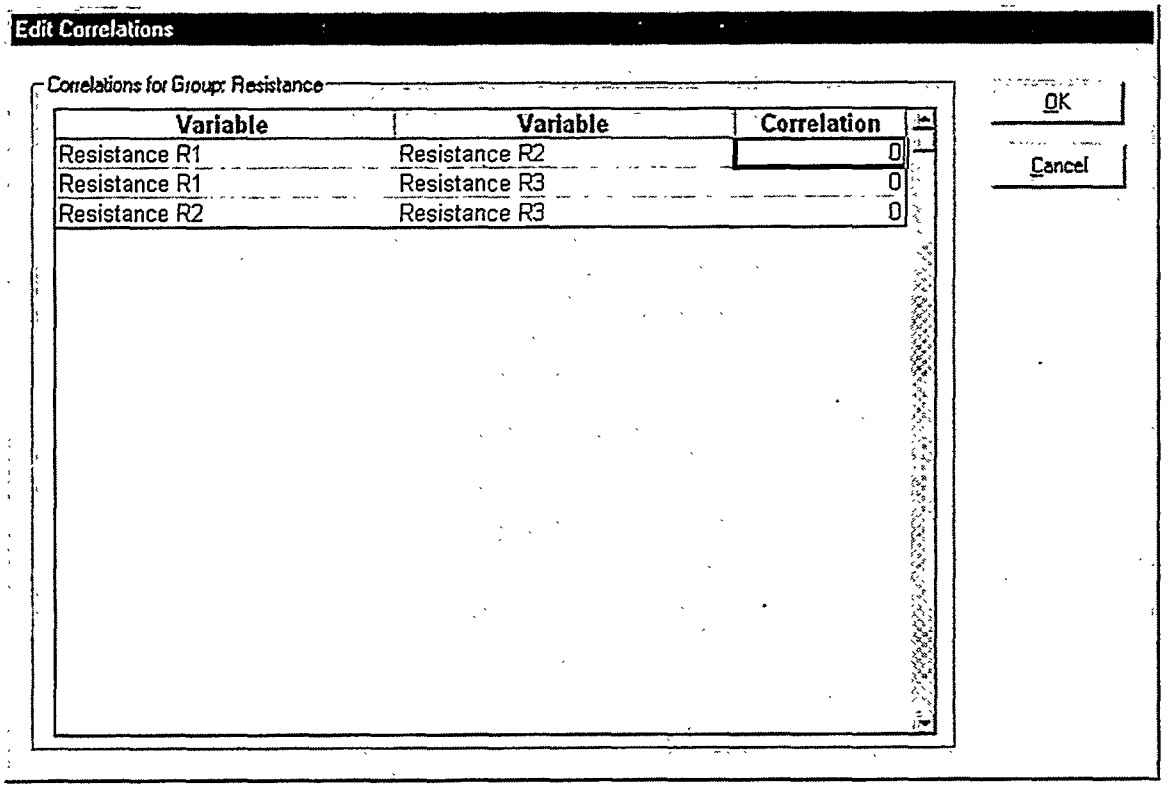


Figure 4.7 Entering Correlation Coefficients

When you have finished entering correlation coefficients, click the OK button on the form shown in Figure 4.7. If your correlations are consistent, you will be returned to the form shown in Figure 4.6. If, however, you have specified correlations that are not internally consistent, you'll see a form like the one shown in Figure 4.8, followed by a form like the one shown in Figure 4.9. Notice that an extra column of Alternate Correlations has been added.

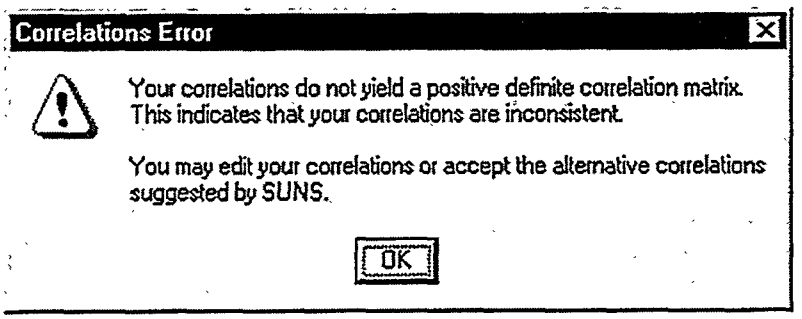


Figure 4.8 Correlation Coefficients are Inconsistent

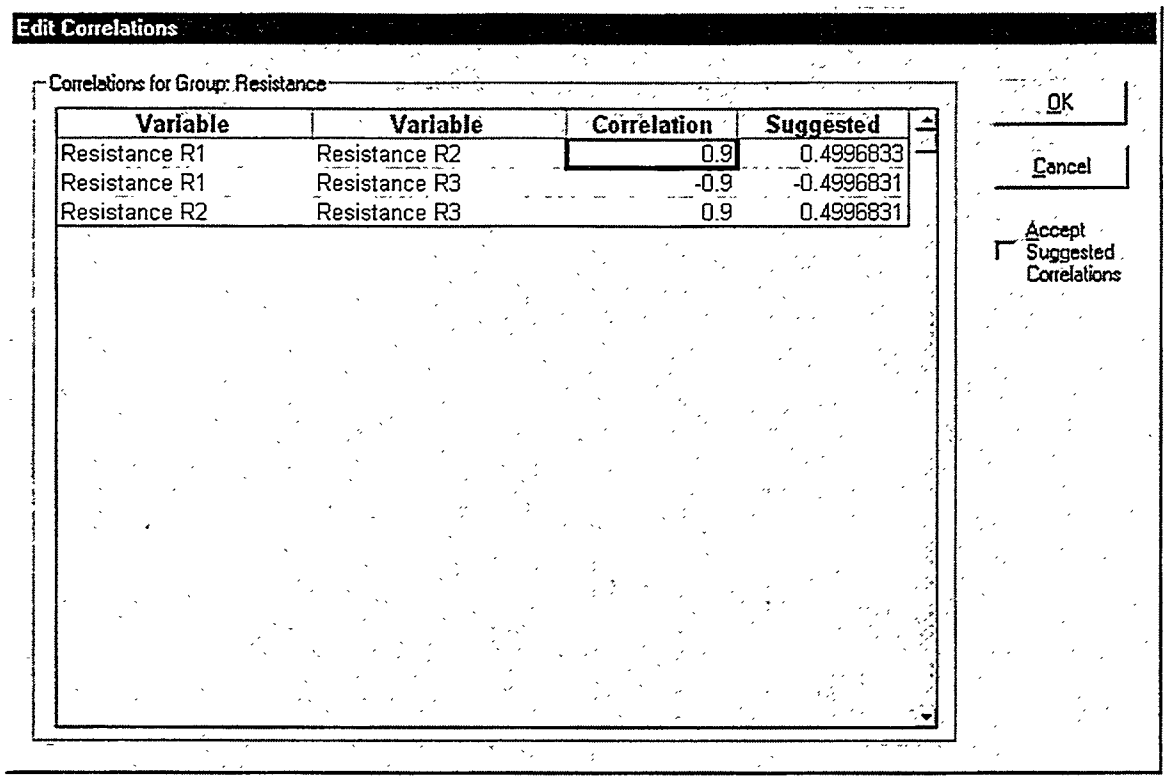


Figure 4.9 Edit Correlation Coefficients or Choose Suggested Correlations

Consider a simple example. Suppose we have three variables A, B, and C. We specify that B has a strong positive correlation with A (lets say +0.9). We also specify that C has a strong positive correlation with B (again, let's say +0.9). If we now specify that A has a strong negative correlation with C (e.g., -0.9) we are asking for something that makes no sense. The impact in such cases is a correlation matrix which is not positive definite. If SUNS detects this condition, it displays a message (Figure 4.8) followed by a form like the one shown in Figure 4.9. The Alternate correlations are the "nearest" correlations to the ones you specified that do create a positive definite correlation matrix. If you want to use the Alternate correlations, check the "Use Alternate Correlations" box and then click the OK button. Otherwise, edit your correlations as needed. SUNS will not let you proceed until you have entered valid correlations or accepted it's Alternate correlations.

4.6 Arranging the Variable Order

It is likely that the application that will use the statistical sample provided by SUNS will need sampled variables to be presented in a particular order. To see the order in which sampled variable values will be written to the SUNS output file (vec file, see Section 3.2), select Arrange Variables from the Edit menu. The form shown in Figure 4.10 will appear. This form shows you the order that variable values will be written to the SUNS output file. If you want the variables written in a different order, you can highlight a particular variable and then use the up and down buttons to move the variable to the location you desire.

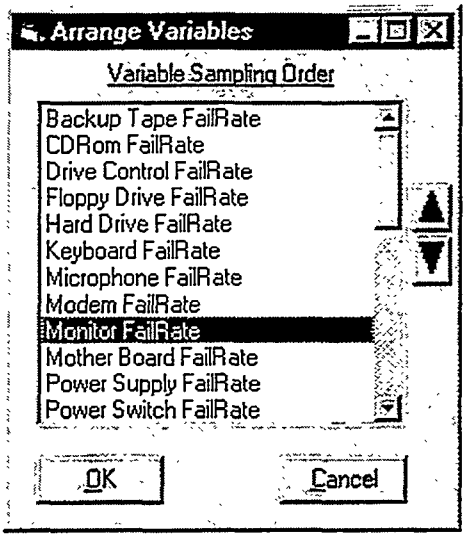


Figure 4.10 Form to View and Arrange Order of Variables

4.7 Setting Sampling Options

To edit sampling options pull down the Edit menu and selected the Options command. You'll see a form like the one shown in Figure 4.11. Here you can select the following:

Type of Sampling — Select LHS or Monte Carlo sampling.

Correlations — Indicate that you want correlations not specified to be zero. Selecting this option causes SUNS to ignore correlation groups, imposing a limit on how many variables can be correlated. This option is typically most useful for small sample sizes. These small samples are usually generated only when the application is very computationally expensive. The other option under this heading may be useful in very large problems where numerical errors cause problems in the positive definite checking that SUNS performs. You may have many correlation groups that independently yield positive definite correlation matrices. However, when these group correlation matrices are "assembled" the resulting matrix may not be positive definite. Checking this option tells SUNS to use it's alternate (i.e., "nearest" good) correlations matrix and proceed with sampling.

Sampling Parameters — Enter a random seed for the random number generator. The seed must be an integer greater than zero and less than 9,999,999. Additionally, specify the number of trials (i.e., the sample size).

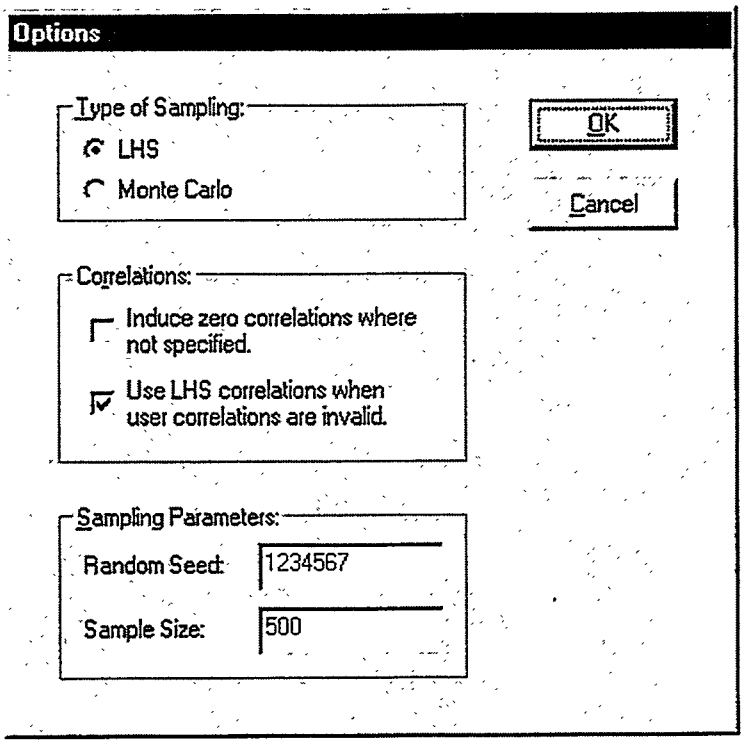


Figure 4.11 SUNS Sampling Options

4.8 Printing

SUNS allows you to print a readable text version of your sun file. To print the currently loaded file, select Print from the File menu.

4.9 Running the Sample

When you have entered variables, correlations, and options, you are ready to perform the sampling. To do so, select the Run! command. SUNS will prompt you for an output file name that, by default will be the same as the fun file but with a vec extension. When you've provided a vec filename, the sampling process begins.

4.10 Using the Toolbar

Many of the SUNS commands that have been described above are accessible directly from a toolbar as well as from the pull-down menus (Figure 4.12). To view the toolbar, choose Toolbar|Small or Toolbar|Large from the View menu. The large toolbar is shown in Figure 4.12.

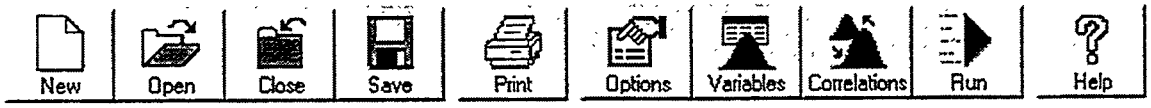


Figure 4.12 The SUNS Toolbar

Working from right to left along the toolbar in Figure 3.10, the buttons perform the following tasks:

New	Creates a new SUNS file. Performs the same function as the New command on the File Menu.
Open	Opens an existing SUNS file. Performs the same function as the Open command on the File menu.
Close	Closes the current SUNS file. Performs the same function as the Close command on the File menu.
Save	Saves the current SUNS file to disk. Performs the same function as the Save command on the File menu.
Print	Prints the current SUNS file. The printout includes options, variables, distributions, and correlations.
Options	Opens the Options form. Performs the same function as selecting Options from the Edit menu.
Variables	Opens the Variables form. Performs the same function as selecting Variables from the Edit menu.
Correlations	Opens the Correlations form. Performs the same function as selecting Correlations from the Edit menu.
Run	Begins the sampling process. You will be prompted for a file where SUNS can write the sampled values. Performs the same function as the Run! Menu command.
Help	Provides access to the online help system.

5 SUNS Results Viewer

5.1 SUNS Result Files

When using SUNS to create a set of input vectors, you first create a SUNS input file to define variables, distributions, and correlations. When you are ready to create the input vectors, you use SUNS to create a file of vectors, usually in a file with a vec extension (type). Your application uses these vectors to calculate results, append the results to the input vectors, and update the same vec file or create a new file (with a vec or some other extension).

The SUNS results viewer (Figure 5.1) can be used to display data from a file of SUNS input vectors either before or after application results have been appended. By default, the SUNS Results Viewer looks for files with the vec type, but any file containing vectors in the required format can be opened. If the file contains only input vectors (with no results appended) some items will not be available in the Results Viewer.

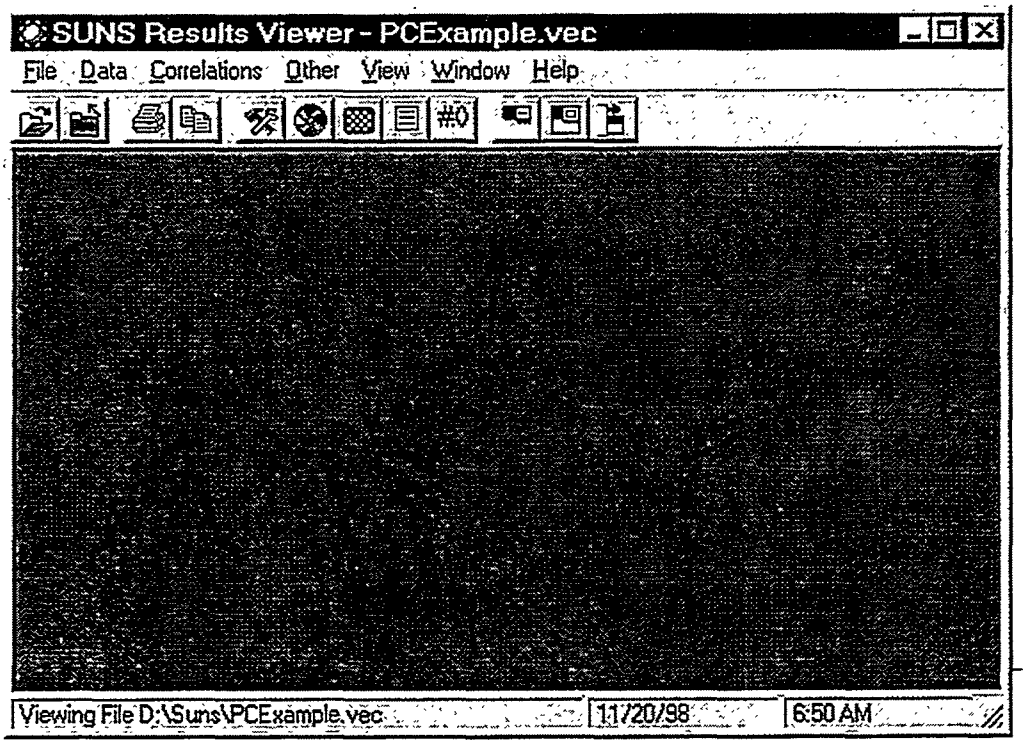


Figure 5.1 SUNS Results Viewer Main Screen

5.2 Displaying Vectors

5.2.1 Input Sample

To display information about input vectors, select Input Sample from the Data menu. Five display types are available on the sub-menu, three graphical

displays (histogram, cumulative distribution function (CDF), and complementary cumulative distribution function (CCDF)), and two text displays (list and summary statistics). After choosing a display type, you will be asked to select from the list of input variables (Figure 5.2).

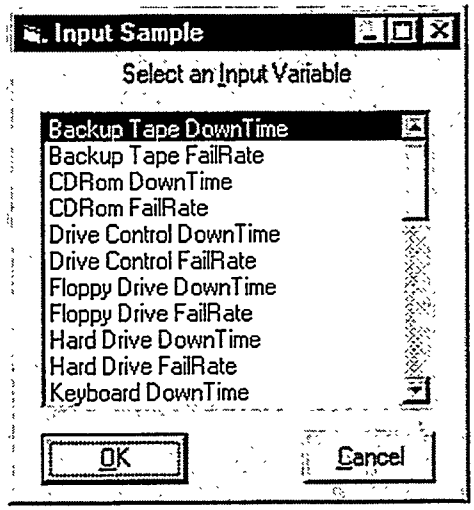


Figure 5.2 Selecting a Variable

Figures 5.3 through 5.7 show examples of the five displays available.

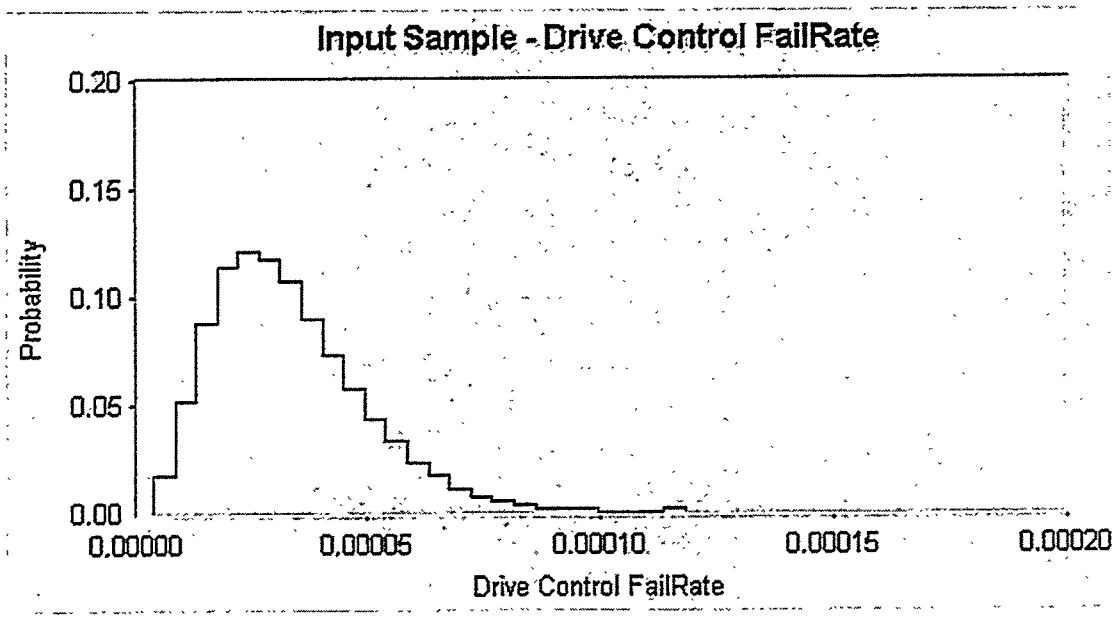


Figure 5.3 Histogram of Input Variable Sample

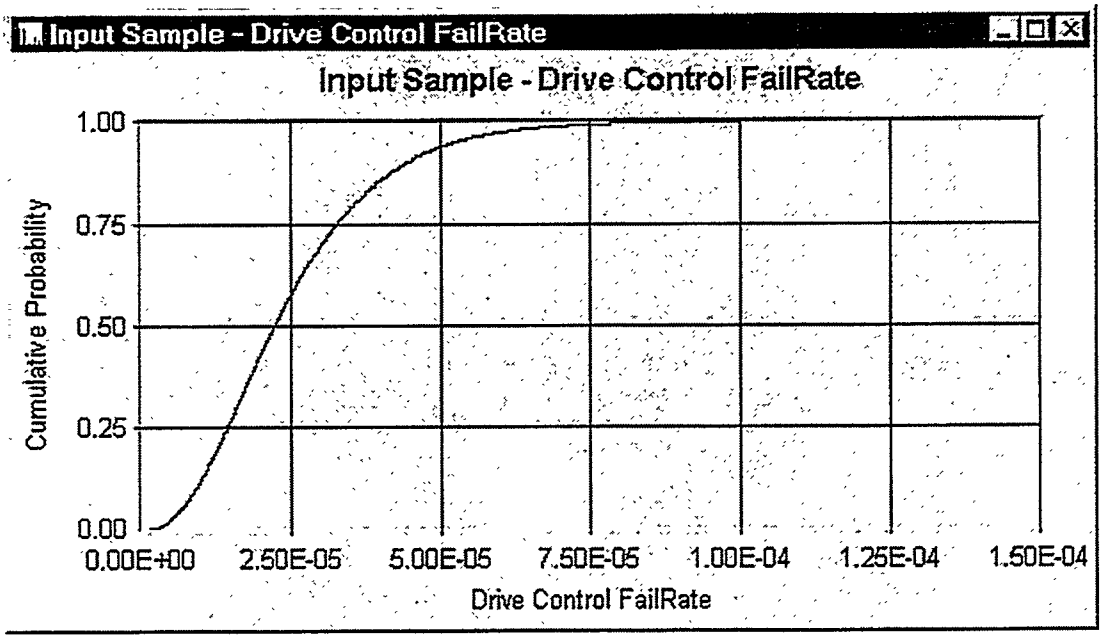


Figure 5.4 Cumulative Distribution Function of Input Variable Sample

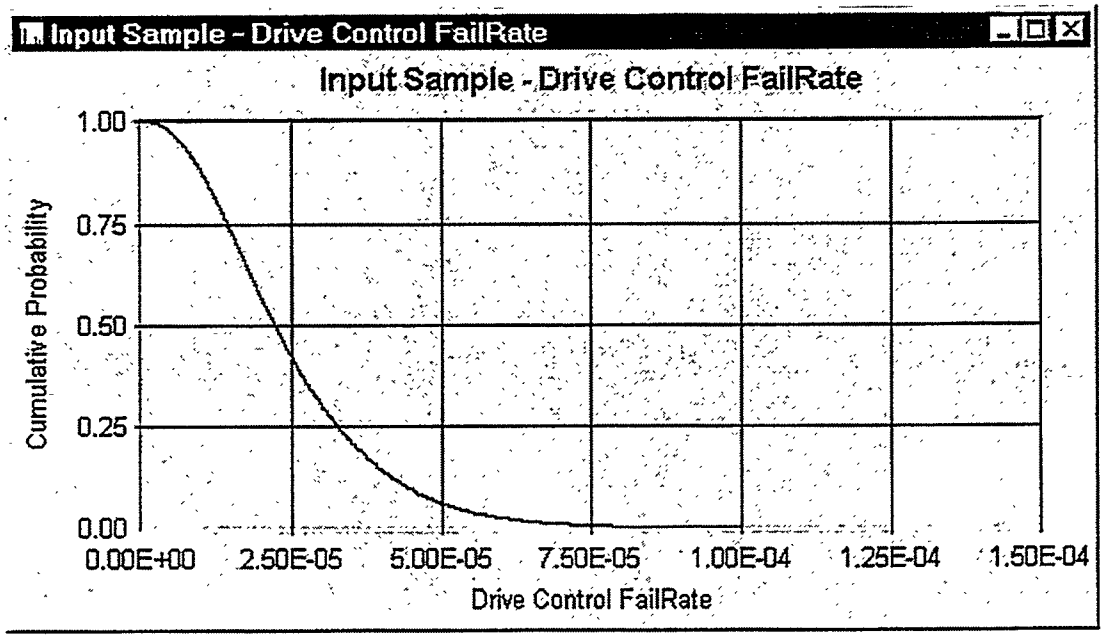


Figure 5.5 Complementary Cumulative Distribution Function of Input Variable Sample

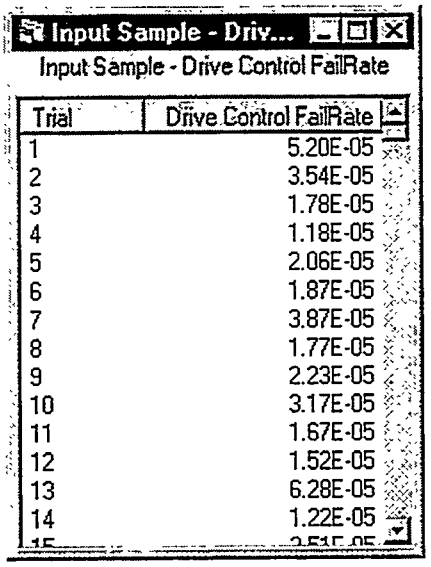


Figure 5.6 List of Input Variable Sampled Values

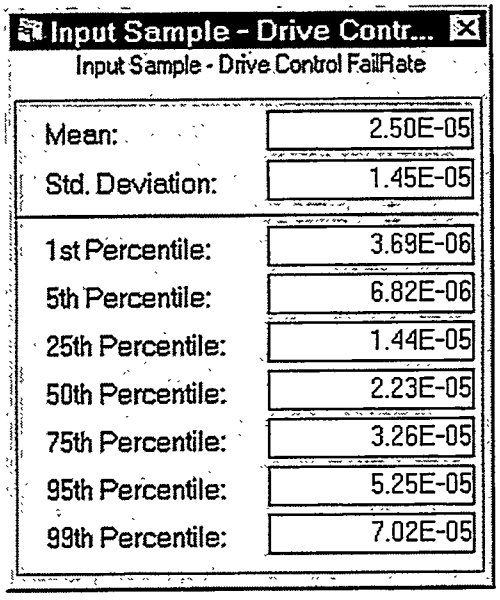


Figure 5.7 Summary Statistics of Input Variable Sampled Values

5.2.2 Output Displays

To display information about result vectors, select Output Results from the Data menu. Five display types are available on the sub-menu, three graphical displays (histogram, cumulative distribution function (CDF), and complementary cumulative distribution function (CCDF)), and two text displays (list and summary statistics). After choosing a display type, you will be asked to select from the list of output results (Figure 5.8). Figures 5.3 through 5.7 show examples of the five displays available.

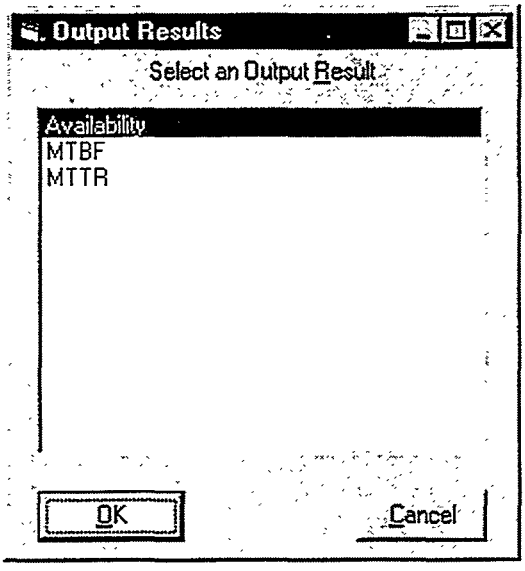


Figure 5.8 List of Available Output Results

5.3 Displaying Correlations

The SUNS Results Viewer can display simple and partial correlations. Each type of correlation may be displayed for raw data values or ranks. Choose the type of correlation you want to display from the Correlations menu. The information can be displayed as a Pareto chart or as a text list.

You must decide whether to look at the correlations between a selected output and all input variables or at the correlations between a selected input variable and all outputs. The choice is made on the form shown in Figure 5.9.

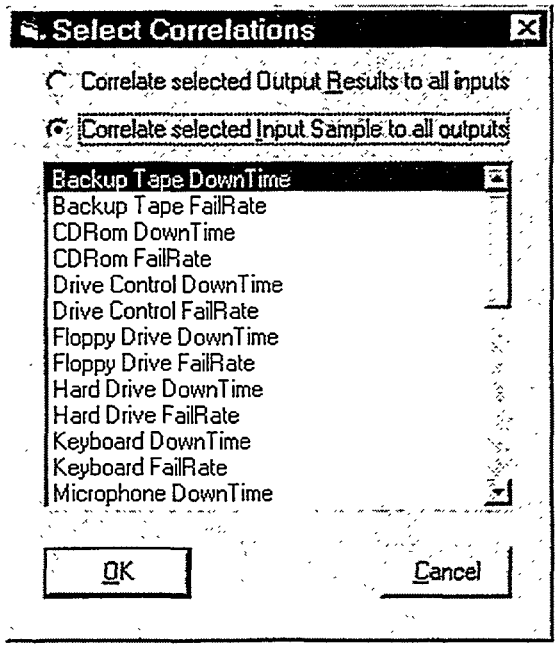


Figure 5.9 Selecting the Correlations to be Displayed

When you have made a selection from the form shown in Figure 5.9, the correlations are shown using the display type that you chose. An example of the Pareto display of simple raw correlations is shown in Figure 5.10.

5.4 Other Displays

The SUNS Result Viewer can provide several additional results and displays. These include Uncertainty Importance as well as scatter plots and a spreadsheet display of all input/result vectors.

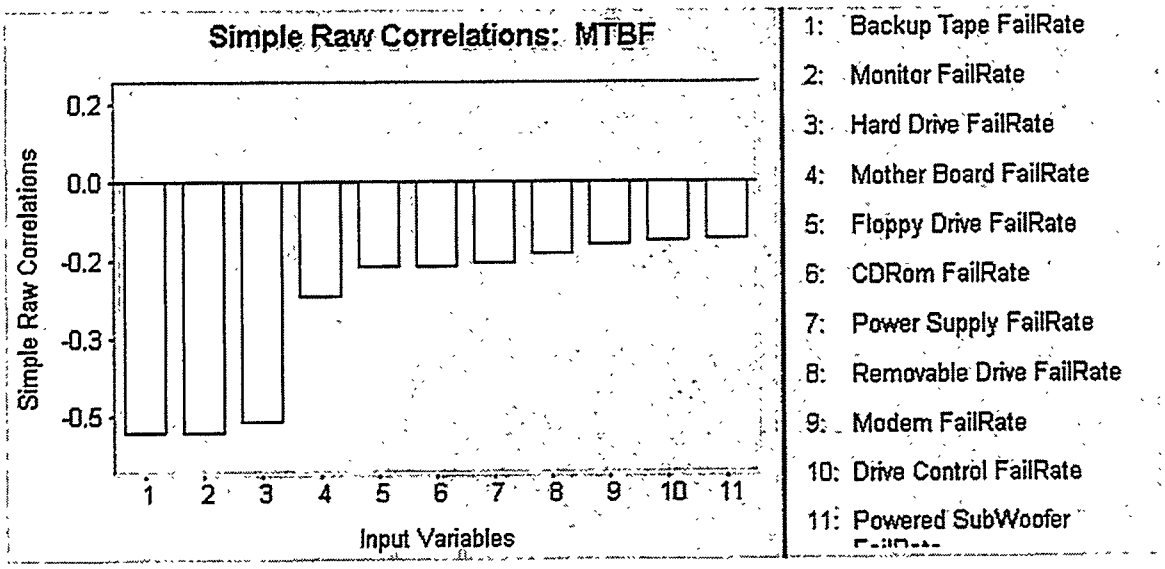


Figure 5.10 Pareto of Simple Raw Correlations

5.4.1 Uncertainty Importance

To display uncertainty importance, select Uncertainty Importance from the Correlations menu. After choosing a display type (Pareto or text list), you must choose an output variable for which uncertainty importance will be calculated for all input variables. Note that the uncertainty importance calculation requires that all input variable sample values be positive. If this is not the case, you will see a warning message and uncertainty importance will not be displayed.

5.4.2 Input and Result Vectors

To display the complete set of input/output vectors, select Display Data from the Other menu. You can then choose to display only input vectors, only output vectors, or both input and output vectors. If you open a file that does not yet have output values appended to the input values, the SUNS Results Viewer will not offer output values as an option. The selected data will be displayed in a form like the one shown in Figure 5.11. This form allows the data to be sorted and the numerical values reformatted. This can be useful when dealing with very large or small numbers, or with numbers that vary greatly in range.

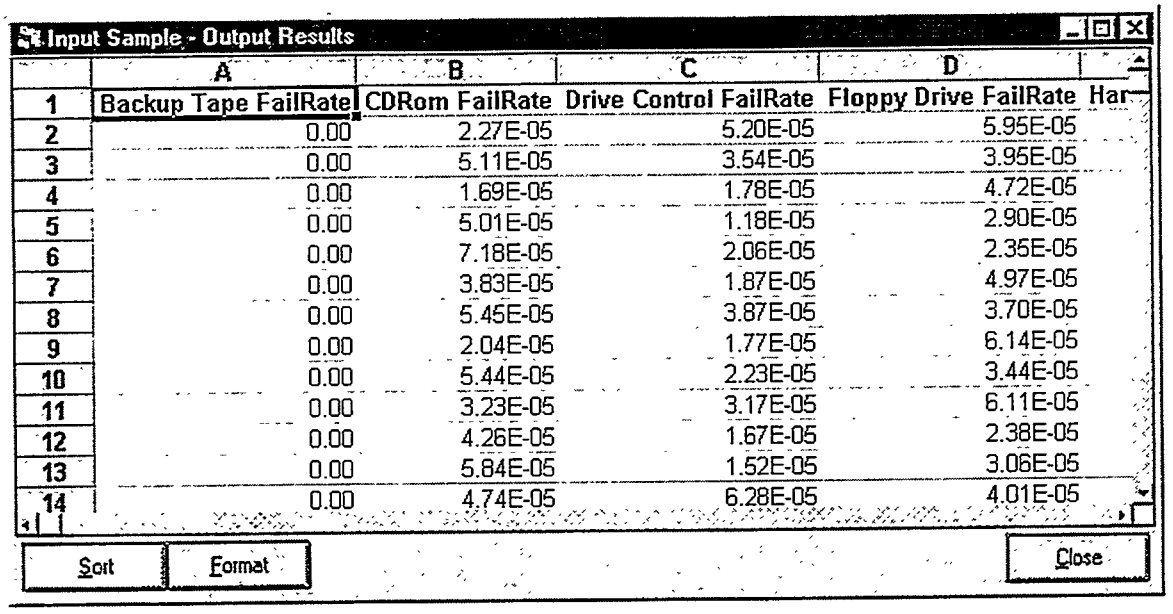


Figure 5.11 Grid Display of Input and Output Values

5.4.3 Scatter Plots

To display a scatter plot of one input variable and one or more output variables, select Scatter Plot from the Other menu. You will see a form like the one shown in Figure 5.12. Select an input variable from the list on the left and one or more output variables from the list on the right. To select more than one output variable from the list, hold down the Control (CTRL) key while making you selection. When you have made your selections, a scatter plot like the one shown in Figure 5.13 will be displayed.

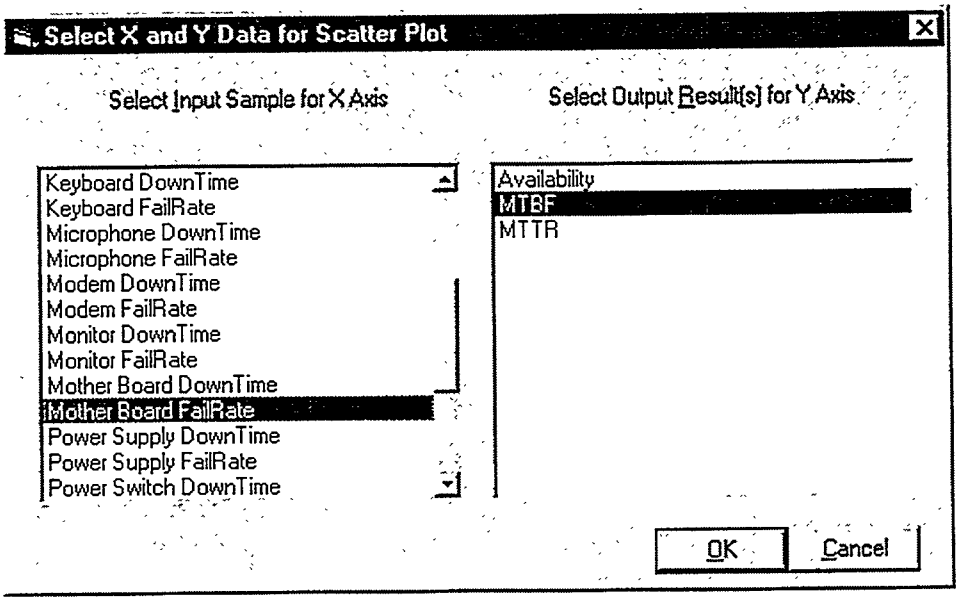


Figure 5.12 Select Input and Outputs for Scatter Plot

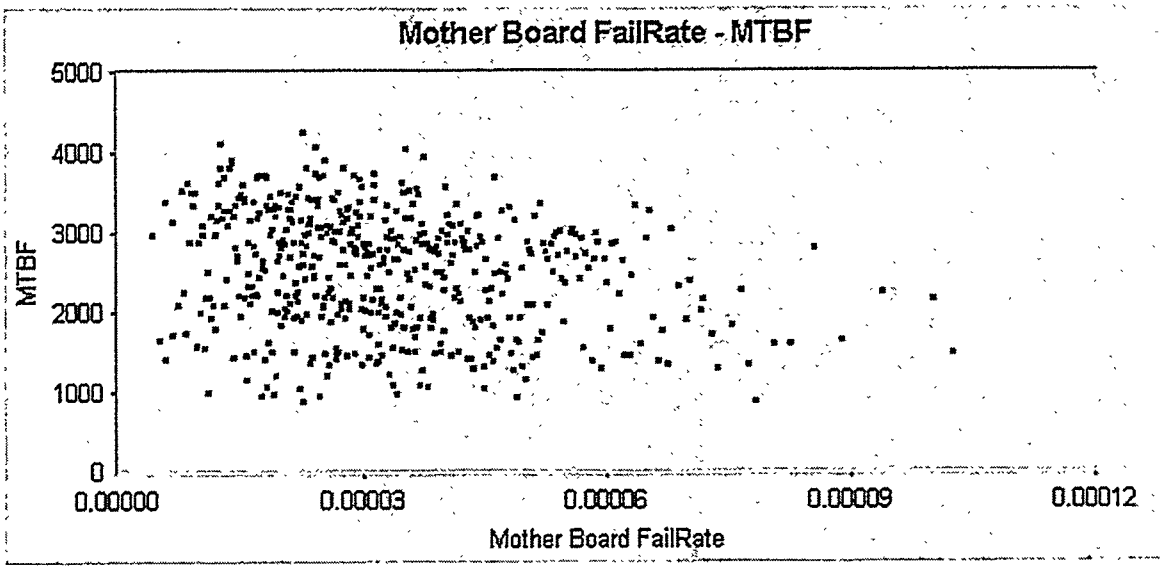


Figure 5.13 Scatter Plot

5.5 The Results Viewer Toolbar

Figure 5.14 shows the SUNS Results Viewer toolbar. The toolbar can be displayed in a large or small format by selecting the appropriate command from the **View** menu.

Figure 5.14 The SUNS Results Viewer Toolbar

Working from right to left along the toolbar in Figure 5.14, the buttons perform the following tasks:

Open	Opens an existing SUNS file containing input vectors optionally appended with output values. Performs the same function as the **Open** command on the **File** menu.
Close	Closes the current SUNS file. Performs the same function as the **Close** command on the **File** menu.
Print	Prints the active results from. If the active form contains a graph, the graph will be printed. If the active form contains text data, the text will be printed.
Copy	Copies the active results form to the Windows clipboard. If the active form contains a graph, the graph will be copied. If the active form contains text data, the text will be copied.
Toolbar	Adds a graphics toolbox to the active form if the form is displaying graphics.
ColorBar	Adds a color bar to the active results form if the form is displaying graphics.

Patterns	Adds a pattern bar to the active results form if the form is displaying graphics.
Memo	Adds an area at the botton edge of the current form where you can type in a note. This can be useful, for example, for recording pertinent information with a graph or table before copying results from the SUNS Results Viewer to a report.
Format	Adds an area to the bottom edge of the current results form where you can change the formatting applied to numeric values in the display. You can choose standard formatting (the default), scientific formatting, or custom formatting. With the custom formatting option, you enter an Excel-like formatting string to specify how numeric values are displayed.
Minimize	Minimizes the active result form.
Maximize	Maximizes the active result form.
Hide	Makes the active result form invisible. To redisplay the form, choose the appropriate command from the **View** menu.

6 SUNS Example

An example based on repairable systems reliability has been developed to illustrate the use of SUNS in a practical application. The application on which the example is based was written in Visual Basic 5. Both executable and source code files as well as example problem input and results are included in the installation files. The example application (called TestApp) calculates Mean Time Between Failures (MTBF), Mean Time to Repair (MTTR), and availability for a series system. The required inputs for each failure mode are:

Failure Rate: The failure rate for a particular failure mode is the number of occurrences per hour of operational time. For example, a failure rate of 0.0001 would mean that the particular failure mode occurs, on average, once every 10,000 hours.

Down Time: Down time (also called repair time) is the total time in hours required to return the system to operation after a particular failure mode occurs. For example, a down time of 2.5 would mean that, on average, the failure mode takes 2.5 hours to repair and return the equipment to operation.

MTBF, MTTR and availability are calculated from the following equations:

$$\mathrm{MTBF} = \frac{1}{\sum_{j=1,N} \lambda_j}$$

$$\mathrm{MTTR} = \frac{\sum_{j=1,N} \lambda_j \tau_j}{\sum_{j=1,N} \lambda_j}$$

$$\mathrm{Availability} = \frac{\mathrm{MTBF}}{\mathrm{MTBF} + \mathrm{MTTR}}$$

where λ_j is the failure rate for failure mode i and τ_j is the down time.

6.1 Input Data

The example is based on hypothetical field failure data for 20 personal computers operated for 3 years with average use of 2000 hours per year. Input failure rate data for this example is given in Tables 6.1and 6.2. Down time distributions are given in Table 6.3. For the down time input variables, uniform distributions use Parameter 1 as a minimum value and Parameter 2 as a maximum value. Triangular distributions take Parameters 1, 2, and 3 as minimum, best estimate, and maximum values respectively.

Failure Rate Variables	Distribution	Shape	Scale
Backup Tape FailRate	Empirical Cont. Freq.	*	
CDRom FailRate	Gamma	4	120000
Drive Control FailRate	Gamma	3	120000
Floppy Drive FailRate	Gamma	5	120000
Hard Drive FailRate	Empirical Cont. Freq.	*	
Keyboard FailRate	Gamma	2	120000
Microphone FailRate	Gamma	1	120000
Modem FailRate	Gamma	2	120000
Monitor FailRate	Empirical Cont. Freq.	*	
Mother Board FailRate	Gamma	3	120000
Power Supply FailRate	Gamma	3	120000
Power Switch FailRate	Gamma	1	120000
Powered SubWoofer FailRate	Gamma	3	120000
Removable Drive FailRate	Gamma	4	120000
Speaker FailRate	Gamma	1	120000
Video Control FailRate	Gamma	2	120000

* See Table6.2 below.

Table 6.1 Failure Rate Distributions for Example Problem

Empirical Failure Rate Distributions	Value	Relative Freq.
Backup Tape FailRate	0	17
	0.000167	2
	0.000333	1
Hard Drive FailRate	0	13
	0.000167	6
	0.000333	1
Monitor FailRate	0	17
	0.000167	2
	0.000333	1

Table 6.2 Empirical Failure Rate Distributions

Down Time Variables	Distribution	Par. 1	Par. 2	Par. 3
Backup Tape DownTime	Uniform	1	4	
CDRom DownTime	Uniform	1	4	
Drive Control DownTime	Triangular	1	2	4
Floppy Drive DownTime	Uniform	1	2	
Hard Drive DownTime	Triangular	2	5	8
Keyboard DownTime	Uniform	0.25	1	
Microphone DownTime	Uniform	0.25	1	
Modem DownTime	Triangular	1	2	4
Monitor DownTime	Uniform	0.25	1	
Mother Board DownTime	Triangular	6	12	26
Power Supply DownTime	Uniform	2	6	
Power Switch DownTime	Triangular	2	6	8
Powered SubWoofer DownTime	Triangular	0.25	0.5	1
Removable Drive DownTime	Triangular	2	3	5
Speaker DownTime	Triangular	0.25	0.5	1
Video Control DownTime	Uniform	2	4	

Table 6.3 Down Time Distributions for Example Problem

6.2 Example Application

The example application (TestApp) is written in Visual Basic 5. Both an executable program and source code are included in the installation package. There is only one user-interface form in TestApp which is shown in Figure 6.1.

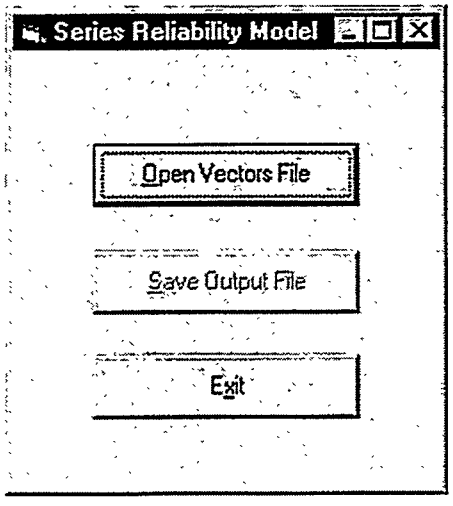


Figure 6.1 TestApp Main Form

You open a SUNS output (vec) file by clicking the Open Vectors File button. When a valid file has been opened, the Save Output File button is enabled. To save a file with MTBF, MTTR, and availability results, just click the Save Output File button.

Figure 6.2 shows a histogram of MTBF values for the example problem. Summary statistics are shown in Table 6.4.

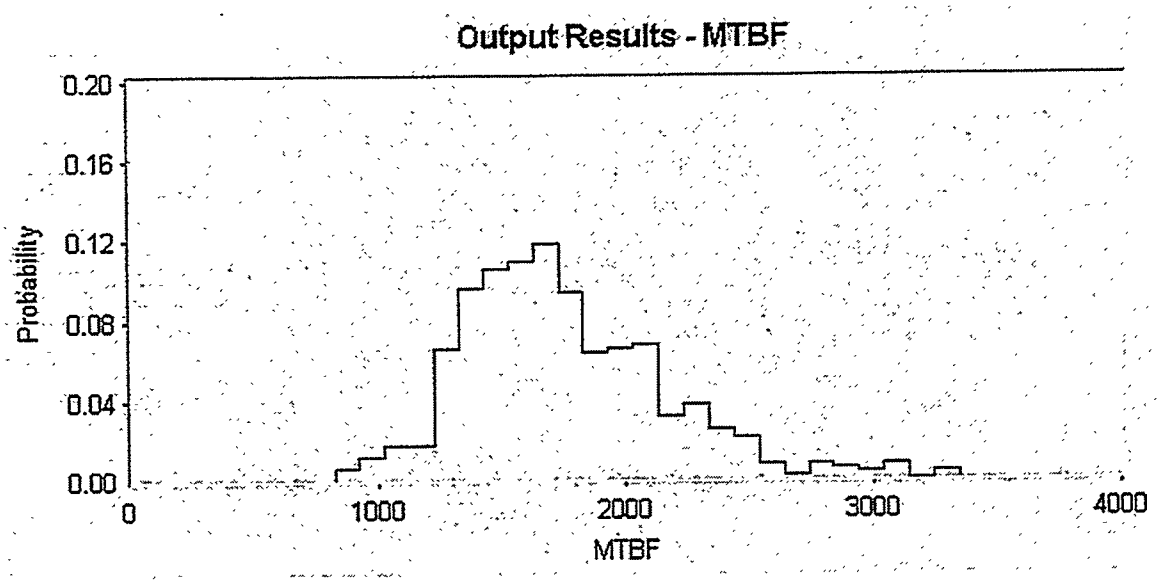


Figure 6.2 Histogram of MTBF Values

Statistic	Value
Mean	1,755.17
Standard Deviation	412.01
1st Percentile	986.51
5th Percentile	1,227.05
25th Percentile	1,475.92
50th Percentile	1,688.75
75th Percentile	1,990.00
95th Percentile	2,477.42
99th Percentile	3,067.72

Table 6.4 Summary Statistics for MTBF

Figure 6.3 shows a Pareto of partial raw correlations indicating that MTBF is most highly correlated with the hard drive failure rate followed by the monitor and backup tape failure rates. The correlations are negative because an increase in a failure rate causes a decrease in MTBF.

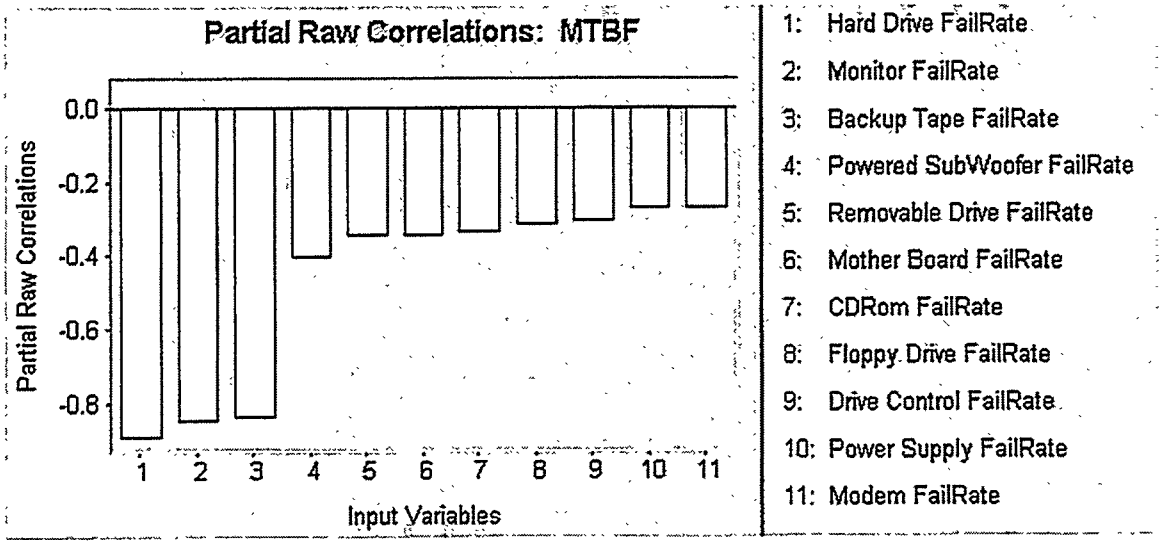


Figure 6.3 Partial Raw Correlations for MTBF

Given the high partial raw correlation between MTBF and hard drive failure rate, it is instructive to look at a scatter plot for these two variables. The scatter plot, Figure 6.4, clearly shows that MTBF is strongly dependent on hard drive failure rate.

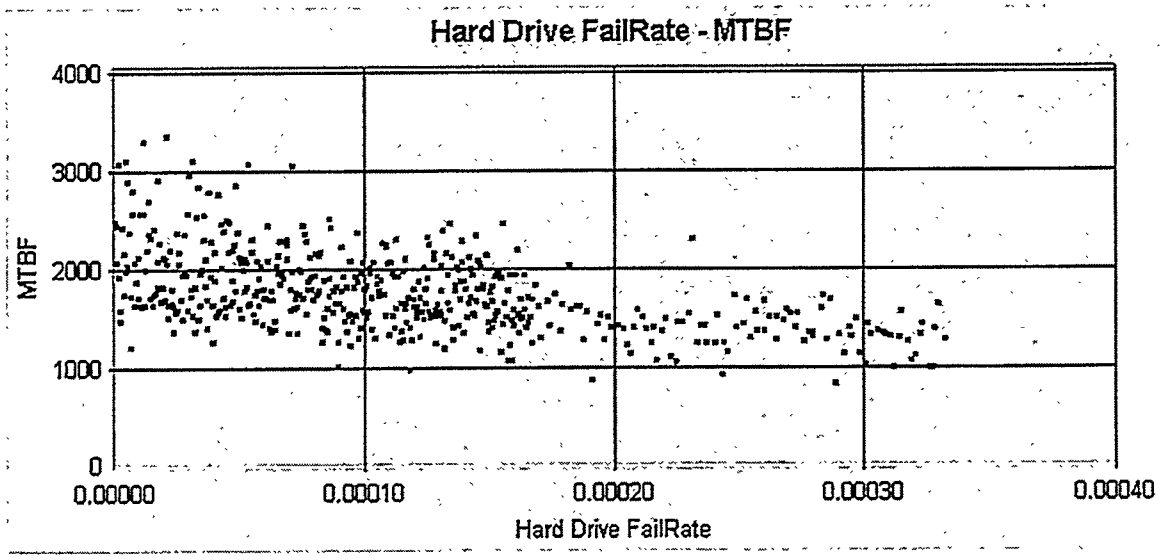


Figure 6.4 Scatter Plot of MTBF vs Hard Drive Failure Rate

We leave it to the user to explore other outputs from the example problem. If you plan to couple an application with SUNS, you may want to review the source code for the example. Source files are included in the installation package and can be examined with any text editor. In particular, you should examine the subroutines that read the SUNS vec file and write the resulting output file.

7 References

[1] Iman, R. L. and M. J. Shortencarier, 1994. *A FORTRAN 77 Program and user's Guide for the Generation of Latin Hypercube and Random Samples for Use With Computer Models*, NUREG/CR-3624, SAND83-2365, Sandia National Laboratories, Albuquerque, NM.

[2] Iman, R. L., and W. J. Conover, 1980. *Small Sample Sensitivity analysis Techniques for Computer Models, with an Application to Risk Assessment*, Communications in Statistics A9: 1749-1842.

[3] Iman, R. L., and W. J. Conover, 1982. *A Distribution-Free Approach to Inducing Rank Correlations Among Input Variables*, Communications in Statistics B11: 311-334.

[4] Iman, R. L., J. C. Helton, 1988. *An Investigation of Uncertainty and Sensitivity Analysis Techniques for Computer Models*, Risk Analysis 8: 71-90.

This page intentionally left blank

Appendix B: GO™ Genetic Optimization User's Reference Manual

This page intentionally left blank

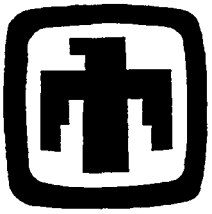
Center for System Reliability

GO-Genetic Optimization User's Reference Manual

Version 1.0

Sandia National Laboratories
Albuquerque, New Mexico

November, 1998



"Exceptional Service in the National Interest"

This page intentionally left blank

Contents

Figures

Tables

1.0 Getting Started

1.1 Introduction to GO

GO is a 32-bit genetic optimization driver that runs under Windows 95/98 and Windows NT. Genetic optimization refers to an optimization scheme used to solve large combinatorial problems using "genetic" algorithms. Genetic algorithms take their inspiration from the biological world. They operate by creating an initial "population" of solutions (usually represented as strings of integers) that "evolve" over successive generations. Each solution string is called a chromosome and chromosomes are composed of genes. Each integer value in the string represents a possible level or value for that gene. To evolve from one generation to the next, the solutions with high "fitness" are "mated" with other solutions by crossing parts of a solution string with another. Solution strings are also "mutated" by replacing the value of a randomly selected integer in a solution string with another value. Over time, the operations of weeding out poor fitness solutions and reproducing by crossing high fitness solutions at random points act to sample the state space very efficiently. As in natural selection, genetic algorithms process fitness information and rank solutions according to their survival capabilities. In genetic optimization, fitness refers to the value of the objective function.

Many optimization problems can be easily formulated in a genetic algorithm framework. Genetic algorithms have been applied to combinatorial optimization problems in engineering design and reliability. We have successfully applied genetic algorithms to reliability improvement problems, reliability allocation, and spare parts inventory analysis. These include problems that are naturally combinatorial (e.g., spares inventories) and problems that involve continuous variables that can be discretized (reliability allocation).

This document provides a brief introduction to genetic optimization capabilities provided by GO.

Note: The version of GO documented here is at the beta development stage.

1.2 System Requirements

To install GO you will need an IBM-compatible PC with a 486 or Pentium processor. A Pentium-based system is recommended. GO is a 32-bit Windows application and will under Windows 95/98 and Windows NT. The installation requires approximately 5 MB of free hard drive space. The amount of RAM required to run GO is dependent on the memory requirements of the application you are going to use with GO.

To use GO you will need to be familiar with basic Windows functions such as menus, dialogs, files, and mouse operation. If you are not accustomed to using Windows applications, please refer to your Microsoft Windows User's Guide. To link your application with GO you will also need a detailed knowledge of your

application source code. A detailed description of the file format used to transfer data to and from your application process model is provided in Chapter 4, *Linking GO with Your Application.*

1.3 Installing GO

GO must be installed on your computer using the provided Setup program. The program files are compressed and cannot be used by copying them directly to your hard drive.

To install GO from floppy disks:

1. Start Windows.
2. Insert GO Disk 1 into your floppy disk drive (A: or B:).
3. Choose "Run..." from the Start menu.
4. Type A:\Setup (or B:\Setup) in the Command Line text box.
5. Follow the on-screen instructions.

To install GO from a CD-ROM:

1. Start Windows.
2. Insert the GO CD-ROM into your CD-ROM drive (typically D:).
3. Choose "Run..." from the Start menu.
4. Type D:\Setup (where D: is the letter for your CD-ROM drive) in the Command Line text box.
5. Follow the on-screen instructions.

When you run the Setup program, you'll see a form like the one shown in Figure 1.1. If you want to install GO in the default directory, click the large Install button to proceed. If you want to install GO in some other location, click the Browse button to select an alternate location. Then proceed with the installation by clicking the install button.

As part of the installation process, Setup will create a Program item "GO Driver" on your Start | Programs menu. In addition to the program files, Setup will also install some example files and an online version of this User's Manual.

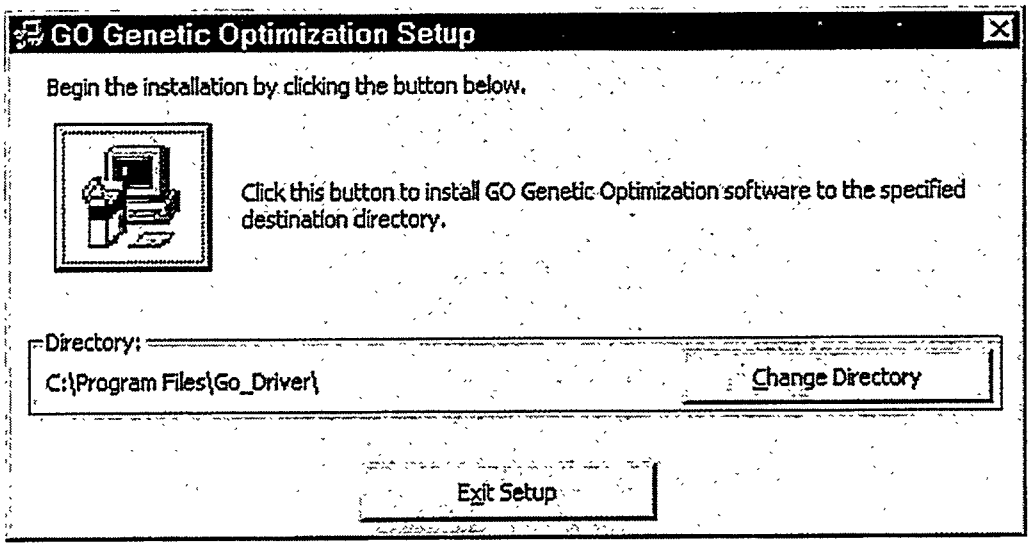


Figure 1.1 GO Installation Form

1.4 User's Manual

The GO documentation set consists of this User's Reference Manual (paper and on-line versions), a context-sensitive help system, and the Readme.txt file copied to your GO porgram directory during installation. The GO User's Manual provides information on genetic optimization and how to use the GO software. Here's what you'll find in each chapter of this manual:

Chapter 1, *Getting Started*, provides system requirements and information on how to install GO on your computer.

Chapter 2, *An Overview of Genetic Optimization*, describes in general how genetic optimization algorithms work and how they are implemented in GO.

Chapter 3, *GO Optimization Input*, provides a detailed description of how to enter the information required for an optimization problem in GO.

Chapter 4, *Linking GO with Your Application*, defines how the GO optimization driver communicates with your application. This includes a detailed description of the file format used to transfer data to and from your process model.

Chapter 5, *GO Results*, shows you how to display the results of an optimization analysis, both during the analysis and at a later time.

Chapter 6, *GO Example Application*, provides a simple example that uses the GO software to address optimization problems.

1.5 In Closing...

If you run into difficulties and cannot find a solution in the documentation, please consult the Readme.txt file installed in your GO program directory. It contains information that has become available since the manual was written and also contains contact information if you need help.

Like most software, GO has benefited from the comments and suggestions of others. If you have comments or suggestions for improvements, please share them with the author. In addition, we would like to hear about your applications of GO.

Lastly, please keep in mind that GO is copyrighted software. You may make backup copies of the software for your own use but you may not distribute the software or documentation to others without prior consent.

Microsoft, Windows, Windows 95, Windows 98, and Windows NT are trademarks of Microsoft Corporation. Pentium is a trademark of Intel Corporation. GO is a trademark of Sandia Corporation.

2.0 An Overview of Genetic Optimization

2.1 Introduction

Genetic algorithms take their inspiration from the physical world. Genetic algorithms operate by creating an initial population of solutions, often represented by bit strings or strings of integers, that evolve over successive generations. The solutions with high fitness are mated with other solutions by crossing parts of a solution string with another. Solution strings are also mutated. Over time, the process of 1) weeding out poor fitness solutions, and 2) reproducing by crossing high fitness solutions at random points, act to randomly sample a large part of the huge solution space very efficiently. Artificial reproduction schemes were first developed in the 1970s [1] and became popular in the 1980s [2,3]. Sandia National Laboratories has successfully applied genetic algorithms to several problems in equipment reliability [4,5].

Genetic algorithms search solution spaces effectively by recombining and maintaining useful *schema* (building blocks) in the population. Each population member samples all the possible *schema* to which its bits belong. For example, the bit-string 1001110 samples the region of space 1######. It also samples #00####, etc. In this way, extensive schema in the space are implicitly sampled. This inherent sampling ability of genetic algorithms is called implicit parallelism. This refers to the sampling of numerous schema and the effective resampling of schema since good schema are maintained in the population over generations.

2.2 The GO Genetic Algorithm

GO creates the initial generation using either a stratified sampling scheme (Latin Hypercube Sampling, LHS) or random initialization. The genetic algorithm is applied to the second and successive generations. Keeping population members from the previous generation creates a portion of the new generation. The *keeping* process uses either tournament or proportional selection. Tournament selection randomly picks two population members and keeps the one having the highest fitness. Proportional selection picks a population member with probability proportional to the member's fitness. All selection is done with replacement. After keeping some members from the previous population, the remainder of the new population is created using mating (illustrated in Figure 2.1).

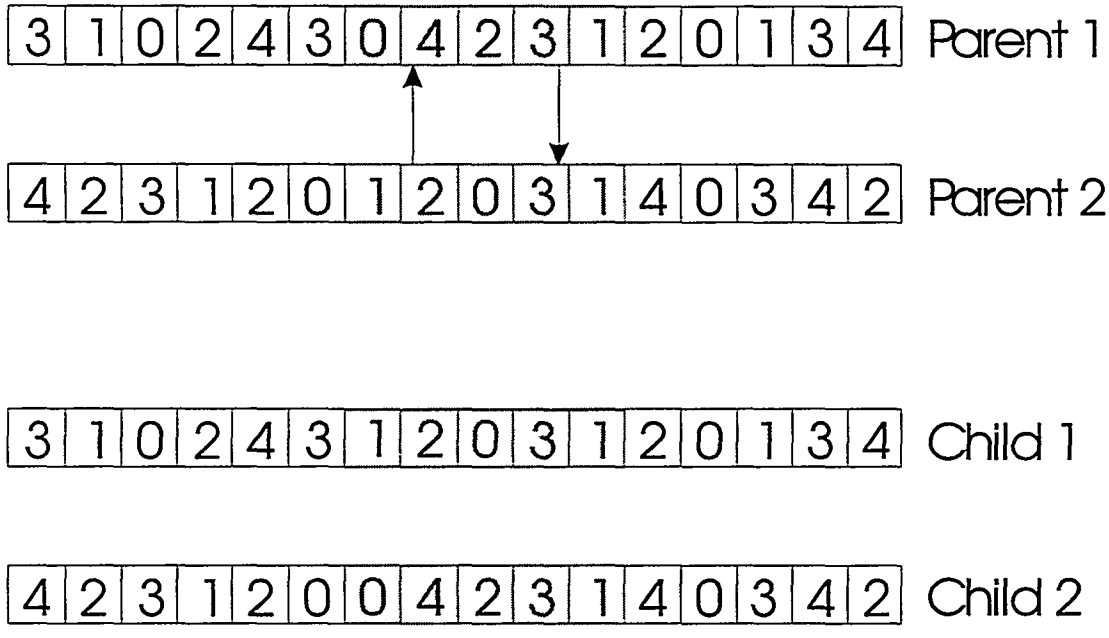


Figure 2.1 Mating Two Population Members

Two parents are selected using either tournament or proportional selection. Mating the parents then creates two offspring for the new generation. To mate the parents, GO randomly selects two points in the chromosome, then swaps the genetic material between these two points. The two children are placed in the new generation.

Once the new generation is filled, some members of the new generation are randomly selected for mutation (Figure 2.2).

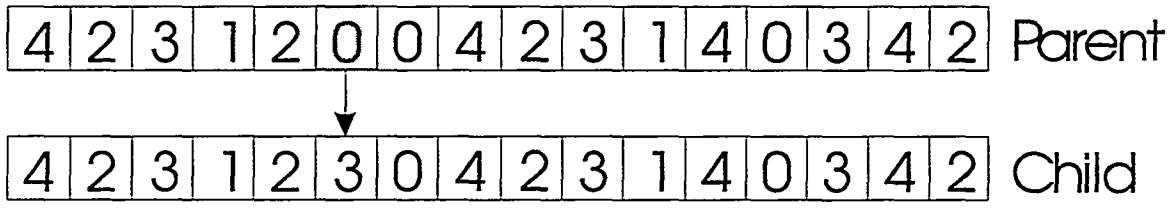


Figure 2.2 Mutating a Gene

Mutation randomly selects one (or a few) genes then randomly changes them to another acceptable value for that gene.

2.3 Calculating Fitness

Chromosomes or population members are selected for keeping and mating based on their fitness. Fitness values for the population members drives the process of evolving better and better results as each new generation is created. Thus, the methods used to calculate fitness are crucial to efficient optimization. GO bases its fitness calculation on performance measures and constraint values

calculated by your application. Up to 10 performance measures and 10 constraints can be included in the fitness calculation. Because different performance measures or constraints will generally have different units, they must be converted to dimensionless values before being incorporated into the fitness calculation. This dimensionless conversion is illustrated in Figure 2.3 for the case where the limiting value is less than the objective.

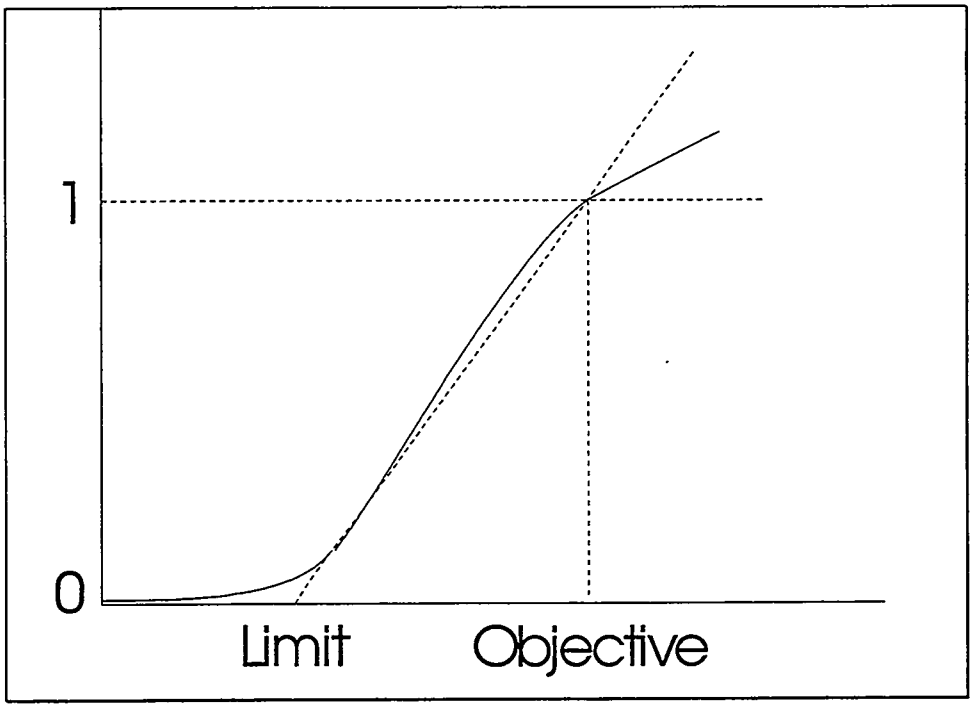


Figure 2.3 Fitness Calculation

The limit shown in Figure 2.3 is the minimum acceptable value of the performance measure or constraint. The objective is the desired value. A third property, called the relative importance, determines the slope of the curve beyond the objective. If the relative importance is set to 0, the curve is horizontal to the right of the objective. If the relative importance is set to 1, the curve follows the upper dashed line beyond the objective. For values between 0 and 1, the curve falls between the two dashed lines. The implication is that values of the performance measure higher than the objective result in higher fitness when the relative importance is close to 1 but smaller gains in fitness result if the relative importance is close to 0. In cases where the limit is larger than the objective, the curve in Figure 2.3 is simply reversed.

Fitness is calculated using the following equation:

$$F = \left[\sum_{k=1}^{N_p} (P_k w_k) \right] \prod_{j=1}^{N_c} C_j$$

where

F = Fitness;

P_k = Performance measure k after applying its dimensionless conversion;

W_k = Relative weight applied to performance measure k;

N_p = Number of performance measures;

C_j = Constraint j after applying its dimensionless conversion; and

N_c = Number of constraints.

You can see from the fitness equation that improving any of the performance measures can increase fitness. If the optimization is unable to meet the objective on any single performance measure, it does not necessarily mean that the fitness will be low. On the other hand, a single constraint that is outside its acceptable range will cause a low fitness value.

2.4 Implementing Genetic Optimization

Optimization analysis requires a processing loop. That is, GO will provide an initial population of solution strings or chromosomes that your application will analyze. Based on the results of these analyses, GO will provide a second generation of solution strings designed to improve on the previous results, and so on. The steps are as follows:

1. The GO optimization driver creates an initial population of solution strings. Every population member contains a value for each gene to be included in the optimization analysis,
2. Your application analyzes all the solution strings in the population, and
3. Control returns to the GO optimization driver and processing continues until a user-specified condition is reached.

Figure 2.4 shows the process graphically.

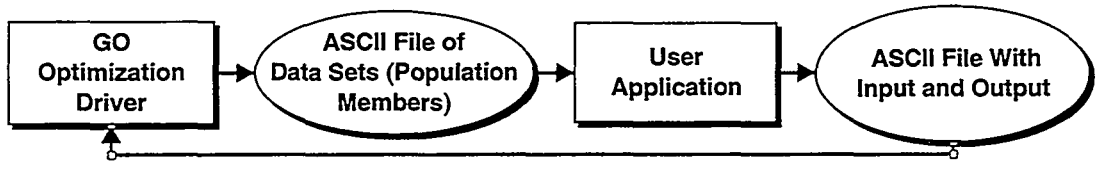


Figure 2.4. GO Structure for Optimization Analysis

Because optimization requires the type of process loop shown in Figure 2.4, GO must be able to exercise at least minimal control over your application. The required control can be exercised by having GO start your application. This approach requires that your application be capable of accepting command line arguments so that GO can specify the file name containing input data for the application. Once your application is launched to process a population of solution strings, the optimization driver will wait in an inactive state until a file of results is available for use in creating the next generation of input data sets. GO will determine when the results file is available by using standard Windows API calls. Processing can be terminated directly by the user or when a specified number of generations is complete.

3.0 GO Optimization Input

The link between GO and your application is illustrated in Figure 2.4. This type of link requires that GO take user input to set up the optimization analysis, launch your application, wait for the application to complete its analysis, create the next generation of optimization input, etc. Thus, GO requires both the input needed to structure the optimization analysis as well as information required to interpret output from the application. The next five sections focus on these two areas of user input to GO.

3.1 Combinations Input

GO uses a genetic algorithm (GA) as its optimization mechanism. The first category of input defines genes and chromosomes. To edit this information, select **Combinations** from the **Edit** menu. The form is shown in Figure 3.1.

For each gene the required input is:

- Gene Name: A name to be associated with a particular gene. For example, applying genetic optimization to reliability problems, a gene name could be "Hard Drive Upgrade" or "Spare Pump Motor(s)." If genetic optimization is used to find a "best" or "worst" point in electrical simulation input space, a gene might be "Resistance R1."
- Number of Levels: The number of different values or "flavors" the gene can take on. In reliability optimization, for example, if a gene represents a component upgrade, there might be 2 levels where level 1 represents no change and level 2 represents the upgrade. If the gene represents a particular input variable and the desire is to find some optimal point in input space, the gene might have say 100 levels where each level will be interpreted by the user's application to represent a discrete point on the input range for that variable.

The **Import** and **Export** buttons on the form in Figure 3.1 form allow you to import data into the form or export data from the form. Both comma-delimited (Excel) and tab-delimited text files are supported. The **Col. Width** button adjusts the width of the data columns to use the entire grid area or to the minimum width necessary to display column results.

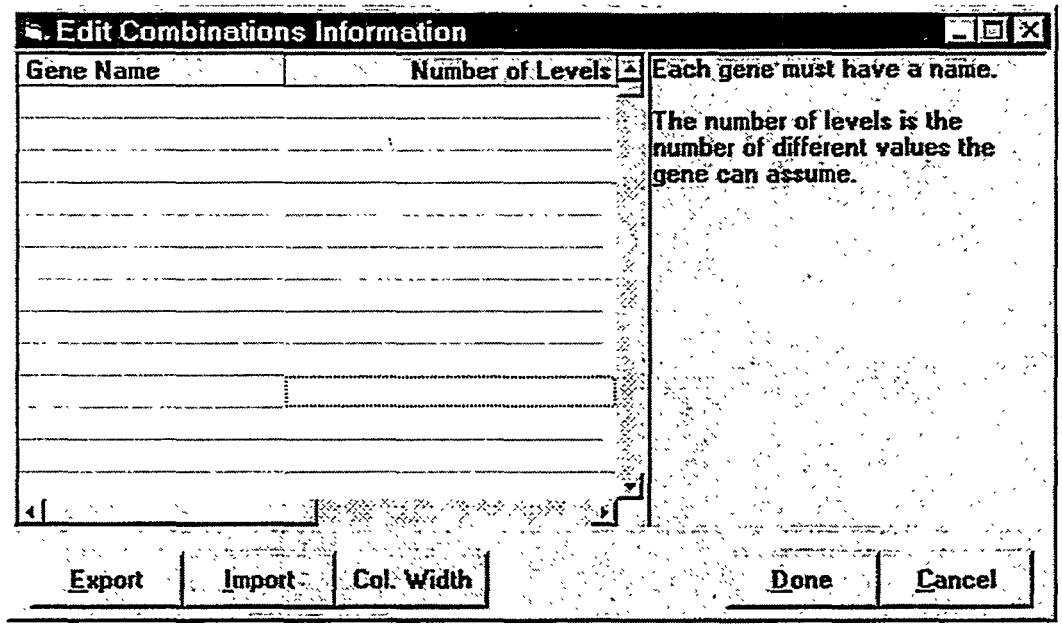


Figure 3.1 Combinations Input to Define Genes

3.2 Problem Setup

This section discusses the parameters that are used to control the operation of the genetic algorithm. To edit this information, select **Problem Setup** from the **Edit** menu. The form for genetic algorithm parameters is shown in Figure 3.2.

You can select one of two analysis types: optimization or enumeration. Optimization analyses apply the genetic algorithm search for a "best" combination from a large space of possible combinations. Enumeration is used to understand the characteristics of the space of possible combinations. Optimization parameters will be discussed first followed by enumeration parameters.

3.2.1 Genetic Optimization Parameters

Clicking the mouse anywhere within the Optimization box on the General tab (Figure 3.2) sets the analysis type to Optimization and accesses optimization parameters. Conversely, clicking the mouse anywhere within the Enumeration box on the General tab sets the analysis type to Enumeration and accesses enumeration parameters. Enumeration means that a fixed number of randomly selected input combinations will be generated.

3.2.1.1 General Tab (Figure 3.2)

- Population Size: The number of chromosomes or population members to be analyzed in each generation. The population size might range from 40 or 50 to 200 or more depending on the number of possible combinations. The maximum population size is 1,000.

- Number of Generations: The number of generations or cycles for the optimization to run.
- LHS or Random Initialization: Determines the method used to initialize the population for the first generation. LHS initialization is a stratified scheme based on Latin Hypercube Sampling. So long as the population size is large compared to the number of gene levels for any gene, this scheme ensures a uniform (but random) scattering of the initial population over the input space. Random initialization simply makes the first population by selecting gene levels randomly. If the number of levels for any gene is comparable to the population size (no. levels >= 0.5 x population size), you should use random initialization.
- Zero Gene Level: Checking this option causes a zero gene level to be included along with the specified number of non-zero levels. Suppose a gene was given 3 levels in the combinations input form (Figure 3.1). If the zero gene level option is not checked, the possible levels for that gene are 1, 2, and 3. If this option is checked, the possible gene levels are 0, 1, 2, and 3. This option can be useful in circumstances such as spares optimization where a gene level of 0 might indicate that none of that particular spare would be included in the inventory.

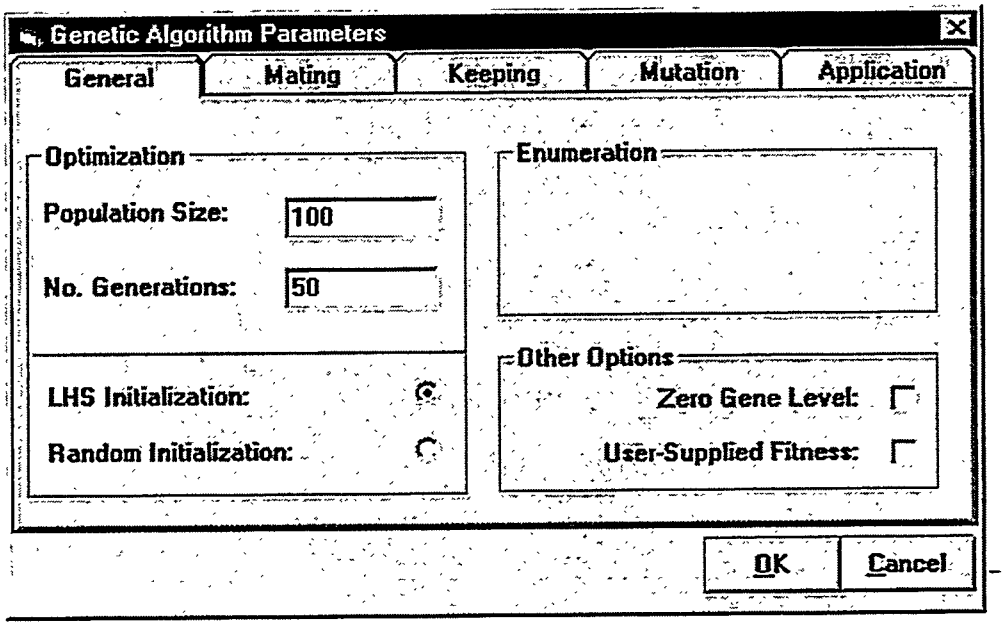


Figure 3.2 General Tab for Genetic Algorithm Parameters

- User-Supplied Fitness: Checking this option means that your application will calculate fitness for each combination. If this option is not checked, the user must provide performance measure and constraint information as described in Sections 3.4 and 3.5.

3.2.1.2 Mating Tab (Figure 3.3)

- Mate Selection Options: Choose tournament or proportional selection.
- Mating Options: Choose segment crossover as illustrated in Figure 3.1 or point crossover which swaps genes between the two parents at randomly selected locations.

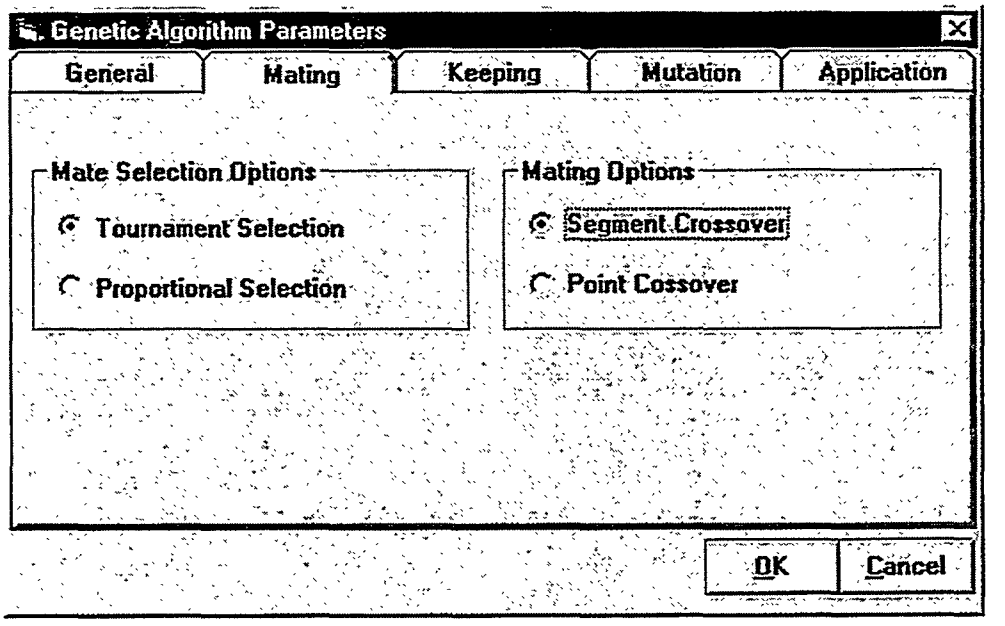


Figure 3.3 Mating Tab for Genetic Algorithm Parameters

3.2.1.3 Keeping Tab (Figure 3.4)

- Keep Selection Options: Population members to be kept for the next generation can be chosen by tournament or proportional selection.
- Start Keep Fraction: The fraction of the population to be kept for the next generation should generally be lower in the early generations than it should be near the end of the optimization analysis. The reason is that early in the analysis, the optimization should aggressively search the space. As the genetic algorithm finds the better areas of the input space, the fraction of the population to move unchanged to the new generation should increase to reduce the risk of losing good solutions by mating and mutation. The start keep fraction is the fraction of the population to be retained for the new generation at the beginning of the optimization. Typical values might be 0.1 to 0.5.
- End Keep Fraction: The fraction of the population to be retained for the new generation at the end of the optimization. Typical values might be 0.2 to 0.7.
- Keep Interpolation: You can choose either linear or exponential interpolation of the keep fraction between the start and end values.

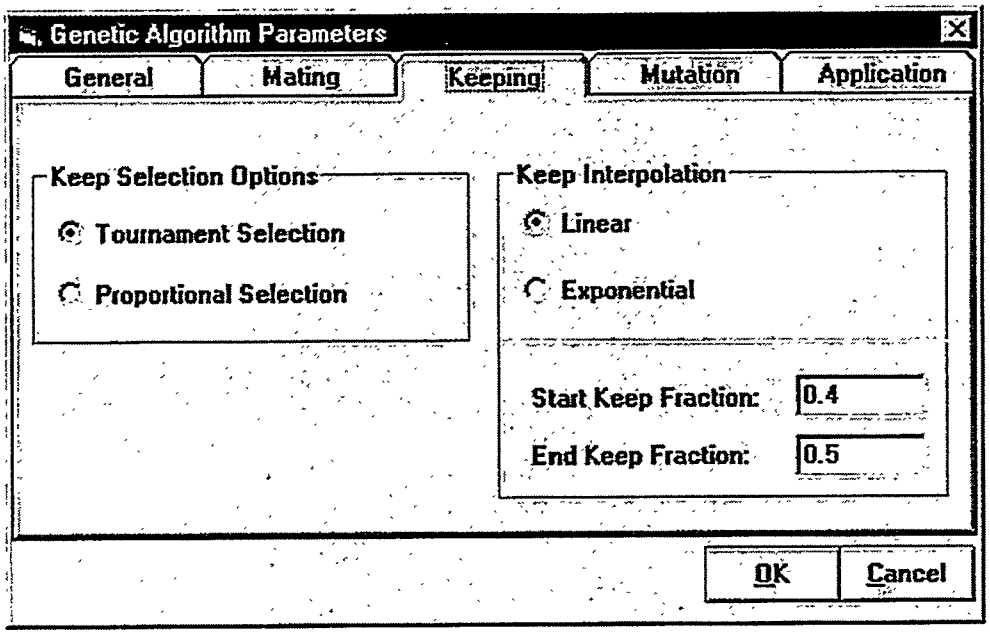


Figure 3.4 Keeping Tab for Genetic Algorithm Parameters

3.2.1.4 Mutation Tab (Figure 3.5)

- Start Mutate Fraction: The fraction of the population to be mutated at the beginning of the optimization analysis. Setting the starting fraction larger than the ending fraction provides more aggressive exploration of the space early in the optimization while reducing the risk of losing good solutions to mutation at the end of the optimization. Typical mutation starting fractions might be 0.1 to 0.3.
- End Mutate Fraction: The fraction of the population to be mutated at the end of the optimization. Typical mutation ending fractions might be 0.05 to 0.15.
- Max. Genes to Mutate: The maximum percentage of genes to be mutated in a selected population member. Once a chromosome or population member is selected for mutation, at least one but not more than this percentage of the genes will be mutated. Typical values for this input might be 5% to 30%.
- Mutation Interpolation: Choose whether the mutation fraction will be interpolated linearly or exponentially.

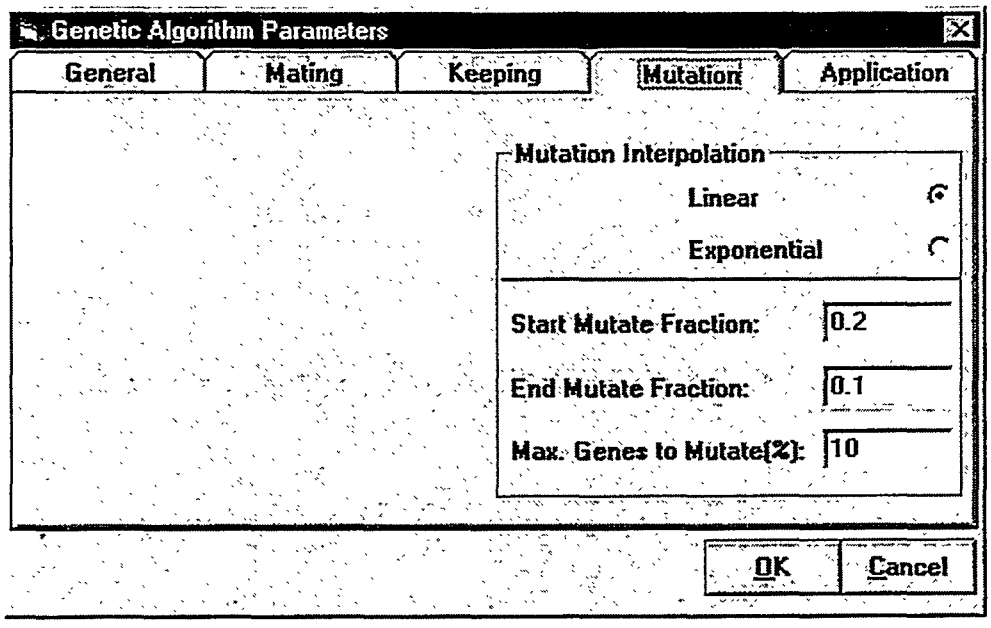


Figure 3.5 Mutation Tab for Genetic Algorithm Parameters

3.2.1.5 Application Tab (Figure 3.6)

- Application: Identify the application that GO will drive for the optimization analysis (see Figure 2.1). You can type in this information directly or click the Browse button to locate the application file.
- Command Line: GO creates a temporary data transfer file that contains population information. This file and path name will appear as a command line parameter when your application is executed by GO (e.g., C:\YourDirectory\YourApp.exe C:\YourDirectory\TempOptData.txt).
- Autosave Results: If you want GO to automatically save intermediate results, check Autosave and specify how often (the number of generations) between saves. If you are running an enumeration problem, GO will automatically save results after every generation.

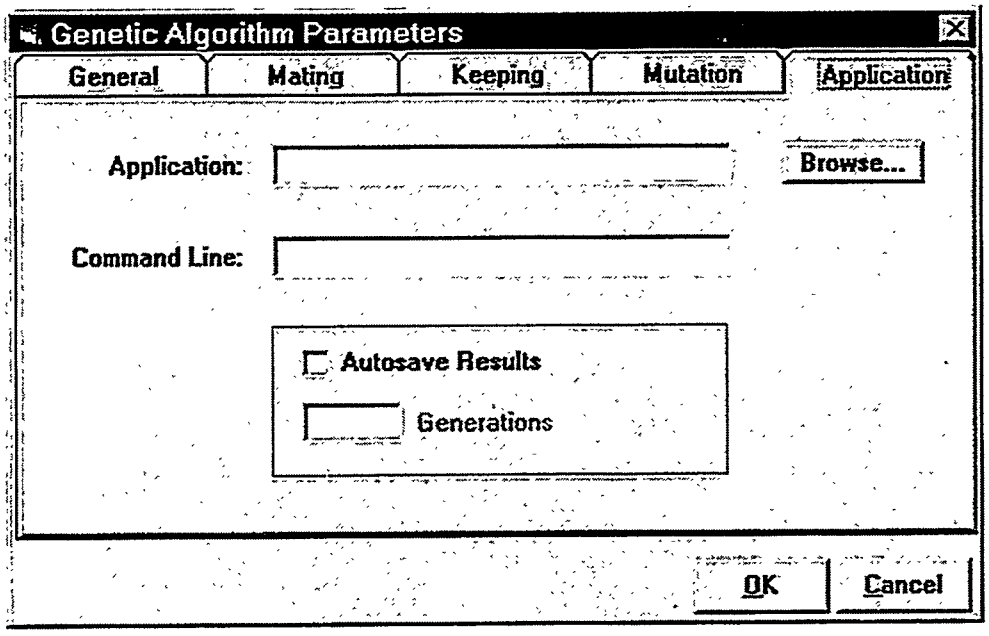


Figure 3.6 Application Tab for Genetic Algorithm Parameters

3.2.2 Enumeration Parameters

Clicking the mouse anywhere within the Enumeration box on the General tab (Figure 3.7) accesses enumeration parameters. Enumeration means that a fixed number of randomly selected input combinations will be generated.

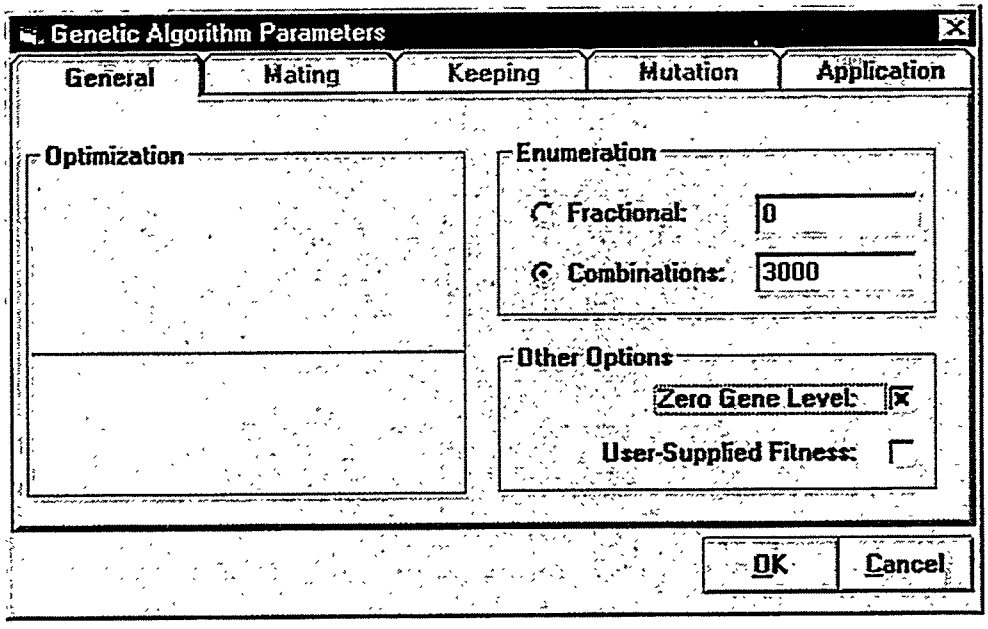


Figure 3.7 General Tab for Enumeration Analysis

3.2.2.1 General Tab (Figure 3.7)

- Enumeration (Fractional): This option specifies that a fraction of the possible combinations will be performed. For example, if you specify fractional

enumeration and provide a fraction of 0.00001, then 0.001% of the possible combinations will be analyzed. Specifying fractional enumeration ensures that no combinations will be repeated. The maximum fraction is 0.00001. Further, the maximum number of combinations that can be analyzed by enumeration is 1,000,000,000.

- Enumeration (Combinations): If the problem is too large for fractional enumeration, you can simply specify the number of combinations to evaluate (maximum number is 1,000,000,000). In this case, there is no guarantee that combinations will not be duplicated. However, the probability of duplicate combinations being analyzed should be very small for large problems.
- Zero Gene Level: Checking this option causes a zero gene level to be included along with the specified number of non-zero. Suppose a given gene was given 3 levels in the combinations input form (Figure 3.1). If the zero gene level option is not checked, the possible levels for that gene are 1, 2, and 3. If this option is checked, the possible gene levels are 0, 1, 2, and 3. This option can be useful in circumstances such as spares optimization where a gene level of 0 might indicate that none of that particular spare would be included in the inventory.

3.2.2.2. Application Tab

The applications tab is described above in section 3.2.1.5.

If you change any of the inputs available through the **Problem Setup** menu item under the **Edit** menu, the analysis will be restarted at the first generation and any previous results will be lost unless saved under a different file name using the **Save As** item under the **File** menu. However, there is a subset of problem setup inputs that can be changed without restarting the problem. These are described in the next section.

3.3 Run Time Parameters

There are some problem-setup parameters than can be changed without restarting the run. These are available from the **Runtime Parameters** menu item under the **Edit** menu. Problem setup inputs that can be changed without restarting the optimization analysis include the number of generations, and all parameters controlling mating, keeping, and mutation. These are described in section 3.2.1 above.

For enumeration, the number of enumeration combinations can be changed without restarting the analysis.

3.4 Performance Measure Inputs

The input form for performance measures is shown in Figure 3.8. See Section 3.2 for a discussion of performance measure parameters. Up to 10 performance measures are permitted.

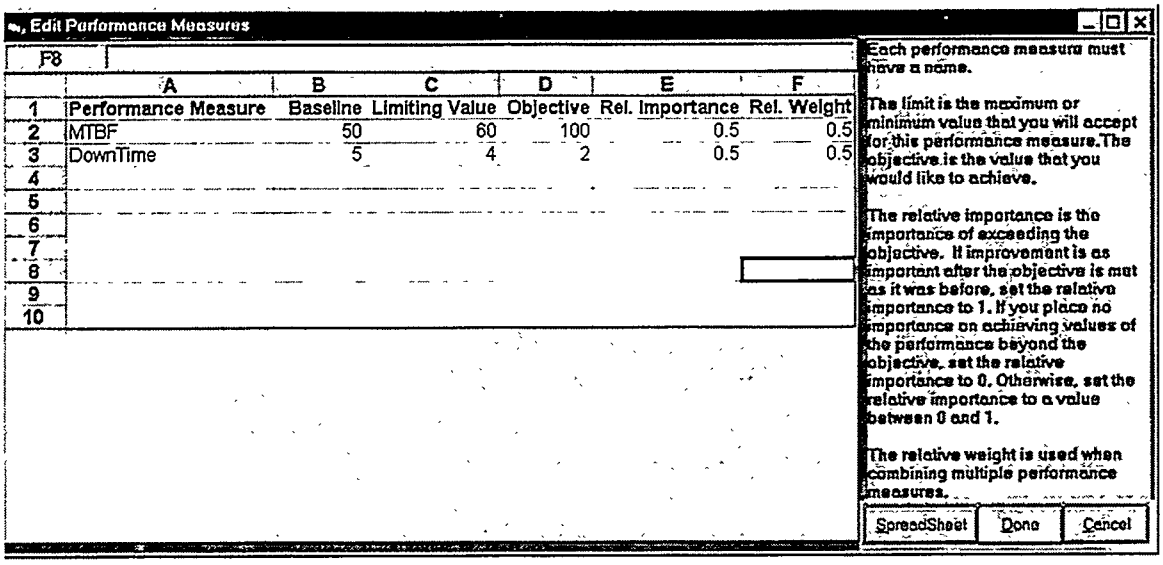


Figure 3.8 Performance Measure Input

- Performance Measure: A string that will be used to identify the performance measure. Each performance measure must have a name.
- Baseline: The initial value of the performance measure. That is, the baseline value is the value the performance measure has prior to the optimization analysis. This value is only used to ensure that performance measure and fitness graphs start at the proper point.
- Limiting Value: The minimum or maximum acceptable value of the performance measure.
- Objective: The desired value of the performance measure.
- Relative Importance: A value between 0 and 1 that indicates the importance of exceeding the objective. A value of 0 places no importance on exceeding the objective while a value of 1 places as much importance on exceeding the objective as on meeting it.
- Relative Weight: This input is only needed if more than one performance measure will be evaluated. When multiple performance measures are used, the relative weight is used to combine performance measures in the fitness calculation (see section 3.2). Relative weights are normalized by GO to sum to 1.

3.5 Constraint Inputs

The input form for performance measures is shown in Figure 3.9. See Section 3.2 for a discussion of constraint parameters. Up to 10 constraints are permitted.

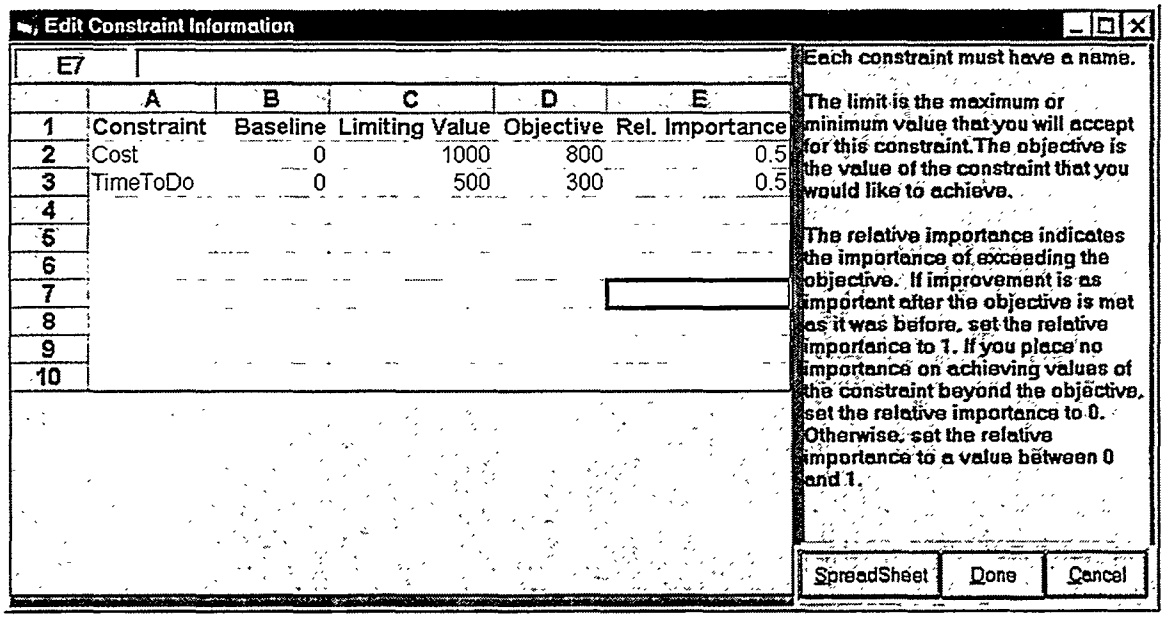


Figure 3.9 Constraint Input

- Constraint: A string that will be used to identify the constraint. Each constraint must have a name.
- Baseline: The initial value of the constraint. That is, the baseline value is the value the constraint has prior to the optimization analysis. This value is only used to ensure that fitness and constraint graphs start at the proper point.
- Limiting Value: The minimum or maximum acceptable value of the constraint.
- Objective: The desired value of the constraint.
- Relative Importance: A value between 0 and 1 that indicates the importance of exceeding the objective. A value of 0 places no importance on exceeding the objective while a value of 1 places as much importance on exceeding the objective as on meeting it.

3.6 Saving Your Data

GO stores your combinations input and problem setup data in a file with the extension *tgo*. If the problem has been analyzed, the file also contains results. To create a new file, select **New** from the **File** menu and supply a path and file name. To open an existing file, select **Open** from the **File** menu.

To save changes to an existing file, use the **Save** command on the **File** menu. If you want to base a new file on an existing file, open the existing file and the use the **Save As** command on the **File** menu to create a new file.

3.7 The GO Toolbar

Figure 3.10 shows the GO toolbar. The toolbar can be displayed in a large or small format by selecting the appropriate command from the **View** menu.

Figure 3.10 The GO Toolbar

Working from right to left along the toolbar in Figure 3.10, the buttons perform the following tasks:

New	creates a new GO file. This command is not available when a file is already open. Performs the same function as the **New** command on the **File** Menu.
Open	opens an existing GO file. This command is not available when a file is already open. Performs the same function as the **Open** command on the **File** menu.
Save	saves the current file to disk. Performs the same function as the **Save** command on the **File** menu.
Close	closes the current file. Performs the same function as the **Close** command on the **File** menu.
Pause	interrupts the current analysis. The analysis can be restarted be clicking on the Run command button. This command is useful if you want to choose new result displays, or if you need to save an analysis and restart it at a later time.
Run	begins an analysis or continues a partially complete analysis from the point at which it was paused. Performs the same function as the **Run** command on the **File** menu.
Toolbar	adds a graphics toolbar to the active form.
Copy	copies the active results from to the Windows clipboard. If the active form contains a graph, the graph will be copied. If the active form contains text data, the text will be copied.
Print	prints the active results from. If the active form contains a graph, the graph will be printed. If the active form contains text data, the text will be printed.
Minimize	minimizes the active result form.
Maximize	maximizes the active result form.
Hide	makes the active result form invisible. To redisplay the form, choose the appropriate command from the **View** menu.

This page intentionally left blank

4.0 Linking GO with Your Application

The link between GO and your application is a text file that passes information. After GO creates a generation of combinations (population members) to be evaluated, it runs your application. For example, suppose your application is called YourApp.exe and resides in directory C:\YourDirectory. Further, assume that you have specified a command line (data transfer file name) of TempOptData.txt. Then, once GO has created a generation of combinations to be evaluated, it will start your application as follows:

C:\YourDirectory\YourApp.exe TempOptData.txt

When your application starts, it must open the file named on the command line and evaluate each of the combinations. The contents of the data transfer file as created by GO are illustrated in Figure 4.1.

```
25,4,1,1
"Gene1", "Gene2", "Gene3", "Gene4", "Fitness", "YourPerfMeas", "YourConstraint"
3, 0, 3, 2, 5, 0, 0, 0
4, 2, 1, 5, 2, 0, 0, 0
7, 1, 4, 3, 0, 0, 0, 0
10, 4, 2, 1, 3, 0, 0, 0
11, 3, 0, 0, 4, 0, 0, 0
14, 1, 2, 1, 2, 0, 0, 0
17, 3, 0, 2, 4, 0, 0, 0
...
```

Figure 4.1 Example Data Transfer File

First Line

- Number of combinations or population members to be evaluated (25 in the above example)
- The number of genes (4 in the above example).
- The number of performance measures (1 in the above example)
- The number of constraints (1 in the above example).

Second Line

- Gene Names (1 to number of genes). In Figure 4.1 the gene names are; Gene1, Gene2, Gene3, and Gene4.
- Fitness (the word 'Fitness' will always follow the list of gene names).
- Performance measure names (1 to number of performance measures). In Figure 4.1, there is one performance measure called 'YourPerfMeas'.
- Constraint names (1 to number of constraints). In Figure 4.1, there is one constraint named 'YourConstraint'.

Remaining Lines

The remaining lines are the population members or combinations developed by GO to be evaluated by your application. The third and remaining lines each contain the following information.

- Population member index. Recall that after the first generation, GO will keep unchanged a fraction of the previous population. Members of the new population that have not been changed do not need to be reevaluated and are not included in the data transfer file. Only new population members are written to the data transfer file to be evaluated by your application. The population member index ensures that results returned from your application are properly slotted back into the population.
- Gene levels (1 to number of genes). In Figure 4.1, there are 4 gene level values on each population member line.
- Fitness. GO writes the fitness value after the list of gene levels for each population member. As written by GO, the fitness value is set to 0 because the population member is new and its fitness is not known. If you have checked the User-Supplied Fitness option (Figure 3.2), your application is expected to calculate and return a fitness value for each population member. Otherwise, fitness will be calculated by GO when population results are returned.
- Performance measures (1 to number of performance measures). These values must be returned by your application.
- Constraints (1 to number of constraints). These values must be returned by your application.

After each generation has been created by GO, the data transfer file is written and GO starts your application. When your application begins execution, it must open the data transfer file named in the command line. Your application must translate each population member into appropriate data input values for evaluation. In evaluating each population member, your application must calculate the specified performance measures and constraints or calculate fitness if you have chosen the User-Supplied Fitness option.

When your application has evaluated all the population members in the data transfer file, you must rewrite the file using the same file name and path. The data transfer file, as rewritten by your application, must provide performance measure and constraint values (or optionally fitness) for each population member.

The data transfer file as returned by your application might look like Figure 4.2.

```
25,4,1,1
"Gene1", "Gene2", "Gene3", "Gene4", "Fitness", "YourPerfMeas", "YourConstraint"
3, 0, 3, 2, 5, 0, 15.7, 32.1
4, 2, 1, 5, 2, 0, 17.1, 30.4
7, 1, 4, 3, 0, 0, 14.9, 25.9
10, 4, 2, 1, 3, 0, 18.5, 31.3
11, 3, 0, 0, 4, 0, 17.2, 28.5
14, 1, 2, 1, 2, 0, 16.3, 27.8
17, 3, 0, 2, 4, 0, 15.2, 24.9
...
```

Figure 4.2 Example Data Transfer File as Written Your Application

Notice that the only changes made the data transfer file have been to replace the performance measure and constraint values (or fitness value) with results calculated by your application for each population member.

When your application has evaluated all population members and rewritten the data transfer file, it must terminate. Termination of your application is a signal to GO that the file can be read and the next generation created. This process will continue until all generations have been evaluated or you end the analysis.

This page intentionally left blank

5.0 GO Results

This section describes results available from the GO Genetic Optimization Driver. All GO results can be accessed from the **View** menu. Results forms can be activated only while the GO driver has focus and not while your application is running. You can activate any of the results forms before you begin the optimization run or while the analysis is paused.

5.1 Summary Information

The summary information form shows the gene levels, performance measures, constraints, and fitness values for the current population. Figure 5.1 shows the form. Each population member is available with the form defaulting to population member with the best fitness. The Rank drop-down list is used to view other population members.

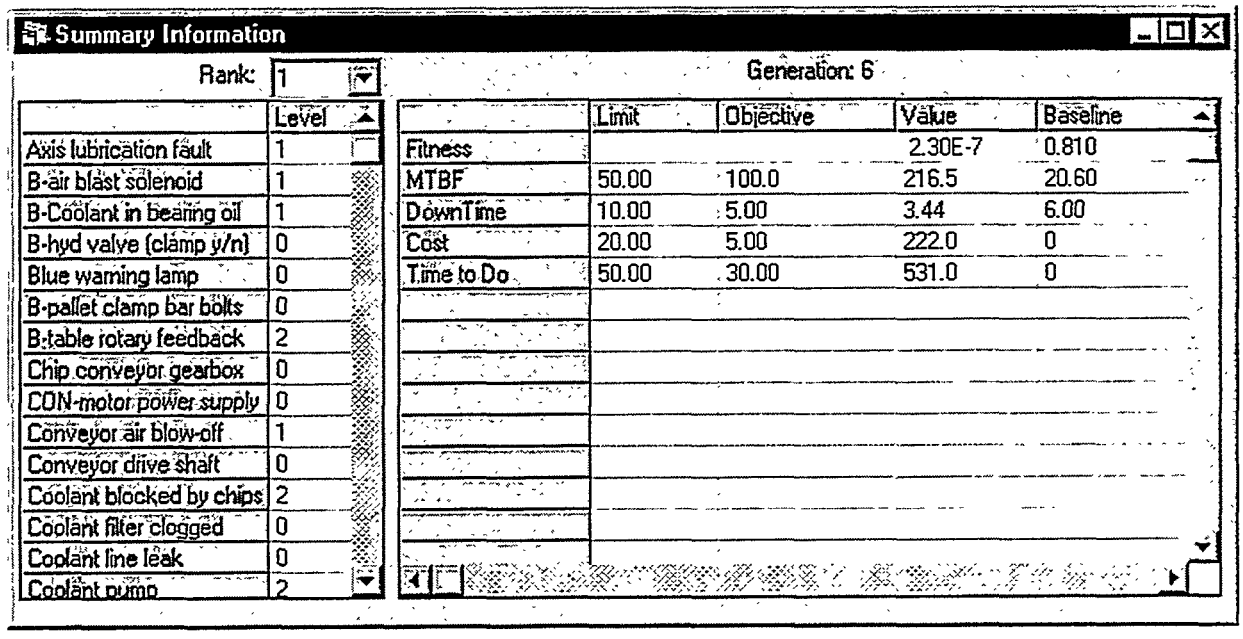


Figure 5.1 Summary Information Form

The left side of the form shows a list of gene names with the gene levels corresponding to the selected population member. The right side of the form shows the fitness followed by all performance measures then all constraints. The 'Value' column shows the fitness, performance measure, or constraint value for the selected population member. Baseline values are shown along with limit and objective values for performance measures and constraints.

5.2 Top Combinations

The top combinations form shows the 10 best population members, in terms of fitness, that have been found so far whether or not they occur in the current generation. This form is shown in Figure 2 and contains similar information to that in the summary information form.

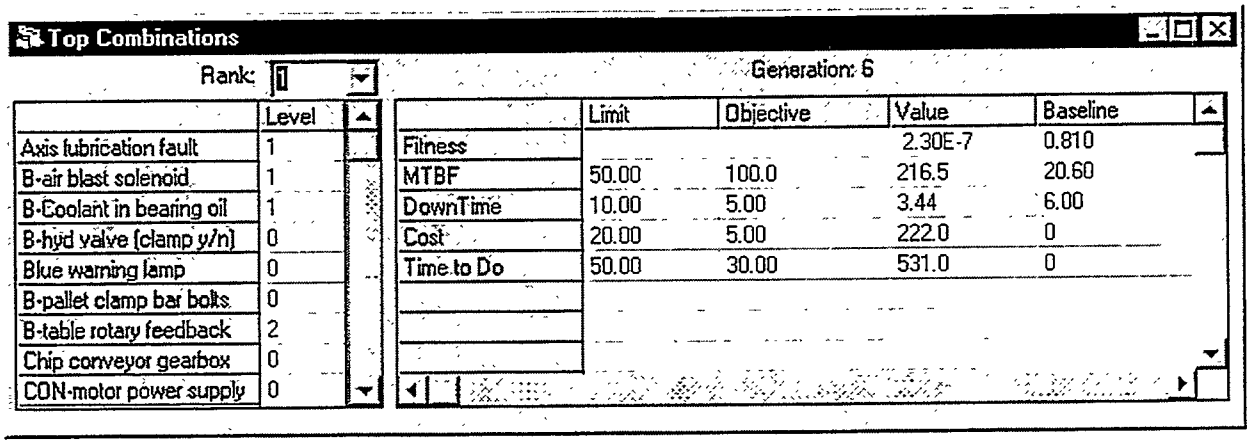


Figure 5.2 Top Combinations Form

5.3 Fitness Graph

To view a graphical display of fitness, select **Fitness** from the **View** menu. If you are running an optimization problem, the graph will show a history of the fitness of the best population member in each generation as shown in Figure 5.3.

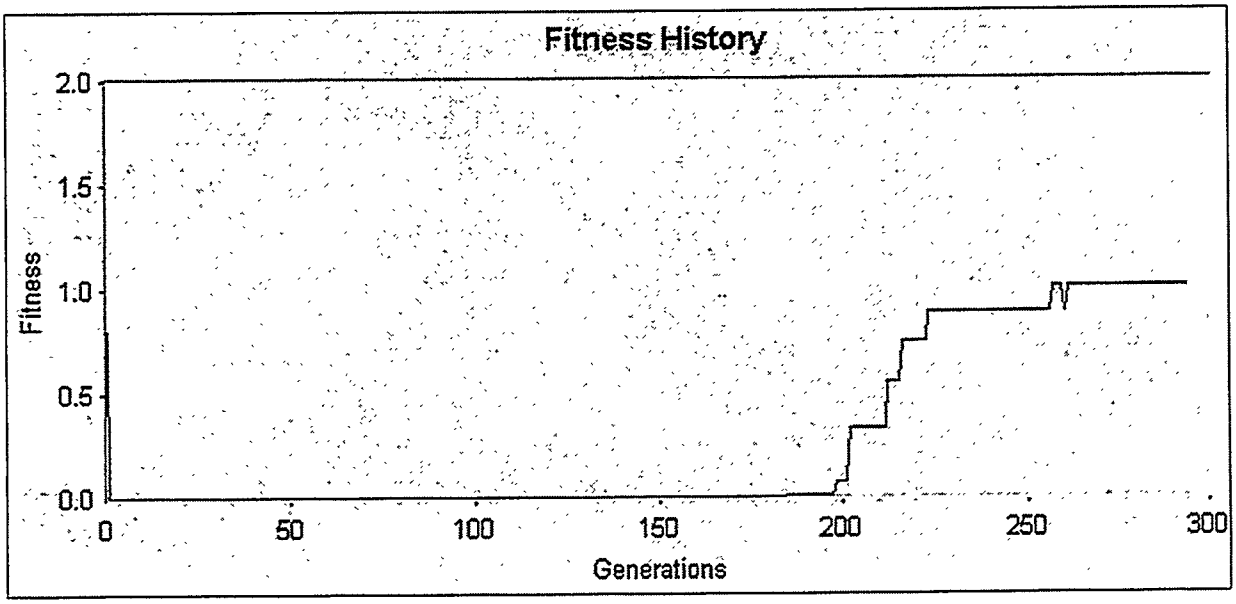


Figure 5.3 Fitness History for an Optimization Run

If you are running an enumeration, the fitness graph is a histogram showing the distribution of fitness values in the combinatorial space (Figure 5.4).

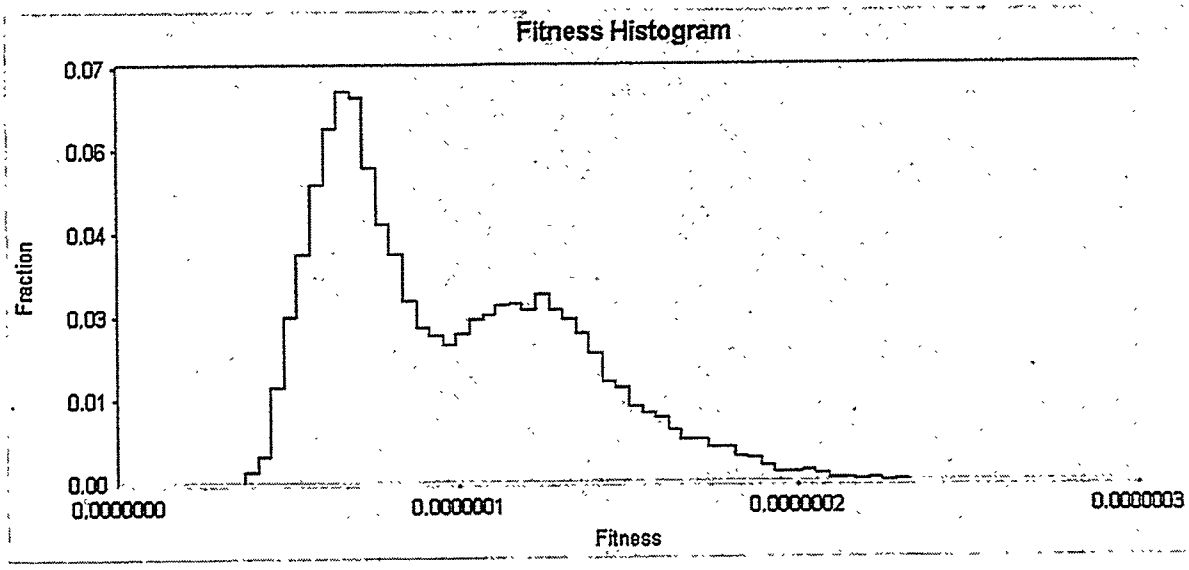


Figure 5.4 Fitness Histogram for an Enumeration Run

Similar results to those shown in Figures 5.3 and 5.4 can be displayed from the **View** menu for all performance measures and constraints.

This page intentionally left blank

6.0 GO Example Application

An example based on repairable systems reliability has been developed to illustrate the use of GO in a practical application. The example is written in Visual Basic 5 and both executable and source code files as well as example problem input are included in the installation files. The example application (called RelOpt) calculates Mean Time Between Failures (MTBF), Mean Time to Repair (MTTR) or down time, and availability for series systems. The required inputs for each failure mode are:

Failure Rate: The failure rate for a particular failure mode is the number of occurrences per hour of operational time. For example, a failure rate of 0.0001 would mean that the particular failure mode occurs, on average, once every 10,000 operational hours.

Down Time: Down time (also called repair time) is the total time in hours required to return he system to operation after a particular failure mode occurs. For example, a down time of 2.5 would mean that, on average, the failure mode takes 2.5 hours to repair and return the equipment to operation.

Since RelOpt was developed for use with GO, there must be a way to relate gene levels to changes in input data. This is done by providing failure rate and down time multipliers. For example, if a given gene level corresponds to a 30% improvement in failure rate for a particular component, the failure rate multiplier for that component and gene level would be 0.7. Similarly, if a particular gene implied a 60% reduction in down time for a component, the down time multiplier would be 0.4. Finally, for use with optimization, RelOpt must provide for constraints or penalties associated with each gene level. To meet his need, RelOpt allows the user to specify a cost and time-to-do associated with the component improvement represented by each gene level. Then, either or both of these measures can be used as constraints for the optimization analysis. Failure rates and down times for each failure mode are read by RelOpt from a file names OptSetup.dat. This file also contains costs, time-to-do, failure rate multipliers and down time multipliers for each gene level and each failure mode. The OptSetup.dat file is included in the GO setup.

MTBF, MTTR and availability are calculated from the following equations:

$$\mathrm{MTBF} = \frac{1}{\sum_{j=1,N} \lambda_j}$$

$$\mathrm{MTTR} = \frac{\sum_{j=1,N} \lambda_j \tau_j}{\sum_{j=1,N} \lambda_j}$$

$$\mathrm{Availability} = \frac{\mathrm{MTBF}}{\mathrm{MTBF} + \mathrm{MTTR}}$$

where λ_j is the failure rate for failure mode i and τ_j is the down time.

The reliability allocation optimization is applied to the design of a cluster tool, a highly complex piece of equipment used in semiconductor manufacturing. Piecewise "effort curves" specifying the amount of effort required to achieve a certain level of reliability for each component or subassembly are defined. The genetic algorithm evolves or picks those combinations of "effort" or reliability levels for each component, which optimize the objective of maximizing reliability while staying within a budget.

Reliability allocation, defined as specifying a level of reliability for each subsystem or module in a system to achieve a given system reliability, should be performed early in the design cycle to guide designers in choosing components, materials, and a design architecture that will meet system objectives. Reliability allocation should start from a base of past experience. Reliability allocation is not the same as detailed design tradeoff studies, where specific design options are evaluated for cost and reliability tradeoffs. Nor is reliability allocation the same as reliability improvement, where the objective is to find the optimal combination of improvements to upgrade an existing design to maximize its reliability. However, reliability allocation, design tradeoff studies, and reliability improvement are all related, subsequent and often iterative steps of the design process. Reliability allocation should be the first step since it can guide later design work: it is not efficient to develop a detailed design and then have to redesign and reallocate reliability if the initial allocation is not achievable.

6.1 Application

The reliability allocation optimization will be applied to a problem of cluster tool design. The cluster tool consists of a robot arm surrounded by various processing chambers in which vapor deposition, etching, cleaning, etc. is performed. A laser alignment system combined with the robot arm move the wafers from one chamber to another, according to the manufacturing "recipe" needed. A schematic of the system is shown in Figure 6.1. Since the cost of

equipment, labor, and materials is high for the cluster tool machine, downtime is very expensive and so correct reliability allocation in design is critical.

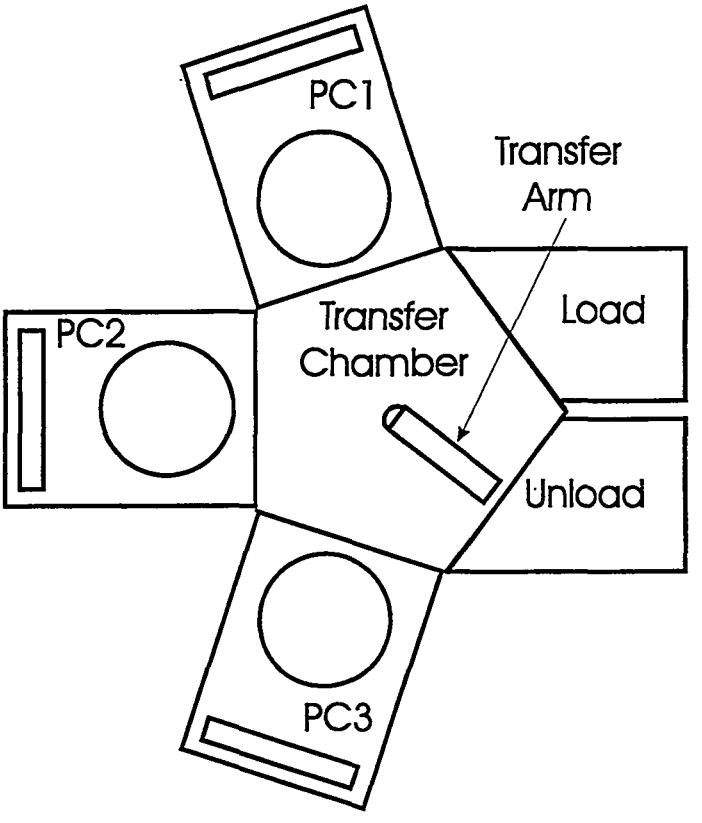


Figure 6.1 Cluster Tool

The reliability of the system is modeled through a fault tree. The top event is defined as the failure of the machine to complete one cycle (i.e., taking one set of wafers through its recipe). This is a task oriented approach to fault trees which is appropriate to equipment failures involved in processing items. Processing involves a series of tasks, so failure to complete any one of those tasks constitutes failure to complete the entire task. Thus, the failure of the cluster tool to complete a cycle can be decomposed into the failure of the machine to load the wafers, a failure to transfer the wafers between stations, a failure to process the wafers in any of the process chambers, or a failure to unload the wafers. These tasks can be further decomposed to their immediate causes. For example, the failure to process the wafers in a chamber could be caused by a failure of the chamber itself or of the central support systems such as the power and the system controller or of the support systems to the chamber. The support systems include the vacuum system for each chamber, the temperature control unit, the gas distribution system, and the RF plasma. These could be further decomposed given more information about their function, however they are not for the purpose of this analysis. See Figure 6.2 for a fault tree of the process. Note that there are events which occur at more than one point throughout the tree. For example, a central support system failure will

affect the functioning of the process chambers and the loading/unloading. This does not imply that failures of these modules are double-counted. Such double counting is eliminated when the fault tree is solved.

The cluster tool is a series system meaning that the system fails if any one of its parts fail. For this reason, it can be analyzed using the simple series reliability application presented here.

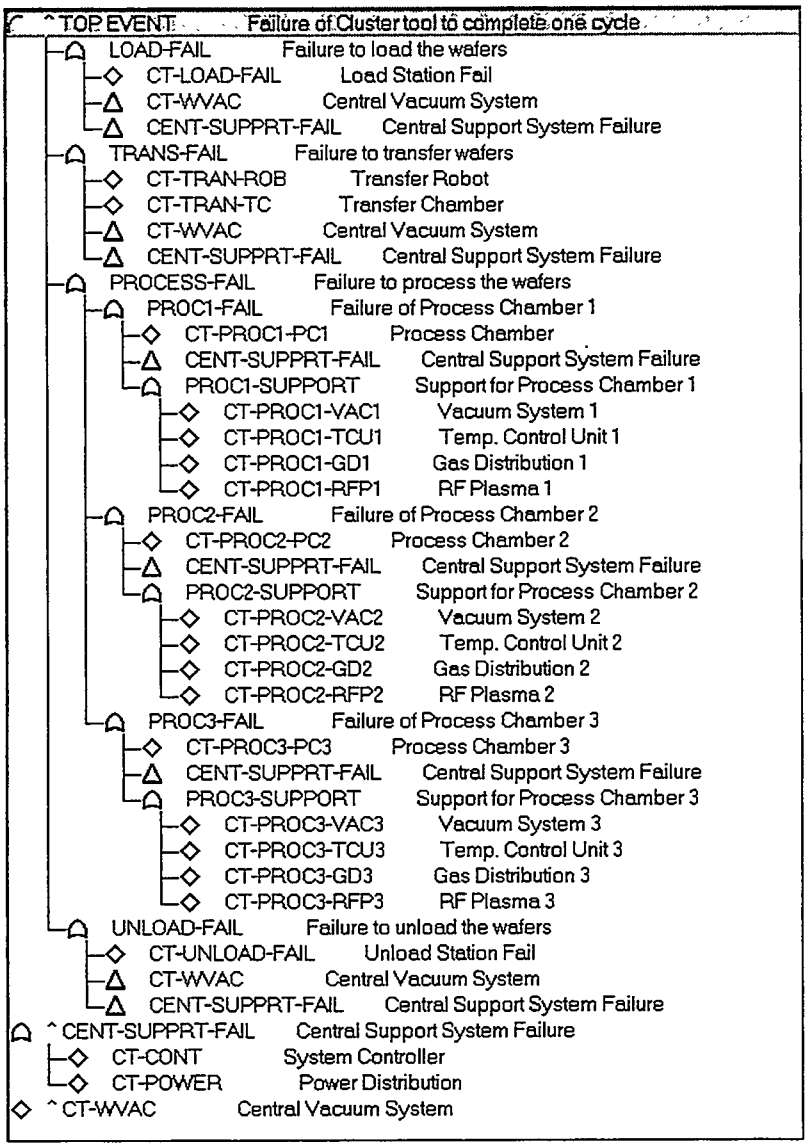


Figure 6.2 Fault tree for Cluster Tool Problem

Failure rate and down time data for the system components are given in Table 6.1 below. Note that identical components (e.g., the three gas distribution components) have been combined on a single line since they are assumed identical and any upgrade to one of them translates into an upgrade for all three. Further, combined identical components are given the summed failure rate of the individual units. These identical components are not redundant since they perform different, necessary functions. The failure data in Table 6.1 give an

MTBF of about 101 hours and an average down time of about 9.65 hours. This represents the estimated reliability before the new design objective is established and is the starting point for improving the system.

Name	Failure Rate (1/Hours)	Down Time (Hours)
System Controller	0.00171	1
Load Station Fail	0.00043	6
Power Distribution	0.00029	8
Gas Distribution (3 Units)	0.00087	14
Process Chamber (3 Units)	0.00129	11
RF Plasma (3 Units)	0.00087	6
Temp. Control Unit (3 Units)	0.00087	13
Vacuum System (3 Units)	0.00129	12
Transfer Robot	0.00057	13
Transfer Chamber	0.00086	16
Unload Station Fail	0.00043	6
Central Vacuum System	0.00043	16
Total	**0.00991**	

Table 6.1 Failure Data for Cluster Tool

6.2 Problem Formulation

Four items must be specified to define the reliability allocation optimization problem:

- Decision Variables
- Performance Measures
- Constraints
- Genetic Algorithm Parameters

These are discussed in more detail below.

6.2.1 Decision Variables

The decision variables in reliability allocation are levels of reliability to assign to each component or subassembly. In formulating the reliability allocation problem, an initial design with preliminary failure rates per component is conceptualized. Then, for each component, potential improvements and the associated effort to make those improvements are postulated. This is done by examining multipliers of the failure rate or failure probability. For example, if the "base case" failure rate on the load station is estimated to have a mean of .00043 failures per hour, the following table is constructed:

Failure Rate Multiplier	Cost (amount of effort) $	Failure rate (failures/hour)
1.0	0	0.00043
0.9	100	0.00039
0.8	200	0.00034
0.7	300	0.00030
0.6	500	0.00026
0.5	700	0.00022
0.4	1000	0.00017
0.3	2000	0.00013
0.2	3000	0.00009
0.1	5000	0.00004

Table 6.2 Reliability Improvement and Corresponding Cost

Table 1 defines a piecewise "effort curve" shown in Figure 6.3. This graph shows the effort to improve the reliability of a particular failure mode increases as the failure rate decreases (as the reliability increases): it usually costs much more to halve or tenth the failure rate than to make incremental improvements.

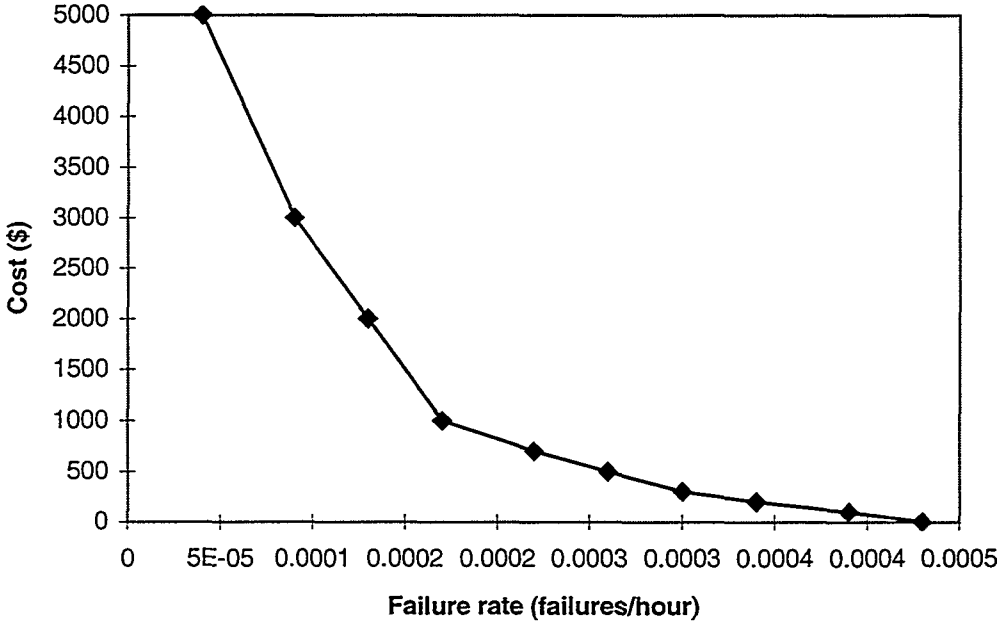


Figure 6.3 Effort Curve for reliability improvement

Effort curves are defined for all components in the initial design. The optimization will pick those combinations of "effort" or reliability levels for each component, which optimize the objective.

Since GO only provides population member definitions in terms of gene levels in its data transfer file, it is up to the user's application to translate gene levels into input data changes. In this example, that means that RelOpt must interpret gene

levels into failure rate and down time multipliers as well as costs and times-to-do. Failure rates, down times, gene-level multipliers, costs, and times-to-do for this example are read by RelOpt from a file called OptSetup.dat which is included in the GO installation package.

6.2.2 Performance Measures and Constraints

Typical reliability optimization problems might seek to maximize some combination of MTBF, availability, average down time, and maintenance costs. Because this example addresses reliability allocation, we will just set an MTBF objective for the system and seek to minimize the cost or effort to achieve that objective.

In addition to the performance measures, it is necessary to define the constraints upon the system. For the purposes of reliability allocation, we only consider the constraint of cost. Usually reliability can be improved substantially, but it may be very costly and not feasible in a world of fixed and shrinking budgets. Thus, budget is the key constraint.

6.2.3 Genetic Algorithm Parameters

The performance of genetic algorithms is dependent on many of the implementation factors and can be tuned to the particular problem at hand. See References 8 and 9 for a detailed discussion of control parameters. We have found genetic algorithms to be fairly robust to the particular choice of control parameters. For this example, a population of size 100 was run for 100 generations. The fraction of the population kept unchanged for the next generation was started at 0.4 and increased linearly to 0.5 by the last generation. The initial mutation fraction was set to 0.3 and reduced linearly to 0.2 by the last generation.

6.3 Results

The cluster tool reliability allocation problem has twelve major subassemblies or components, each with 10 possible levels of reliability (the base case plus nine additional levels from 0.9 to 0.1 times the base case). Thus the total number of combinations is 10^{12} or 1 trillion. This is a relatively large optimization and obviously there are not 1 trillion "realistic" combinations of reliability levels for the design of this equipment. However, the point of the reliability allocation optimization is to provide guidance for what the approximate reliability levels for each subassembly should be, and there certainly could be 1 trillion combinations of potential allocations.

The top combination as well as fitness, MTBF, and cost histories are shown in Figure 6.4. The history plots are typical of optimization problems. Fitness rises rapidly the first few generations then ultimately flattens out as the genetic algorithm as found its "best" combination. The cost history begins at 0 since there is no cost involved until improvements are considered. In the first generation, the cost goes up as combinations are found that increase MTBF.

The optimization then tries to find combinations that reduce cost while increasing MTBF.

The failure rate multipliers were entered in the order of decreasing cost and the zero gene level option was used. These choices result in the selected gene level (times 10) being the recommended percentage improvement in failure rate. The best combination results in an MTBF of 249.5 hours compared to the objective of 250. Note that the cost of 13,500 came in below our objective of 15,000.

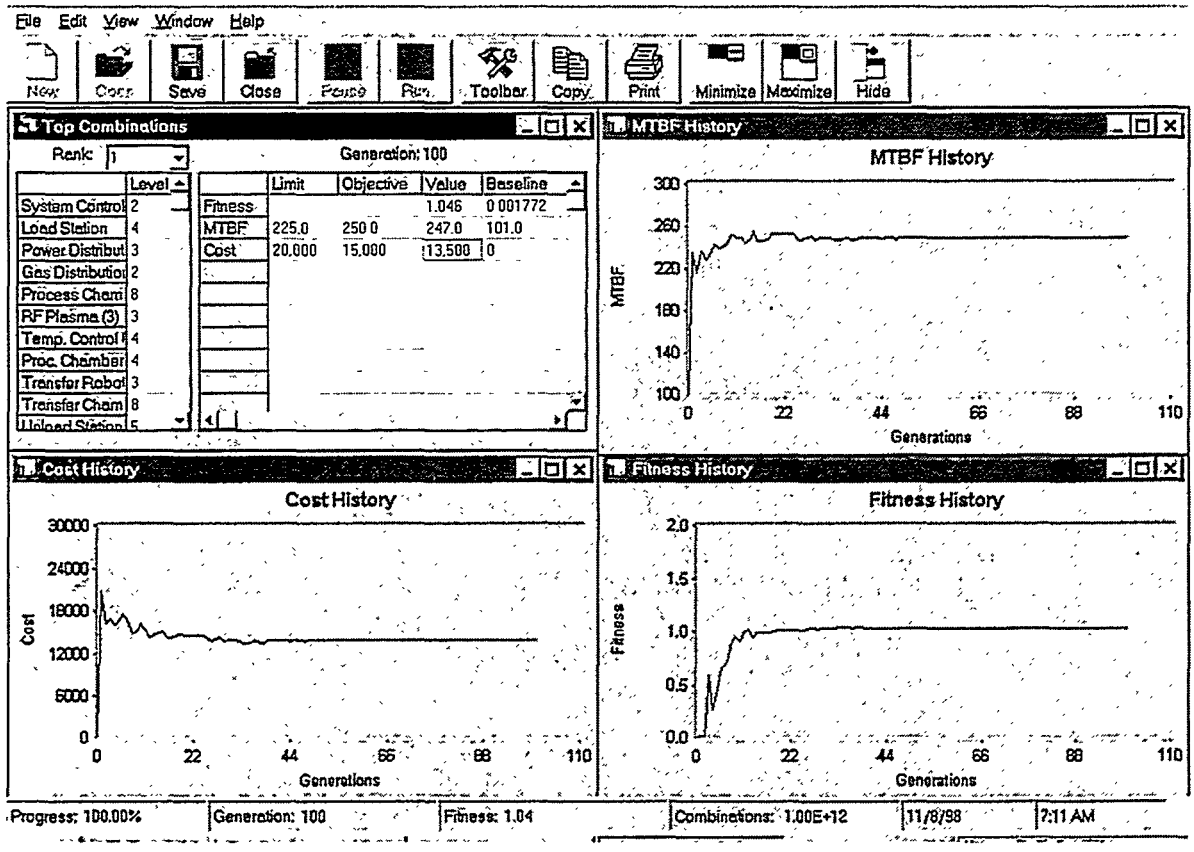


Figure 6.4 Optimization Results for the Cluster Tool Reliability Allocation

GO tracks the top ten solutions over all generations. The top three solutions are shown in Table 6.2. Notice that they are very similar. This indicates that the solution is robust. For example, all solutions have the failure rate of the transfer robot allocated at 30% of its base case, the failure rate of the central vacuum system at 30% of its base case, the failure rate of the system controller at 20% of its base case, etc. There are differences with respect to the reliability allocation for the power distribution and the unload station. The fitness values for the top three solutions range from 1.046 to 1.040. These results can provide guidance for the target levels of reliability that each subassembly should attain in a more detailed design. If you run this application, you might not get exactly the same results because of the randomness inherent in the optimization algorithm. You should get results that are very similar to those shown here.

Component	Solution 1	Solution 2	Solution 3
System Controller	0.2	0.2	0.2
Load Station	0.4	0.4	0.4
Power Distribution	0.3	0.3	0.6
Gas Distribution (3)	0.2	0.1	0.2
Process Chamber (3)	0.8	0.8	0.8
RF Plasma (3)	0.3	0.3	0.3
Temp. Control Unit (3)	04	0.4	0.4
Proc. Chamber Vacuum (3)	0.4	0.4	0.4
Transfer Robot	0.3	0.3	0.3
Transfer Chamber	0.8	0.8	0.8
Unload Station	0.5	0.6	0.4
Central Vacuum	0.3	0.3	0.3
RESULTS			
Fitness	1.046	1.040	1.040
System MTBF (hours)	247.0	249.7	249.6
Improvement Cost ($)	13,500	14,000	14,000

Table 6.3 Top solutions in population

An enumeration of 100,000 randomly selected combinations was run to better understand the solution space for this problem. Histograms of results are shown in Figure 6.5. The most interesting result from the enumeration is that most of the fitness values are very near 0. In fact, in 100,000 randomly selected combinations, only 8 had fitness greater than 0.00001. The best combination found by the enumeration had fitness of 0.846 compared to the optimization result of 1.046.

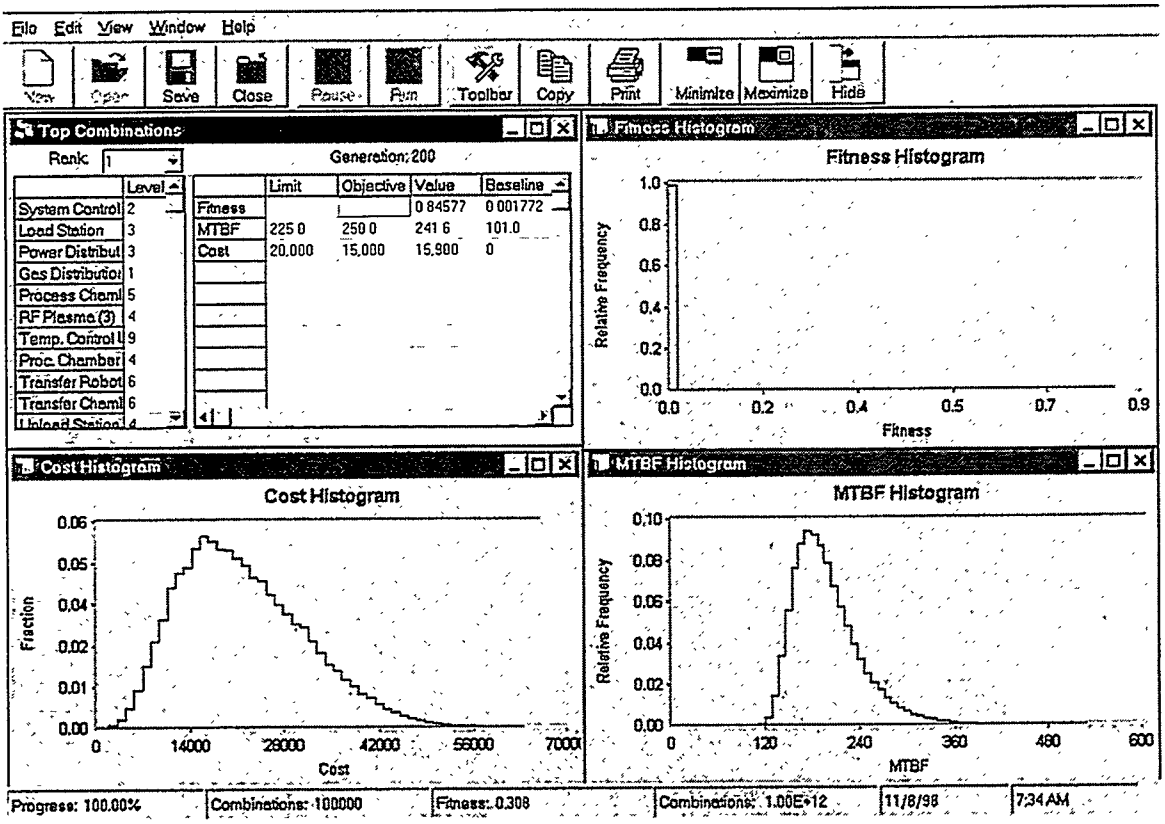


Figure 6.5. Results of Enumeration Analysis

7.0 References

[1] Holland, J. H., 1975. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI. Reprinted in 1992 by MIT Press, Cambridge, MA.

[2] Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.

[3] Powell, D. and M. M. Skolnick, 1993. *Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints*, Proceedings of the Fifth International Conference on Genetic Algorithms, S. Forrest, ed., Morgan Kaufman, Los Alto, CA.

[4] Campbell, J. E., and L. A. Painton, 1996. *Optimization of Reliability Allocation Strategies Through Use of Genetic Algorithms*, American Institute of Aeronautics and Astronautics, AIAA-96-4193-CP.

[5] Painton, L. A., and J. E. Campbell, 1995. *Genetic Algorithms in Optimization of System Reliability*, IEEE Transactions on Reliability, **44**(2), 172-178, June 1995.

This page intentionally left blank

Appendix C: SUNS™/SPICE Design Applet Parse Code

This page intentionally left blank

```
#!/usr/local/bin/perl
#
# ************************************************************************
# Usage: WXXparse -j <jobid> -d <directory> -n <netsuffix> -e <opsys> -h   *
# ************************************************************************
#       Defaults:                                                         *
#    -d <directory> = /home/spice3                                         *
#    -n <netsuffix> = def                                                  *
#    -j <jobid >    = WXX            -h <help> or h or help                *
#    -e <exec >     = unix   (or nt)                                      *
# ************************************************************************
    $jobid        = "test";
    $unixdir      = "/home/spice3";
    $netsuffix    = "def";
    $spice3syslog = "spice3" . "sysnet.log";
    $execos       = "unix";
    $ntdir        = "c:\\spice";
    $homedir      = "";
    $pathdelim    = "/";
  unless (open(stdout, ">$spice3syslog")){
       print stdout (
      "***Error(WXXParse):0002-Unable to open system log: $spice3syslog\n");
       die (
      "***Error(WXXparse):0002-program terminated. Check log file\n");
   }
#
*************************************************************************
#  Check if more than one argument; if so, there needs to be an even number
*
#     meet the format of -argcmd <argvalue>                               *
#
*************************************************************************
   @cmddesg = ""; @cmdargv = ""; $argerr = 0; $help = "";
   $nargs = @ARGV;
   if ($nargs > 1) {
       $r = $nargs % 2;
       if ($r != 0) {
           $argerr = 1;
           print stdout (
           "***Error(WXXParse):0003-Illegal command line\n");
         exit;
        }
    }
#
*************************************************************************
#  Capture the command line aruments and values in array for processing:
*
#
*************************************************************************
    if ($argerr == 0 && $nargs > 0) {
        $i = 0; $j = 0;
        while ($i < $nargs) {
          if (@ARGV[$i] =~ /^-/) {
              @cmddesg[$j] = @ARGV[$i];
                $i++;
              @cmdargv[$j] = @ARGV[$i];
          } elsif (@ARGV[$i] == "help") {
              $help = "-h";
          } else {
```

```
            $argerr = 2;
              print stdout (
               "***Error(WXXParse):0004-Syntax error on command line\n");
              exit;
              $i = $nargs;
               }
          $i++; $j++;
         }
     }
#
*****************************************************************************
#       - look for a -h (help)                                                *
#
*****************************************************************************
    if ($nargs > 0) {
        for ($i = 0; $i < @cmddesg; $i++) {
         if (@cmddesg[$i] eq "-h" || @cmddesg[$i] eq "-help" ||
             @cmddesg[$i] eq "help") {
                 $help = "-h";
                   $i = @cmddesg;
          }
         }
     }
#
*****************************************************************************
#  Search the array and update the program command line argument values:
*
#
*****************************************************************************
    if ($argerr == 0 && $nargs > 0) {
       for ($i = 0; $i < @cmddesg; $i++)    {
         if (@cmddesg [$i] eq "-j") {
               $jobid      = @cmdargv[$i];
           } elsif (@cmddesg[$i] eq "-d") {
              $homedir    = @cmdargv[$i];
           } elsif (@cmddesg[$i] eq "-n") {
             $netsuffix  = @cmdargv[$i];
           } elsif (@cmddesg[$i] eq "-e")  {
             $execos     = @cmdargv[$i];
         } elsif (@cmddesg[$i] eq "-h" || @cmddesg[$i] eq "-help"
               || @cmddesg[$i] eq "help") {
                   $help       = "-h";
         } elsif ($help != "-h") {
                  $argerr = 5;                                          -
                print stdout (
               "***Error(WXXparse):0004-Unknown argument: @cmddesg[$i]\n");
             exit;
           }
        }
    }
#
*****************************************************************************
#  tell user there are errors in the command line arguments / values:
*
#
*****************************************************************************
    if ($argv > 0 || $help eq "-h") {
        print stdout (
            "***Error(WXXparse):0005 Usage: wxxparse -j <jobid> -d <dir>" .
```

```
            " -n <netsuffix> -e <opsys> -h <help> \n");
      exit;
     }
#  set the type of path delimiter for nt vs unix:
    if ($execos eq "nt") {
        $pathdelim = "\\";
     } else {
        $pathdelim = "/";
     }
    if ($homedir eq "" && $execos eq "nt") {
        $dir = $ntdir;
     } elsif
       ($homedir eq "" && $execos eq "unix") {
        $dir = $unixdir;
     } else {
        $dir = $homedir;
          }
#
#
*************************************************************************
# Build the dir / file names for spice output files. Directory structure:
*
#   Directory structure:
*
#        dir = /home/spice3
*
#           jobid:    WXX
*
#             dirs:        images_def  models  out_def plots_def
*
#                files:                  .mdl    .cir    *.plt
*
#                                                .out
*
#                                                .log
*
#
*************************************************************************
    $dir        = $dir    . $pathdelim . $jobid;
    $outdir     = $dir    . $pathdelim . "out_"    . $netsuffix;
    $plotdir    = $dir    . $pathdelim . "plots_" . $netsuffix;
    $spice3net  = $jobid . "spice_" . $netsuffix . ".cir";
    $spice3out  = $jobid . "spice_" . $netsuffix . ".out";
    $spice3log  = $jobid . "spice_" . $netsuffix . ".log";
#
#
*************************************************************************
# Print options to execute the job to the system log:
*
#
*************************************************************************
   print stdout (
     "jobid = $jobid, dir = $dir, exec = $execos, netsuffix = $netsuffix\n" .
     "netlist = $spice3net, spiceout = $spice3out, \n" .
     "spice3log = $spice3log, plotdir = $plotdir, \n" .
     "outdir = $outdir \n");
#
#
*************************************************************************
```

```
# Check to make sure directory and files are found:
*
#
*********************************************************************
   $filerr = chdir ($dir);
   if ($filerr eq false) {
       print stdout (
      "***Error(WXXParse):0006 - Directory ($dir) does not exist. Quit.\n");
       die (
      "***Error:0006 - WXXparse terminated due to errors. Check log
file.\n");
   }
   $filerr = chdir ($plotdir);
   if ($filerr eq false) {
       print stdout (
      "***Error(WXXparse):0007 - Directory ($plotdir) not found. Quit.\n");
       die (
      "***Error:0007-Program terminated due to errors. Check log file.\n");
   }
   $filerr = chdir ($outdir);
   if ($filerr eq false) {
       print stdout (
      "***Error(WXXparse):0008 - Directory ($outdir) not found. Quit.\n");
       die (
      "***Error(WXXparse):0008-Program terminated due to errors." .
      " Check log file.\n");
   }
   if (!-e $spice3net) {
       print stdout (
      "***Error(WXXparse):0009 - File ($spice3net) does not exist. Quit.\n");
       die (
      "***Error(WXXparse):0009 - Program terminated due to errors." .
      " Check log file.\n");
   }
#
  close (spice3syslog);
  unless (open(stdout, ">$spice3log")){
    print stderr (
      "***Error(WXXparse):0010 - Unable to open log message" .
        " file: $spice3log\n");
       die (
      "***Error(WXXparse):0010 - program terminated. Check log file\n");
   }
#
*********************************************************************
# Open SPICE files:INT Comment lines **PRINT Qnn BE CB
*
#
*********************************************************************
   print stdout (
     "jobid = $jobid, dir = $dir, exec = $execos,\n" .
     " netlist = $spice3net, spice3out = $spice3out, spice3log =
$spice3log\n");
#
  unless (open(netlistfile, "$spice3net")) {
       print stdout (
      "***Error(WXXparse):0011 - Unable to open netlist input file:" .
      " $absspice3net\n");
```

```
      die ("***Error(WXXparse):0011 - program termintaed. Check log
file\n");
   }
#
#
*************************************************************************
*
#    Search netlist input for **PRINT Qnn BE V 3_4
*
#     - save the requested plot files in an array to match with
*
#          spice3 output file:                                          *
#     - Model data in netlist must start with $startmodel comment
*
#
*************************************************************************
*
  $i = 0; $j = 0; $endloop = 1; @prntarr ="";
  while ($endloop == 1) {
     $netline = <netlistfile>;
     if ($netline =~ /^\*\*PRINT/) {
        chop ($netline);
        $netline      =~ tr/a-z/A-z/;
        @prntarr      = split(/ +/,$netline);
        @arrcmpnt[$i] = @prntarr[1];
          @arrplotd[$i] = @prntarr[2];
          @arrplotn[$i] = @prntarr[3];
        @arrnodes[$i] = @prntarr[4];
      $i++;
      } elsif ($netline eq "***Start Model Definitions:\n") {
        $endloop = 0;
      } elsif (eof) {
           $endloop = 0;
         print stdout (
           "***Error(WXXparse):0012 -" .
         " Missing ***Start Model Definitions comment" .
         " in spice netlist input: $spice3net. Job terminated.\n");
         die (
         "***Error(WXXparse):0012 - Program terminated. Check log file.\n");
      }
  }
  $plotfiles = $i;
  if (@prntarr == 0) {
       print stdout (
      "***Error(WXXparse):0013 -  No **PRINT statements in spice netlist" .
        " input: $spice3net. Job terminated.\n");
       die (
      "***Error(WXXparse):0013 - Program terminated. Check errlog.\n");
   }
#
*************************************************************************
#   Create the plot file names and set the spice node name to N
*
#     in an array, and then write the array to a file. Write one file for
each*
#     set of plot data:                                                 *
#
*************************************************************************
   for ($i = 0; $i < @arrcmpnt; $i++) {
```

```
        @arrplotnames[$i] = @arrcmpnt[$i] . @arrplotd[$i] .
                            @arrplotn[$i] . @arrnodes[$i];
        @arrplotspice[$i] = "N";
        @arrplotnames[$i] =~ tr/a-z/A-Z/;
    }
#
*************************************************************************
#   Open Spice output file, advance to the plot data, save data as xy values
*
#     in an array, and then write the array to a file. Write one file for
each*
#     set of plot data:                                                  *
#
*************************************************************************
    $endloop  = 1;   $advxy    = 1;  $endfile = 0;  $savexy = 0;
    $chkhdr   = 0;   $wrtplot  = 0;  $newplot = 0;  $getspice = 0;
    $currplot = "";  $nextplot = ""; @xarray  ="";  @yarray ="";
    $ip       = 0;   $plotnbr = 0;   $filenum = 0;
#
#
*************************************************************************
#     Open the Spice3 Output file to process plot data:                  *
#
*************************************************************************
  unless (open(spiceout, "$spice3out")){
        print stdout (
           "***Error(WXXparse):0014 - " .
           "Unable to open spice3.out file: $spice3out\n");
        die (
            "***Error(WXXparse):0014 - Program terminated. Check
errlog.\n");
    }
#
   while ($endloop == 1) {
#
*************************************************************************
# ********  Advance to first plot file:                                  *
#
*************************************************************************
     if ($advxy == 1) {
         $advxy = 0; $stplot = 1;
         while ($stplot == 1) {
           $netline = <spiceout>;
           if ($netline ne "") {
              if ($netline =~ /Index/ && $netline =~ /time/) {
                 print stdout ("*** Advance to plot data: " . "$netline");
                 $stplot = 0; $chkhdr = 1; $currplot = "";
              }
           } else {
             print stdout (
           "***Error(WXXparse):0015 - No plots found in $spice3out file\n");
             $stplot = 0;
             die (
           "***Error(WXXparse):0015 - Program terminated. Check log
file.\n");
           }
         }
      }
#
```

```
#
*****************************************************************************
#  *******  process header in form:  Index   time   v(n) - v(n)
*
#
*****************************************************************************
     if ($chkhdr  == 1) {
         $chkhdr   = 0;
         @plot     = split(/ +/, $netline);
         $nextplot = @plot[2];
         if ($nextplot eq $currplot) {
             $getspiceout = 1;
             print stdout (
            "(WXXparse):0016 - Hdr found: Continue plot " . "$currplot\n");
         } else {
             $newplot  = 1;
             print stdout (
            "(WXXparse):0017 - Hdr found: New plot " . "$nextplot\n");
             if ($currplot ne "") {
                 $wrtplot = 1;
             }
           }
      }
#
*****************************************************************************
# ******* Save xy values from the array for current plot:
*
#
*****************************************************************************
     if ($savexy  == 1) {
         $savexy   = 0;
           @xarray[$ip] = @plot[1];
           @yarray[$ip] = @plot[2];
           $ip++;
       }
#
*****************************************************************************
# ******* Write out xy values from the array for current plot:
*
#
*****************************************************************************
     if ($wrtplot  == 1) {
           $plotfilename = &cplotname($currplot);
         $points = (@xarray);
         $absplotfilename = $plotdir . $pathdelim . $plotfilename;
         print stdout (
            "plot file = $absplotfilename\n");
           open (spiceplot, ">$absplotfilename");
         for ($i = 0; $i < @xarray; $i++) {
                print spiceplot ("@xarray[$i] @yarray[$i]\n");
#           print stdout (
#           "(WXXparse):0017 - x_value = @xarray[$i] y_value = @yarray[$i] " .
#            "*** Save in array for $currplot\n");
           }
          close (spiceplot);
          $filenum  = $filenum + 1;
          $points = (@xarray);
          print stdout (
           "(WXXparse):0018 - Num (x,y) points written to file = $points\n");
```

```
          $wrtplot   = 0;
          @xarray = ""; @yarray = ""; $ip = 0;
          if ($endfile == 1) {
                $endloop = 0;
              print stdout (
               "(WXXparse):0019 - EOD - End of Spice output\n");
           }
      }
#
*************************************************************************
# ******* New Plot File Found - initialize values:
*
#
*************************************************************************
      if ($newplot == 1) {
          $newplot =  0;
          @xarray  = ""; @yarray = ""; $ip = 0;
          $currplot = $nextplot;
          $getspiceout = 1;
          print stdout (
             "(WXXparse):0020 - Initialization for plot: $currplot \n");
      }
#
*************************************************************************
# ******* read a record from SPICE output file:                          *
#          - record can be a plot header record:  Index   time v(1) - V(2)  *
#          - record can be a xy value to be saved in the array             *
#          - record can be heading, separator, information, or null line    *
#
*************************************************************************
      if ($getspiceout == 1) {
          $getspiceout  = 0;
          $netline = <spiceout>;
          chop $netline;
          if ($netline ne "") {
              if ($netline =~ /Index/ && $netline =~ /time/) {
                  $chkhdr = 1;
                  print stdout (
                   "*** Hdr Record Found: " . "$netline\n");
               } else {
                   $getspiceout = 1;
                   @plot = split(/\s+/, $netline);
                   if (@plot == 3 && @plot[0] =~ /^-?\d+$/) {
                       $savexy = 1;
                   } else {
#                      print stdout (
#                  "* $netline\n *** record is not an xy value\n");
                     }
                   }
           } elsif (eof) {
                $wrtplot = 1; $endfile = 1;
           } else {
                $getspiceout = 1;
#                   print stdout (
#                  "* $netline\n   *** record is not an xy value\n");
             }
        }
    }
```

```
#*************************************************************************
*
#       Print out summary to log:
*
#*************************************************************************
*
   print stdout (
      "(WXXparse):0021 - \n\n Summary of job execution results\n");
   $i = @arrplotnames;
   print stdout (
      "(WXXparse):0022 - No. plot files requested on input  = $i:\n");
   for ($i = 0; $i < @arrplotnames; $i++) {
        print stdout (
            "          " . "(plotname = @arrplotnames[$i]," .
                " spice name = $arrplotspice[$i])\n");
    }
   print stdout (
            "    *** No. plot files processed in Spice Output " .
            " = $filenum\n");
   $i = @arrspice;
   print stdout (
            "    *** No. plot files with UNKNOWN plot name  = $i\n");
   for ($i = 0; $i < @arrspice; $i++) {
        print stdout (
            "          " . "(spice name = $arrspice[$i])\n");
    }
   close (spice3log);
   $filerr = chdir ($outdir);
   if ($filerr == 1) {
       opendir (FILES,$outdir);
       @filelist = readdir(FILES);
       chmod (0755,@filelist);
       closedir (FILES);
   }else {
      print stdout (
      "***Error(WXXParse):0024 - Directory ($outdir) does not exist.
Quit.\n");
      die (
      "***Error:0024 - WXXparse terminated due to errors. Check log
file.\n");
   }
   $filerr = chdir ($plotdir);
   if ($filerr == 1) {
       opendir (FILES,$plotdir);
       @filelist = readdir(FILES);
       chmod (0755,@filelist);
       closedir (FILES);
   }else {
      print stdout (
      "***Error(WXXParse):0025 - Directory ($plotdir) does not exist.
Quit.\n");
      die (
      "***Error:0025 - WXXparse terminated due to errors. Check log
file.\n");
   }
#
*************************************************************************
#       Create Plot Filename subroutine
*
```

```
#       Input: currplot = v(nn)_v(nn)   names given to plot data by spice   *
#           Index time  <nodes> where nodes are:                             *
#              -v(nn)                                                         *
#               v(nn)                                                         *
#               v(nn)-v(nn)                                                   *
#         Returns: plotfilename = value
*
#
*************************************************************************
sub cplotname {
    my ($currplot) = @_;
    my ($plottype, $node1, $node2, @arrayn, $nodes, $currnode);
#
*************************************************************************
#          - format nodes:   -v(n) = 0_9; v(8) = 8_0; v(2)-v(4) = 2_4
*
#
*************************************************************************
    $currplot     =~ tr/a-z/A-z/;
    if ($currplot =~ /^-/) {
      @arrayn   = split(/\(/, $currplot);
      $plottype = chop (@arrayn[0]);
        @arrayn   = split(/\(/, $currplot);
        chop (@arrayn[1]);
        $nodes    = "0_" . @arrayn[1];
    } elsif ($currplot =~ /-/) {
           @arrayn = split(/\(/, $currplot);
        $plottype = @arrayn[0];
        @arrayn = split(/-/, $currplot);
        @arrnode1 = split(/\(/, @arrayn[0]);
        chop @arrnode1[1];
        @arrnode2 = split(/\(/, @arrayn[1]);
        if (@arrnode2[1] =~ /\)$/ ) {
           chop @arrnode2[1];
         }
        $nodes = @arrnode1[1] . "_" . @arrnode2[1];
      } else {
        @arrayn = split(/\(/, $currplot);
          $plottype = @arrayn[0];
        chop (@arrayn[1]);
          $nodes = @arrayn[1] . "_0";
        }
#
#
*************************************************************************
#      Based upon the node (e.g., 1_2), find the Print statement        *
#           in the spice3 netlist array arrnodes:                         *
#
*************************************************************************
   $i = 0;
   $findnode = 0;
   while ($findnode == 0) {
     if ($nodes eq @arrnodes[$i] && $plottype eq @arrplotn[$i]) {
        $findnode = 1;
        @arrplotspice[$i] = $currplot;
        $plotfilename     = @arrplotnames[$i];
        print stdout (
        "(WXXparse):0022 - Plot found in netlist .cir file:
$plotfilename\n");
```

```
        } elsif ($i < @arrnodes) {
            $i++;
          } else {
            $findnode     = 2;
            $plotfilename = "UNKN" . $plottype . $nodes . "_00" . $plotnbr;
            $plotnbr      = $plotnbr + 1;
            $j            = @arrspice;
            @arrspice[$j] = $plotfilename;
            print stdout (
              "(WXXparse):0023 - Warning - plot file name inconsistent:\n" .
                "     Spice name = $currplot, Plot name = $plotfilename\n");
          }
       }
     return ($plotfilename);
}
```

This page intentionally left blank

Distribution:

1	MS0311	M. J. De Spain, 2125
1	MS0311	R. P. Roberts, 2671
1	MS0316	P. F. Chavez, 9204
1	MS0405	T. D. Brown, 12333
1	MS0525	P. V. Plunkett, 1734
1	MS0525	S. D. Wix, 1734
1	MS0661	R. F. Billau, 4812
5	MS0746	J. E. Campbell, 6411
1	MS0746	R. M. Cranwell, 6411
1	MS0746	B. M. Thompson, 6411
1	MS0830	E. W. Collins, 12335
1	MS0830	K. V. Diegert, 12335
1	MS1010	R. J. Sikorski, 15222
1	MS9201	P. K. Falcone, 8114 Attn: M. M. Johnson
1	MS9202	W. P. Ballard, 8418
5	MS9202	R. L. Bierbaum, 8418
1	MS9202	K. D. Marx, 8418
1	MS9217	P. E. Nielan, 8920 Attn: R. C. Armstrong
1	MS9217	J. C. Meza, 8950 Attn: T. G. Kolda
3	MS 9018	Central Technical Files, 8940-2
1	MS 0899	Technical Library, 4916
1	MS 9021	Technical Communications Department, 8528/ Technical Library, MS 0899, 4916
1	MS 9021	Technical Communications Department, 8528 For DOE/OSTI
1	MS 0188	D. Chavez, LDRD Office, 4001