USING DESIGNER CONFIDENCE AND A DYNAMIC MONTE CARLO SIMULATION TOOL TO EVALUATE UNCERTAINTY IN SYSTEM MODELS

by

JEFFREY M. LYONS

B.S. Mechanical Engineering
Florida State University, Tallahassee, FL, 1998

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

Signature of Author.....
Department of Mechanical Engineering
May 5, 2000

Certified by.....
David Wallace
Esther and Harold E. Edgerton Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by.....
Ain A. Sonin
Chairman, Department Committee on Graduate Students

USING DESIGNER CONFIDENCE AND A DYNAMIC MONTE CARLO SIMULATION TOOL TO EVALUATE UNCERTAINTY IN SYSTEM MODELS

by

JEFFREY M. LYONS

Submitted to the Department of Mechanical Engineering
on May 5, 2000 in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Mechanical Engineering

ABSTRACT

As the use of distributed engineering models becomes more prevalent, engineers need tools to evaluate the quality of these models and understand how subsystem uncertainty affects predictions of system behavior.

This thesis develops a tool that enables designers and engineers to specify their perceptions of confidence. These data are then translated into appropriate probability distributions. Monte-Carlo-based methods are used to automatically provide correct propagation of these distributions within an integrated modeling environment.

A case study using an assembly tolerance problem is shown and different confidence modeling methods are compared. The methods benchmarked are: worst case; statistical; conventional Monte Carlo simulation; and the dynamic Monte Carlo tool developed in this thesis. Finally the dynamic Monte Carlo tool is used together with surrogate modeling techniques. Comparisons based on implementation time, model execution time, and robustness are provided.

Thesis Supervisor: David Wallace

Title: Ester and Harold Edgerton Associate Professor of Mechanical Engineering

ACKNOWLEDGEMENTS

I would like to thank some people who have made this thesis possible. Thanks to my family, especially my wife Katie, for their support and patience during my time at MIT. To Professor David Wallace for his vision of product development that has inspired me. To all the members of the MIT CADLAB for a truly collaborative work environment.

Special thanks to Matthew Wall for his help and the underlying software architecture that much of this material builds upon. Also to Nick Borland for his continued tutelage throughout my time in the lab.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION: THE NEED FOR ASSESSING DESIGNER CONFIDENCE

Engineers and designers need information about the quality for the data used in their models. Current methods for providing data quality are either simple and non-robust, or robust yet difficult to program. This thesis proposes tools that enable engineers to generate quality information and provide quality feedback to their models.

The World Wide Web (WWW) and other technologies have made it possible to create distributed product design models. Engineers have greater access to data and there has been an explosion of different modeling tools. Thus engineers are increasingly modeling aspects of their designs using computational tools.

Simultaneously, the development of products is increasingly multidisciplinary and distributed in nature. Companies sub-contract parts of the larger system to other companies or different departments of the same company.

As models become more prevalent and more distributed, the need to assess the quality of individual simulations and how they affect the overall performance predictions is becoming increasingly important. When individuals use models created by others they can be misused. Sometimes the user perceives the model as inaccurate when the model creator intended high accuracy. (See section 2.1.)

Other times models represent data deterministically when it should be probabilistic. This is because the normal modeling tools used have little or no provisions for non-deterministic modeling. The model users then see the deterministic models and assume the model is more accurate than intended. If supporting documentation is created it is often not seen by the model users. (See section 2.1.)

The quality of the overall predictions will in part be determined from the quality of the individual data parameters it is built upon. Thus a tool for assigning quality information to the parameters is needed. Also, the model itself introduces uncertainty. The tool must

allow the user to account for this uncertainty. It is suggested that the engineers[1] performing the simulations, estimating the parameters and building the models are in the best position to provide this information. This thesis develops a tool for mapping engineers' perceptions of quality to probability distributions.

Once these distributions exist, the model evaluation becomes much more complex. Current methods for evaluating uncertainty are either inaccurate, leading to over-design, or complex and tedious to set up. Engineers tend to use the easiest methods (worst-case analysis or statistical analysis) if possible (Eggert 1995). These methods lead to over-design and higher cost. If necessary, complex Monte Carlo methods are implemented (Kalos 1986).

Difficulties of the Monte Carlo method can be grouped into three areas.

1. Experienced computer experts, as opposed to the engineer designing the component or system, must implement simulations.

2. Once the system is implemented, it is not dynamic; it must be rebuilt if there are even minor design changes.

3. Assigning distributions to design parameters. Engineers are not good at determining probability distributions. (See section 4.2.)

4. Monte Carlo simulations are computationally intensive and thus require long computation time.

The method proposed in this thesis targets the difficulties of the Monte Carlo method by creating a Dynamic Monte Carlo Tool (DMCT). DOME (Distributed Object-based Modeling Environment) has created a modeling environment that allows an engineer to

[1] Throughout this paper when referring to engineers, the statements often apply to all parties participating in the product design process. This can include design engineers, cost experts, managers, and many others.

integrate models into a heterogeneous distributed system model (Wallace *et al* 2000). DOME provides a graphical user interface to modify the system model dynamically.

The software architecture allows for system analysis of the model such that all the dependencies in the model can be determined. The tool shown in this paper uses this architecture to analyze the system, determine if a Monte Carlo simulation is needed and run a simulation if necessary. When the model is modified, the DMCT reanalyzes it and adjusts the simulation accordingly.

Also provided are methods to aid the engineer in creating probability distributions based on his perceptions of the design. The tool can then be used in collaboration with surrogate modeling tools (already available in DOME) to reduce the computation time.

Thus the four greatest difficulties of the Monte Carlo method are addressed. Building the system is done graphically, not by a computer expert. The Monte Carlo analysis is dynamic; it doesn't need to be rebuilt. A methodology and graphical user interface is provided to help engineers create distributions to describe their uncertainty of design parameters. Surrogate modeling tools can be applied to the models to significantly reduce computation time.

2 BACKGROUND

2.1 Necessity of quality feedback

Uncertainty in engineering may occur through three paths (Siddall 1983).

1. Uncertainty may be due to measured quantities or variables dependent on measured quantities.

2. There may be uncertainty concerning an event that may or may not occur; or the event may be certain to happen but its time of occurrence may be uncertain.

3. There can be uncertainty regarding the validity of a hypothesis or theory used to predict the performance of an engineering design. An example of this type of uncertainty would be confidence in an engineering model to predict the performance of a design.

A DOME application to an automotive door moveable glass system at Ford Motor Company (Wallace *et al* 2000) was used to study issues related to the quality of individual models. The perception of model quality was observed from the standpoint of the model creator and other parties using the models. The main uncertainty issues found were those listed as one and three in the preceding list, namely uncertainty in measured parameters and confidence in an engineering model of a physical system.

It was found that an engineer's confidence in his own model often exceeds that of those using the model. The model builder who spent time and knows the underlying theory under the model may be prone to have more confidence in it. The model user may not be as familiar with the theory and thus not perceive it as accurate.

Occasionally the opposite happens; individuals using another's model misuse the model by overestimating its accuracy. The modeler's intention may have been to provide rough

estimations, yet the modeling tool provided no mechanisms to incorporate the quality observations into the model.

To remedy this, experts in an area create guidelines for the use of the models and parameterization of products. However it was found that these guidelines were not used. This was most likely because the models have no way of being linked to the related guidelines. Although the guidelines were referenced in the models, the users did not take the time to look them up and use them.

Another issue was the lack of model verification. Often there is a single expert in a discipline that creates the model and thus no other experts are available to check it. One particular example seen was in the area of finite element analysis. At a smaller company there is often only one FEA expert. One expert is sufficient to complete the FEA analyses required for the entire company, but having one expert makes model verification problematic.

Even when other experts are available, usually the models were not verified. It was assumed that the modeler had made no errors, which is unlikely in models that can be very complex.

2.2 Methods for providing designers with quality feedback

As previously discussed, one of the greatest difficulties in engineering is creating models of physical systems that accurately reflect reality. An engineer's perception of the physical system is not always exact. Also, systems must be simplified to make modeling possible. Thus errors are introduced into the model. Engineers are aware of these uncertainties and use different tools address them.

2.2.1 Worst Case Analysis

One common analytical method is the "worst case" approach. It is easier to implement than complex probability analyses. The engineer can use his deterministic model. The model parameters are set based on what would result in the poorest performance. If the

model returns a satisfactory design with an appropriate safety factor, then the reality (that should never be worse than what was modeled) should also be satisfactory (Eggert 1995).

This method is simple to implement because it requires no extra modeling in addition to what was already done for the deterministic model. One has to determine the worst case parameters and use them. However this method has many inherent problems.

Deciding what the worst case parameter values are may be trivial for a simple model. But once a model becomes complex, it may be difficult to determine. Once there are more than about ten parameters and as many relationships between them, it is difficult to determine the combination of values that will result in the poorest performance (Eggert 1995).

Also, the likelihood of all parameters being at worst case is very low. The model gives no indication as to the likelihood of failure. The more complex the system, the less the model mimics reality. The engineer is forced to create a design that will perform under the most unlikely (and often impossible) circumstances. This leads to a final product that is over-designed, which often means wasted material and high cost.

To get the product to perform to these unrealistic circumstances, the engineer has to keep tolerances very low. As tolerances decrease, production cost increases at a great rate (Haugen 1980, see Figure 2-1).

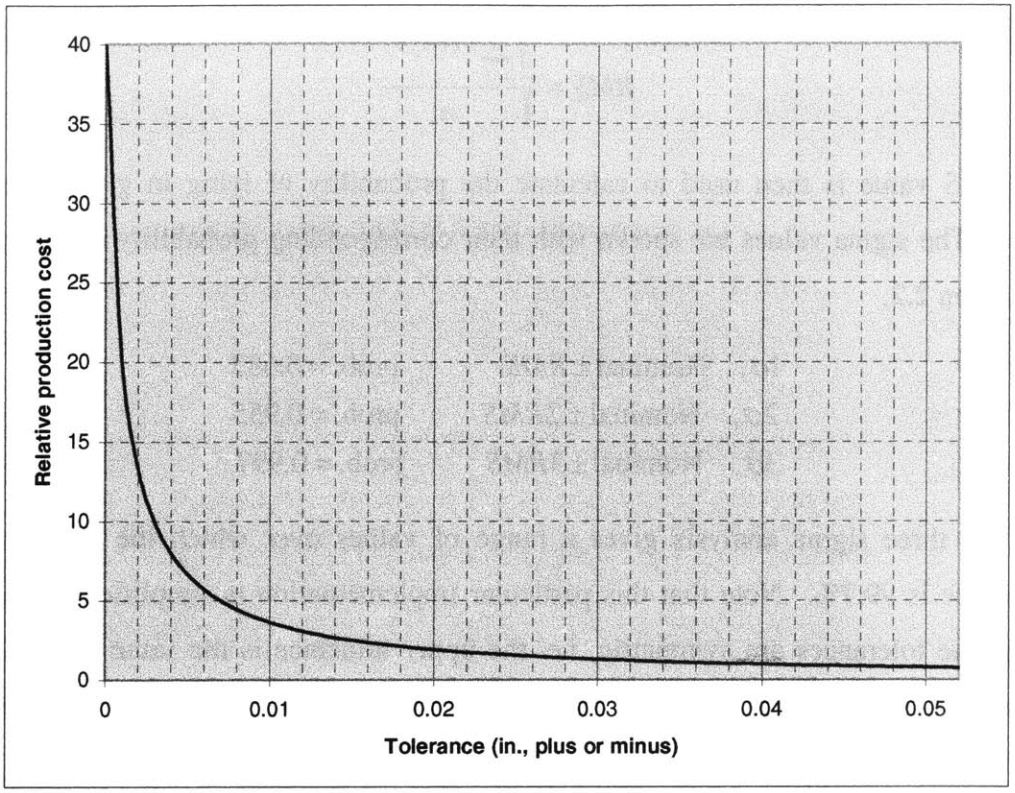


Figure 2-1[2] Relative production cost of tolerances

2.2.2 Statistical Analysis

A second, more robust, set of methods uses elementary probability theory to determine the likelihood of parameters existing within a certain range. In the area of dimensional analysis this method is called the statistical tolerance system (Shigley *et al* 1989). Here one sets the tolerance ranges of a system's parameters not so that worst case will be successful, but so that failure only occurs within a certain percentage of cases, the percentage depending on the requirements of the application. Two common acceptance ranges are 3 sigma (99.73% acceptable) and 6 sigma (nearly 100% acceptable). For this reason this analysis is sometimes called a three or six sigma analysis.

To perform this analysis, first a deterministic result is calculated using the nominal values. Then a root mean square constant is calculated. Suppose a series of n dimensions has tolerances Tol_n. The root mean square is calculated using equation 2.1:

[2] Adapted from Probabilistic Mechanical Design, Haugen (1980)

$$RMS = \sqrt{\frac{\sum_{i=1}^{n} (Tol_i)^2}{n}} \qquad 2.1$$

This RMS value is then used to calculate the probability of lying in given tolerance ranges. The sigma values are shown with their corresponding probability of occurrence in equation 2.2.

$$\begin{aligned} 1\sigma, &\quad \text{Nominal} \pm RMS && \text{prob.} = 0.683 \\ 2\sigma, &\quad \text{Nominal} \pm 2RMS && \text{prob.} = 0.955 \\ 3\sigma, &\quad \text{Nominal} \pm 3RMS && \text{prob.} = 0.997 \end{aligned} \qquad 2.2$$

Thus the three sigma analysis gives a range of values over which the likelihood of occurrence is 99.7%. Note that this particular implementation is simplified by the fact that all the tolerances are symmetric, i.e. the upper tolerance is the same as the lower. This does not have to be the case, but non-symmetric tolerance ranges complicate the analysis.

A system designed with this method will not be as over-designed as one using the "worst case" method. It is not as complex as many other probabilistic methods and can be implemented rather easily. However, it does require extra work by the engineer. Many don't know how to use it and those who do often do not understand it well enough to implement it properly. An assumption of this method is that all points within a tolerance range are equally likely. Thus the distribution is modeled uniformly, treating a point at the very end of the range as equally likely as the mean. Reality usually emulates a normal distribution, modeling occurrences near the mean values as more likely.

2.2.3 Monte Carlo Simulation

A third method is assigning probability distributions to the parameters and combining those to determine probabilistically the resulting system. A method to do mathematical operations on the distributions is the Monte Carlo method. This method takes samples from the independent distributions, executes the pre-defined relationships and creates a list of resulting values for the dependent distributions.

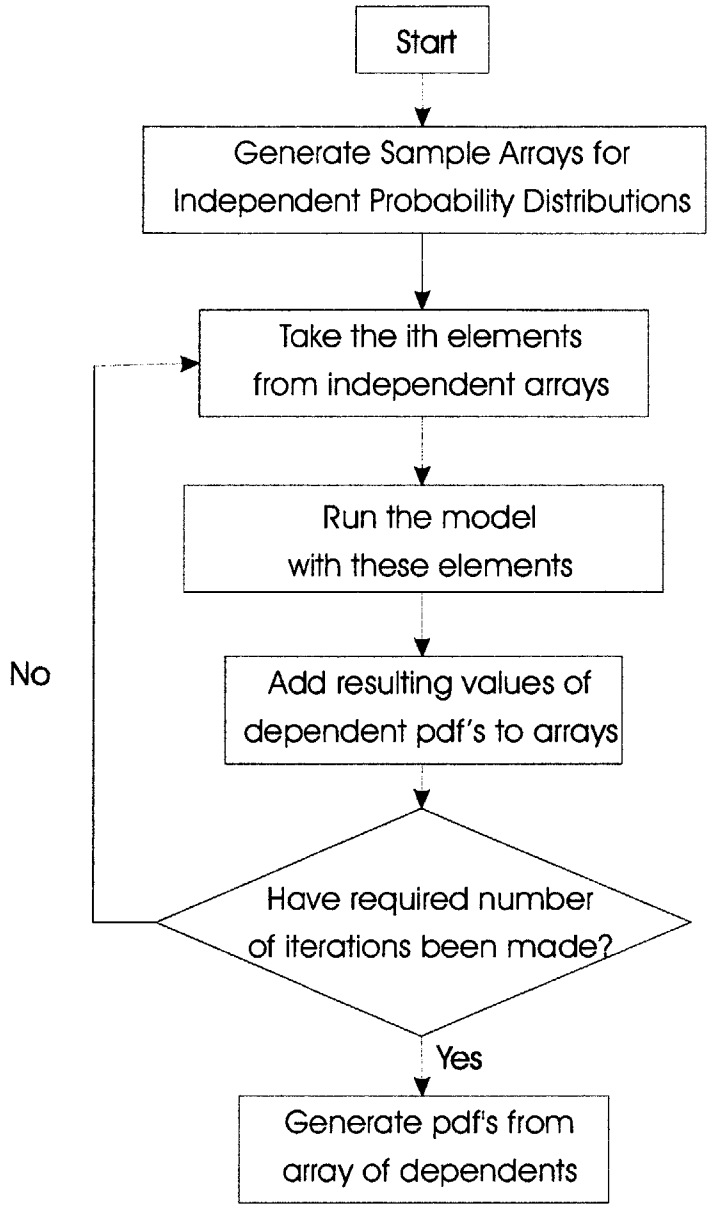


Figure 2-2[3] Monte Carlo simulation process

Appropriate distributions are assigned to all of the independent parameters. Arrays of samples for these are created based on their associated distributions. Then the simulation iterations begin.

The model's mathematical relations are executed based on the first array element for the independents. The dependents' resulting values are added as the first elements in their own arrays. If there is any correlation (i.e. independence can not be assumed) then the

dependent parameters must be correlated. (Correlation is explained in detail below.) The process is then repeated with the second array elements, and so on until the required number of iterations has been achieved.

Depending on the simulation requirements, different operations can be performed on the arrays of the dependent parameters. Distributions can be built from the histogram, or perhaps only the mean and standard deviation are required.

Correlation can make a Monte Carlo simulation much more complex. Suppose a model has four parameters and two relationships. The parameters are A, B, C and D. The relationships are "A + B = C" and "B + C = D". Obviously A and B are independent; they depend on no other parameter. C, however depends on A and B, and D depends on B and C. Thus B and C are correlated; C is an output of B yet they both are required to calculate D.

[3] Adapted from Monte Carlo Methods, Kalos (1986)

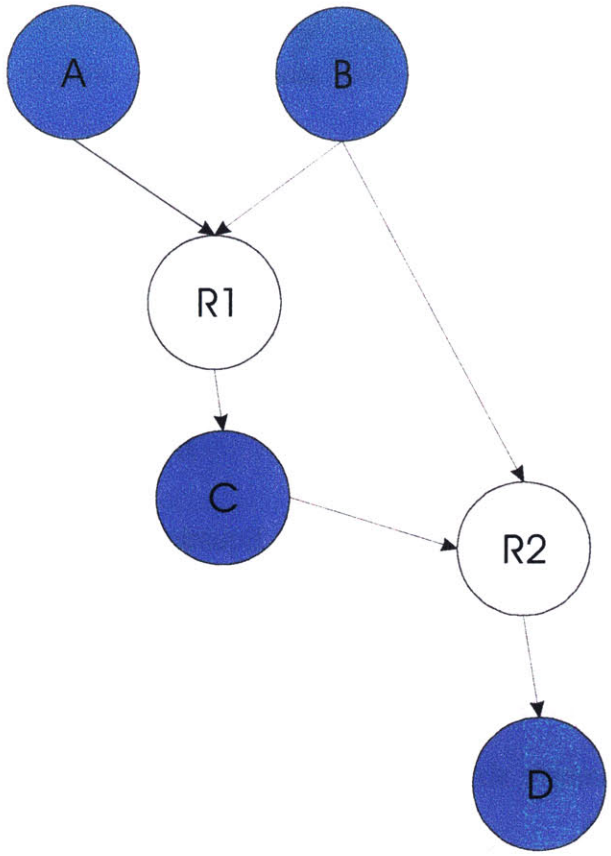


Figure 2-3 Model with correlation

Due to the correlation, for each iteration the C value used to calculate D must correspond to the B value that resulted in it (the C value). For example, suppose that both A and B have a mean of 10.0. If, on a given iteration, samples are generated such that A is 10.1 and B is 9.9, C would then be 20.0. Thus D would be calculated as 9.9 + 20.0. So relation R1 would need to be executed first. Then using its result for C and the B sample used in R1, R2 would execute to get the correct value for D.

Determining an appropriate sample size is a difficult tradeoff with Monte Carlo simulations. Simulations require several thousand iterations, but many iterations are computationally expensive.

To understand the sample size problem, consider the dependent variable C. From the simulation, C's mean is calculated using 2.3:

$$\overline{C} = \sum_{i=1}^{n} \frac{C_i}{n} \tag{2.3}$$

where n is the total number of samples. Consider the sample mean $\overline{C}$ as a random variable. For an accurate simulation, we would like to minimize its variance. The variance of $\overline{C}$ is calculated using 2.4:

$$E(\overline{C} - \mu_C)^2 = \frac{\sigma_C^2}{n} \tag{2.4}$$

where σ_C^2 is the variance of the sample array. Equation 2.5 follows:

$$\sigma_{\overline{C}} = \frac{\sigma_C}{\sqrt{n}} \tag{2.5}$$

Thus increasing the sample size only improves the simulation by the square root of the increase (Siddall 1983).

Suppose A and B both have a standard deviation of 1.0, thus C's standard deviation should be 1.4. So if the sample size is 10,000, the standard deviation of the sample mean is 0.014. This standard deviation represents the error of the sample mean. So although this deviation is relatively small for distributions, it is not ideal for the sample mean which is theoritically deterministic. Thus this deviation should be minimized as much as possible.

This method is the most robust and accurate. Parameters can be modeled as deterministic values as well as any type of probability distribution. Thus reality can more closely be represented. The results are in terms of probability distributions, giving a complete picture of the possible system output.

This method is so difficult that it tends to be avoided. When it is used, it is implemented by an experienced computer programmer familiar with probability theory as opposed to the engineer designing the system. Once the system is implemented, it is not dynamic; it

must be rebuilt if there are even minor design changes. Another difficulty is assigning distributions to the parameters. Engineers are not good at determining probability distributions based purely on their experience.

The difficulties of the Monte Carlo method are addressed in this quote:

> There are several characteristics that are necessary to all well-designed simulations. *A clear statement of the system to be simulated is needed.* Once this is accomplished, the probability distribution functions that will be involved must be identified and explicitly defined. Methods must then be established to sample all the required pdf's. And finally, *one must be able to understand how to interpret the information provided by the simulation.*
>
> Kalos 1986, italics added

This author states that the simulation designer must have a complete understanding of the system in order to correctly build the simulation.

2.2.4 Approximation methods

2.2.4.1 Polynomial approximation

Due to the computational intensity of Monte Carlo simulations, much research has focused on achieving the method's accuracy through approximation (Tatang 1994). These methods draw on the same mathematical methods used to reduce complexity of simpler models (e.g., differential equations to algebraic equations).

The basic procedure is to run the model a few iterations (as if running a Monte Carlo simulation) and then create polynomial approximations of the model based on these. The error is calculated, and if it is too high the order of the approximations is increased.

These methods have been very successful in reducing computational intensity. However they share many of the limitations and barriers of Monte Carlo simulations. For example, computer and probability experts must implement these simulations and the simulations are not easily adaptable.

2.2.4.2 Neural networks

An alternative to the polynomial approximation method would be to use a Monte Carlo simulation tool in conjunction with a neural network. Neural networks are algorithms based on their biological namesake that approximate system outputs based on inputs. Some applications include classification, time series prediction, optimization, and computational solving (Deniz 2000, Masters 1993).

In order for this to be a feasible tool, the Monte Carlo tool must overcome the barriers previously discussed and the neural network must be robust and easy to use.

2.3 DOME

The DOME (Distributed Object-based Modeling Environment) project going on currently at the MIT CADLAB is addressing the problem of system engineering. Product complexity has increased while organizations are becoming larger and more spread out. More and more OEM's prefer to out–source components. This has made creating system models very difficult. DOME uses recent information technology developments to address these challenges (Wallace *et al* 2000).

Engineers, managers, cost analysts, and others work with the tools they decide are the best to accomplish their job functions. They should not all be forced to use certain tools just because they are compatible with one another.

Using DOME, engineers, managers, and others involved in the design process publish modeling services on the web to allow others access to their models. The level of access is completely determined by the owner of the model. They are not turning over their model, just the capability to run the model remotely and get outputs from it based on their inputs. The publishing process depends on the software tool, but it is always trivial and impacts workflow minimally.

After the publishing process, the service interfaces are accessible via the internet through a DOME server network. System integrators can then use DOME to create relationships between the published services to complete an overall system model.

This modeling environment is dynamic and non–centralized. The model may constantly evolve. There is no central control over the system model. Relationships may be defined by whomever has the knowledge to do so (Abrahamson *et al* 1999).

This dynamic modeling tool needs a dynamic probabilistic analysis tool. Users may choose to define their services as probability distributions. To perform the mathematics on these parameters, the system modeler should not have to write a probabilistic simulation for every iteration of the model. The underlying simulation should happen automatically and appropriately without custom programming.

This paper focuses on adding functionality to DOME that allows users to add probabilistic distributions to DOME system models. The mathematics between these distributions is implemented in a way that the DOME user does not have to implement the underlying simulation. This underlying simulation is based on Monte Carlo methods.

3 PROPAGATION OF PROBABILISTIC UNCERTAINTY IN DISTRIBUTED MODELS

3.1 Probabilistic propagation

Most research in this area focuses on increasing the performance of Monte Carlo simulations and/or reducing their complexity.

The probabilistic collocation method is used when running the model is very expensive. For many simulations, the number of iterations required for a Monte Carlo simulation takes a very long time due to the computational intensity of the model. Instead this method runs just enough times to get a polynomial approximation of the system. Then it solves the system using the less expensive polynomial approximation. (Webster 1996)

This method increases the performance of the simulations, but it does not address the issues of time consuming implementation or level of expertise necessary.

Another method is to use algebra of expectation (basic probability mathematics) with the expected value and standard deviations of the parameters (Haugen 1980). This method has the advantage of simple equations that could be implemented in a spreadsheet program.

The method is limited by the types of distributions supported. There are infinite probability distributions possible, yet the equations are limited to the basic types (normal, uniform, etc.). Also, when there is correlation in the model, correlation constants must be determined and the mathematics is more complex (Haugen 1980).

As spreadsheet capabilities have increased, some have tried to implement a Monte Carlo simulation completely within a spreadsheet (Eggert 1995). Many engineers now use spreadsheets so this format would be more familiar to them. A deterministic spreadsheet model is used, with the addition of coefficients of variance for the variables that will be

probabilistic. To generate probabilistic outputs, advanced spreadsheet algorithms must be written to vary the inputs and keep track of the results.

This method reduces the complexity for engineers since they are more accustomed to working with spreadsheets. However, current research has implemented this on a "per spreadsheet" basis, redesigning the simulation for each system model generated. Thus the issue of implementation time and complexity is not addressed.

If this research were extracted so that it could easily be applied to any spreadsheet in a given program, it would be a powerful tool. This would greatly reduce time to implement a Monte Carlo simulation and the level of expertise necessary. However, it would still confine an engineer to working in a given spreadsheet program, reducing his system modeling capabilities.

3.2 Application to DOME

As stated in section 2.2.3, a simulation designer needs a complete understanding of the system to correctly build the simulation (Kalos 1986). While this is still generally true, the combination of sorting algorithms and an appropriately designed modeling architecture change the paradigm. The model understanding necessary to design the simulation no longer need be understood explicitly by the human modeler; the Dynamic Monte Carlo Tool analyzes the model to gain this understanding and builds the simulation accordingly.

Thus the DMCT has in a way become the simulation designer, or at least has assumed most of his responsibilities. This next section explains how this has been accomplished.

3.2.1 Local scope

3.2.1.1 Theory and algorithms

To make a Monte Carlo method that was useable to all engineers, it should be easy to use, have minimal impact to their normal work process and have good documentation on use. Also it should handle correlation (not assume independence) and not require the

user to have vast knowledge in the areas of probability and software implementation. Model calculation time should be short enough to allow iterative design searches (Regnier *et al,* 1995).

To satisfy these requirements, it was determined that the tool should be self-sorting and should handle the correlation automatically. It would be possible to accomplish this through basic graph sorting algorithms (Sedgewick 1992; also see Appendix A).

Figure 3-1 is a modification of Figure 2-2, the typical Monte Carlo simulation.

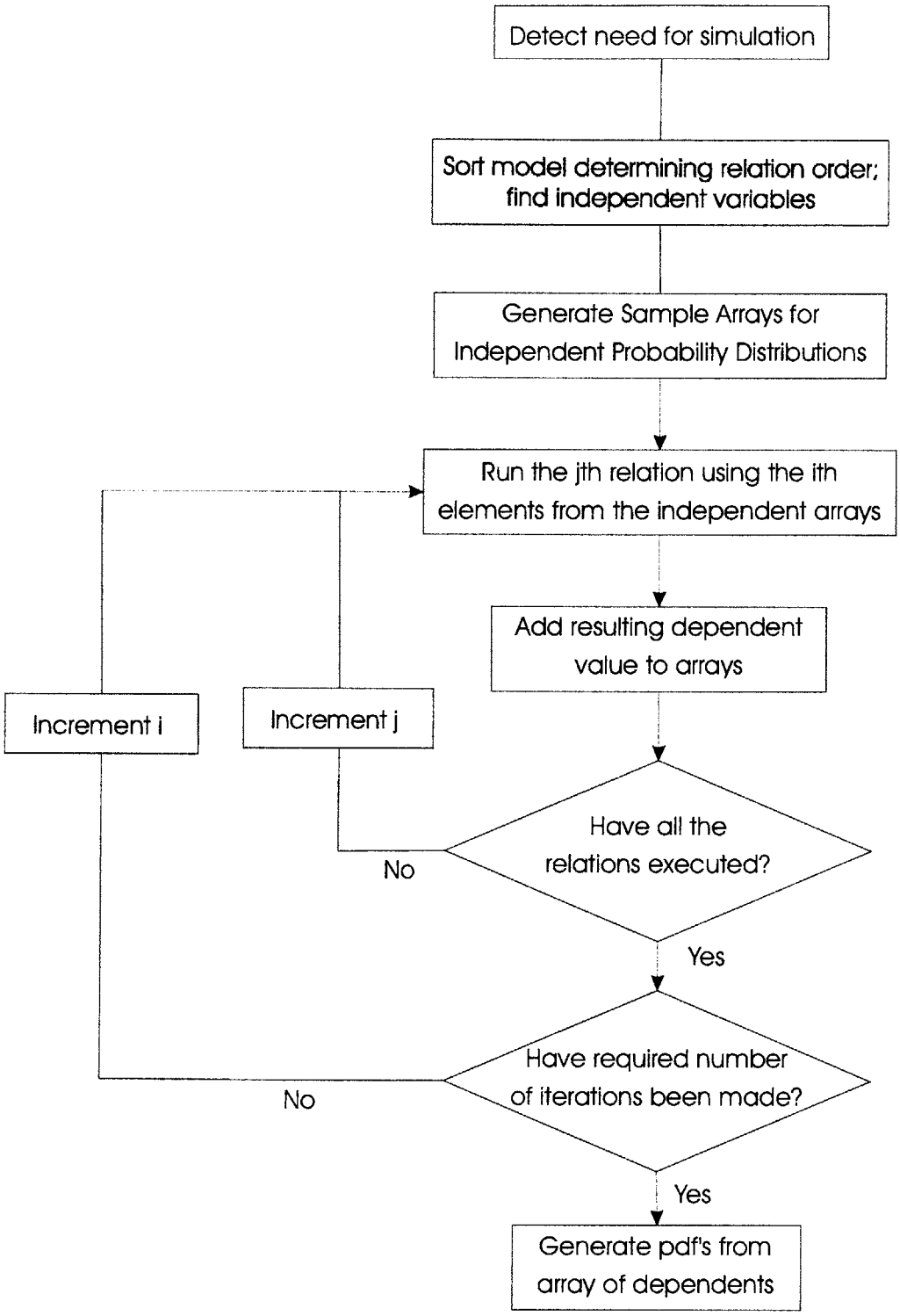


Figure 3-1 DMCT algorithm

The principle difference between the DMCT and the conventional Monte Carlo simulation is that the model determines the relation order and correlation automatically.

There are two loops, one inside the other. The inner loop executes all the relations in the appropriate order. Once all relations have run and one set of dependent samples has been created, the model starts the next iteration and repeats until all iterations are complete.

To accomplish the sorting and execution of relations, the DMCT acts as a Monte Carlo Manager. There is one manager per model, thus the manager handles all the sorting of the model, executes all the relations, and keeps track of the generated and resulting samples. The DMCT is integrated with the modeling environment. It is transparent to the user.

Determining the sample size dynamically was not trivial. It was not considered desirable to have a preset sample size for any simulation, as different simulations have different requirements. Thus the required sample size had to be calculated based on each simulation run.

The formula for determining variable accuracy as described in the background section is shown in (3.1):

$$\sigma_{\overline{C}} = \frac{\sigma_C}{\sqrt{n}} \qquad 3.1$$

where C is a dependent variable, n is sample size, $\overline{C}$ is the sample mean and σ represents standard deviations of these parameters. Thus finding accuracy during the simulation is difficult because the standard deviation of the resulting variables is required.

To determine appropriate values for $\sigma_{\overline{C}}$, experimentation was done. The appropriate value was not constant for all distributions. To achieve smooth normal distributions (when normal distributions are appropriate) it was found that $\sigma_{\overline{C}}$ should be equal to approximately 1/100 the value of σ_C. The actual formulas used are seen in (3.2):

$$\sigma_{\bar{C}} = .0102\sigma_C + 0.0006 \quad \text{if} \quad \sigma_C \leq 1$$
$$\sigma_{\bar{C}} = .0124\sigma_C - 0.0011 \quad \text{if} \quad \sigma_C > 1 \qquad 3.2$$

Or solving for n, we get (3.3):

$$n = \left(\frac{\sigma_C}{.0102\sigma_C + 0.0006} \right)^2 \quad \text{if} \quad \sigma_C \leq 1$$
$$n = \left(\frac{\sigma_C}{.0124\sigma_C - 0.0011} \right)^2 \quad \text{if} \quad \sigma_C > 1 \qquad 3.3$$

Thus the number of samples could be solved for based on the standard deviation of the dependent variables. To determine the standard deviation of the dependent variables, the simulation had to start running. Due to the relative computational intensity of computing the standard deviation of an array, it was undesirable to compute it every iteration. Thus it was calculated every couple hundred iterations. Once the standard deviation converges, n is calculated and the simulation continues running until n iterations have completed.

3.2.1.2 Implementation and example

Through a simple GUI[4], real numbers, probability density functions, containers, and many other types of modeling services can be added to your system model. For example to build a simple model where a normal probability density function named "A" plus a normal probability density function "B" equals a probability density function named "C", an engineer would merely add the distributions to the model, and then define a relationship that stated "C = A + B". This procedure would take 30 seconds or less, and the following model would be created.

[4] Graphical User Interface

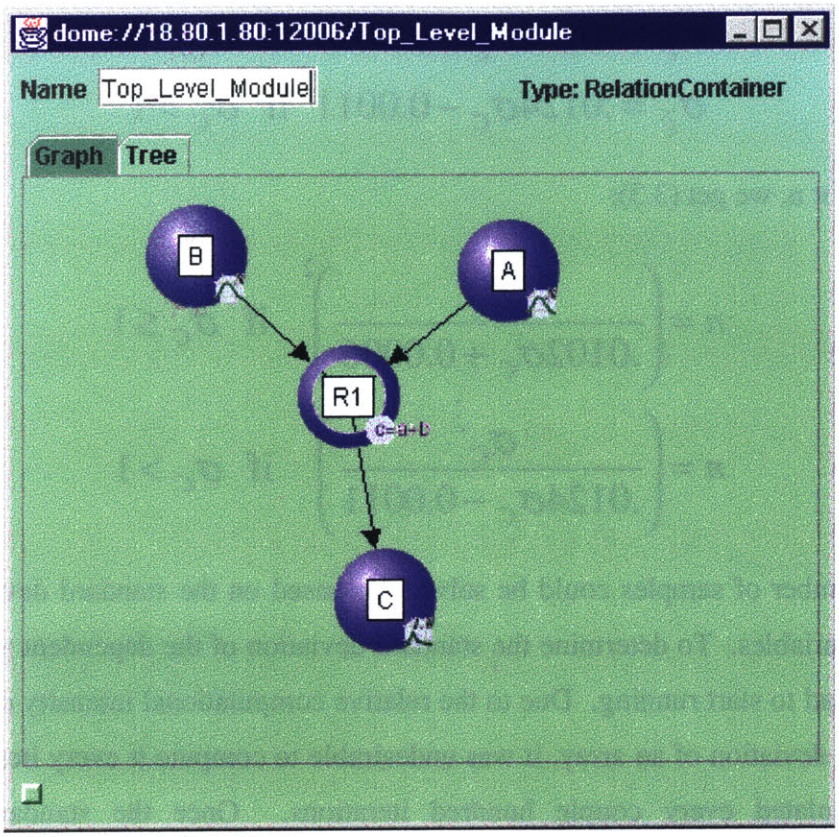


Figure 3-2 DOME model building through R1

To add complexity, suppose a probability density function "D" was added that is equal to the result of "C + B".

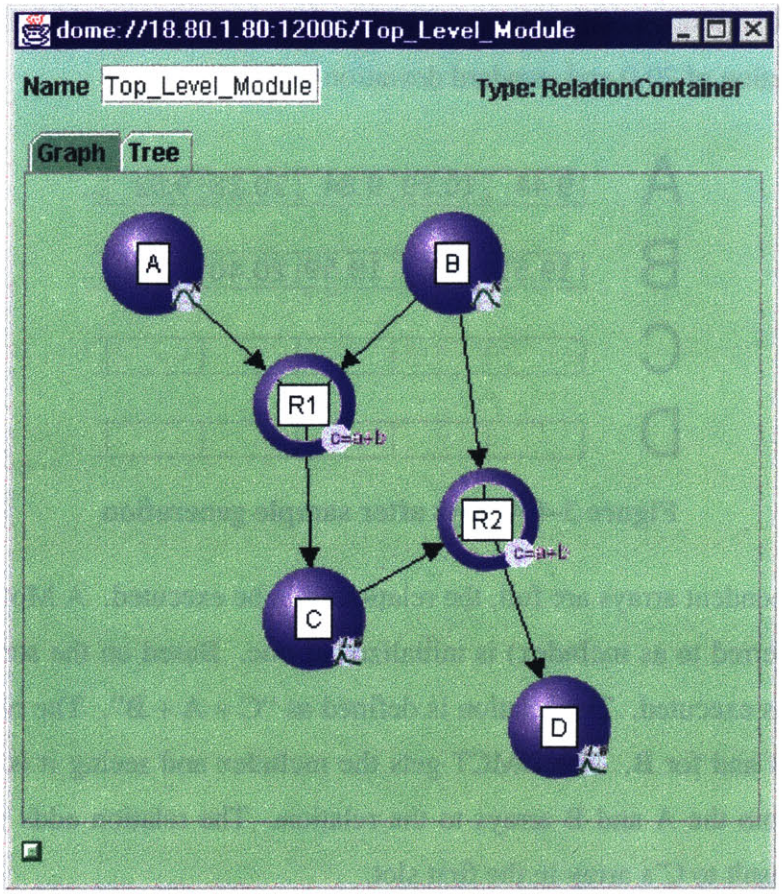


Figure 3-3 DOME model complete

The model now has correlation between the distributions, i.e. assuming independence will give erroneous results. The samples of B must be correlated with the resulting samples of C in the calculation of D.

Here we see why the relations must be executed in the appropriate order to achieve correct results. R2 could not be run before R1 because we need the result of R1 (C) to calculate R2. The DOME architecture was designed to make it possible to determine relationships in a dynamic model. The graph algorithm used was a basic C++ graphing algorithm (Sedgewick 1992). The algorithm implementation in DOME is explained in Appendix A.

After the DMCT determines relation order, it generates samples for the independent parameters. In the example shown, A and B are found to be independent. Suppose that

A is a normal distribution with a mean of 10.0 and a standard deviation of 0.5 and B is a normal distribution of 20.0 and standard deviation of 0.5.

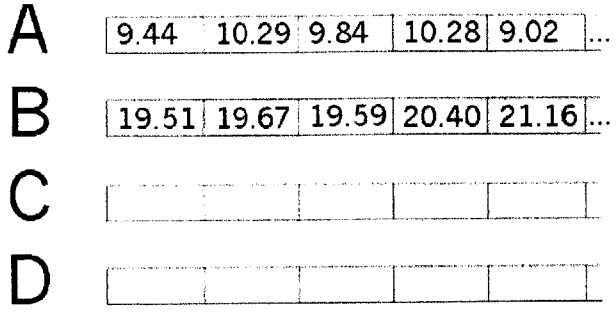


Figure 3-4 Arrays after sample generation

Once the independent arrays are full, the relations can be executed. A Monte Carlo index (heretofore referred to as mcIndex) is initialized at one. Based on the sorting procedure relation "R1" is executed. The relation is defined as "C = A + B". The relation requests a sample for A and for B. The DMCT gets the mcIndex and seeing it is one, sends the first sample from the A and B arrays to the relation. The relation adds 9.44 and 19.51 and adds the result to C's array in the first slot.

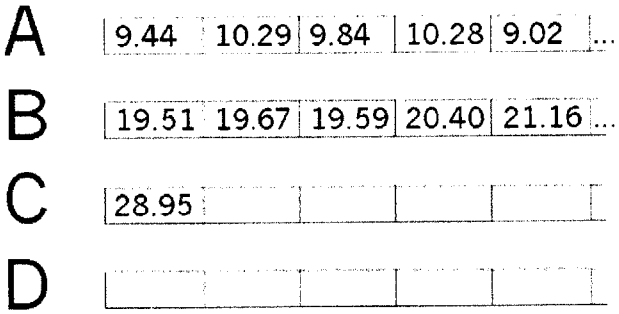


Figure 3-5 Arrays after R1, iteration 1

The DMCT then executes relation "R2" which defines "D = C + B". In the same way, the relation requests samples for B and C. mcIndex is still one, therefore the first elements of B and C are sent. Here it is seen how critical it is that the relations execute in the appropriate order. If "R2" went first, the first element of C would still be blank. Using these samples, D is calculated.

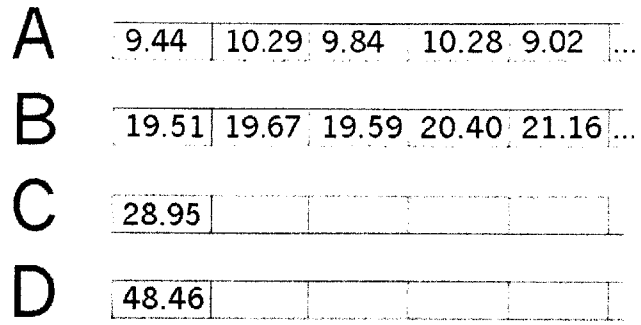


Figure 3-6 Arrays after R2, iteration 1

Now all the relations have fired and the first elements of all the dependent arrays have been filled. So mcIndex is incremented to two, and the relations are again executed in the same manner.

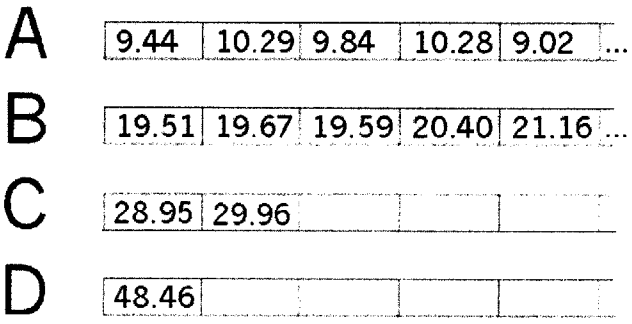


Figure 3-7 Arrays after R1, iteration 2

In this manner it continues until all the arrays are full. The dependent probability density functions are then generated based on the resulting arrays of samples.

Thus the problem of correlation is addressed by the solving architecture. Each "column" from the figure is a consistent, deterministic model state. By executing relations in the appropriate order and treating each increment of mcIndex as its own state, the model is consistent for each increment and the dependents are automatically correlated appropriately.

This architecture also solves the problem of running a Monte Carlo through a "black box". As explained in the background section, DOME has the capability of interfacing with multiple programs. These programs are treated as black boxes, meaning that the

contents and functionality of the external models are not known. The owner of the black box defines what inputs he requires from DOME and what outputs he will provide to DOME. The calculations done on the numbers in the black box are not needed outside of the black box.

The DMCT treats the black box as a relation. Samples are sent to the black box (based on mcIndex), and the black box output (or outputs) is added to the output array. So the black box simulation is run once for each iteration.

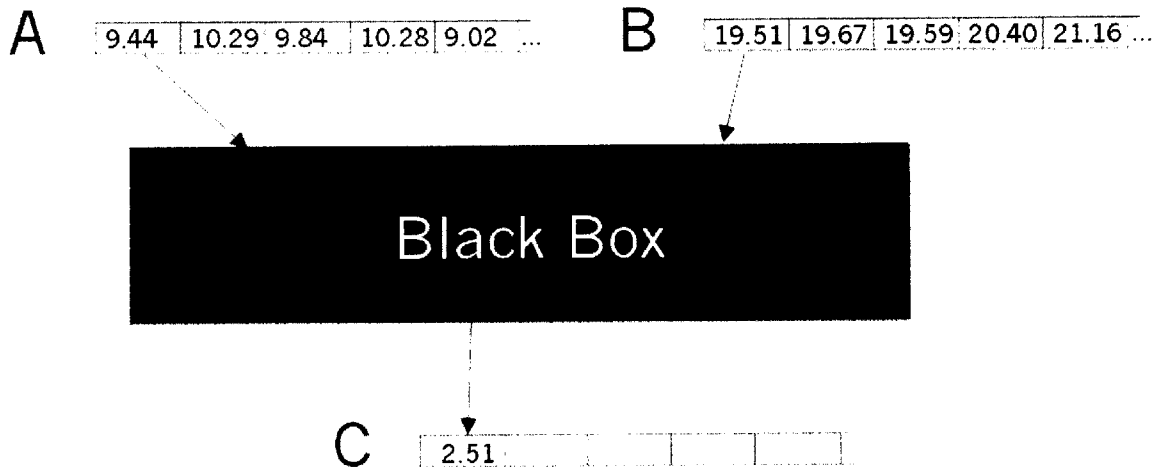


Figure 3-8 Black box

This black box implementation has obvious limitations. If the black box simulation is slow, running the full Monte Carlo simulation will be very slow. Running the Monte Carlo simulation with DOME defined relations is very quick; adding in the black box simulation can greatly reduce efficiency depending on the speed of the third party program.

Another limitation is that the inputs to the black box are assumed to be independent. There can not be correlation between A and B for example.

3.2.2 Distributed Model

The above implementation assumes that the overall model structure can be analyzed. This is not always the case. The DOME architecture and many other current systems

have been designed with distributed capability. A model existing on a DOME server can be logged into by those with the appropriate permissions at a remote site. Services from the remote DOME server can be put into a local model. These remote services are known as "aliases". The original service is known as the "target". The aliases in turn can be related to local services through mathematical relationships. This concept is key in today's environment where suppliers and even departments of the same company are at remote sites and need an overall system model.

Implementing a mathematical solver for a distributed architecture is problematic. On a local scale, DOME and other mathematical tools solve models using directed graph algorithms. (See appendix A.1.) However when a model is distributed, the overall model structure is not easily known, making solving the model impractical.

The current DOME architecture listens for changes, and when they occur it solves the model locally and runs it accodingly. This architecture is usually sufficient for solving deterministic distributed models. Although the model architecture is not required to be managed on a system level, the local solving is usually sufficient except ocassionally when an inifinite loop might be created remotely. Changes propagate to remote models and those get solved appropriately.

This problem affects the DMCT in a more serious manner than the deterministic case, as understanding the model on a system level is key to running the relations in an appropriate manner. Take this model for example.

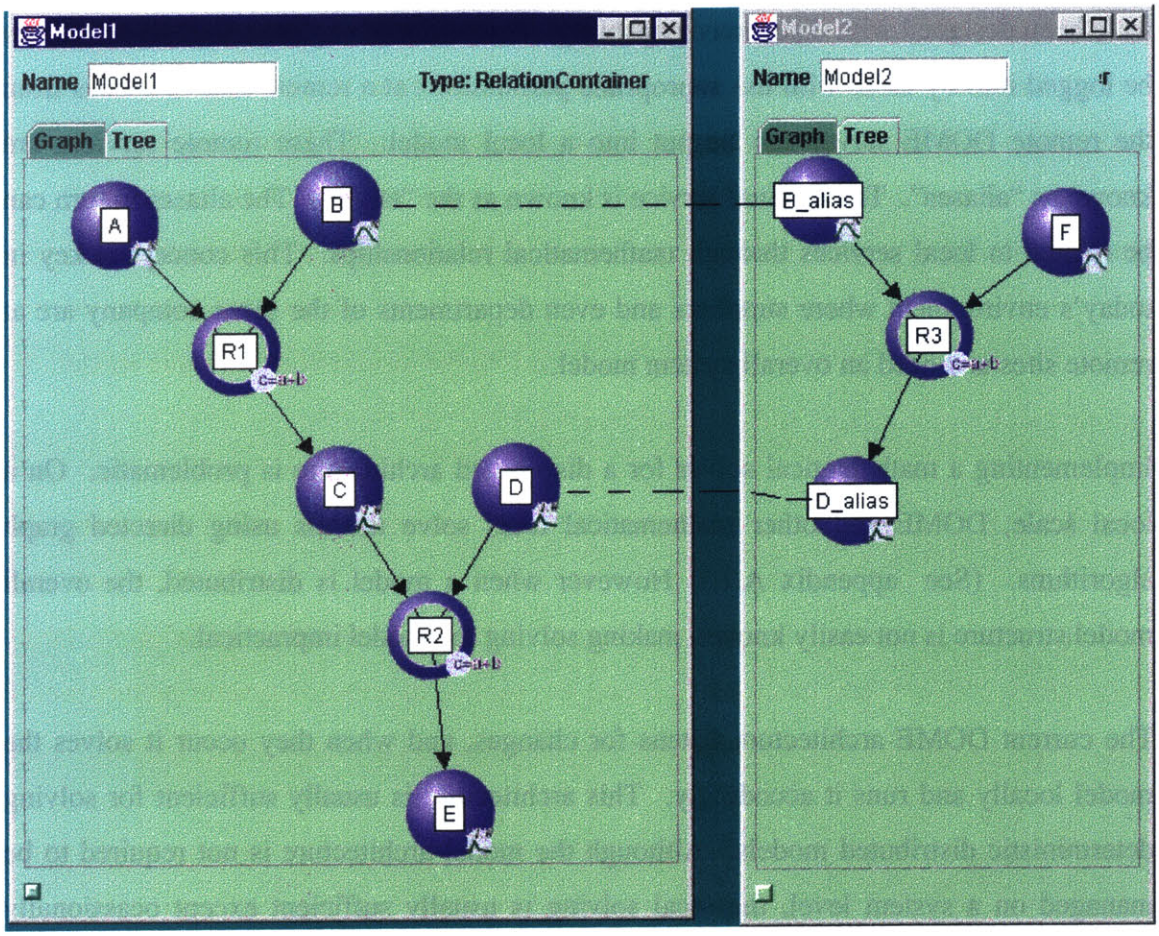


Figure 3-9 Distributed DOME model

Remote aliases to B and D, B_alias and D_alias, exist in Model2. In this case B_alias and D_alias are correlated in Model2, yet Model1 has no way of knowing that. Thus Model1 has no way of knowing that R3 should run before R2. It would interpret D as an independent. There are some possible solutions to this problem.

Some current research in the MIT CADLAB is focusing on sensitivity analyses that could aid in overall model understanding. The model would be able to detemine dependencies not based on their explicit definitions, but listening to the results of simulations. The model could then be run accordingly. This is problematic however, in that for the model to do these sensitivity analyses the system model must run. Thus when it ran it would be done assuming independence for B and D. This would affect the sensitivity analysis negatively.

Another possibility would be having the local managers work together to create an overall system picture. They would analyze their own scope and then exchange information with eachother to complete the distributed picture. This is most likely not a plausible nor desirable solution. One of the advantages of the DOME architecture and other distributed systems is that you can create distributed system models while keeping your proprietary services protected. Permissions can be set up on a module by module basis. By giving a DOME server complete access to the structure of a remote model, the remote model's security would be compromised.

Also, DOME allows for linear scalability of models, meaning the model can grow very large with relatively low computational cost since local objects manage their own data without requiring much outside information. Thus models can grow very large. Requiring a Monte Carlo manager to create a complete distributed system picture would result in far too much computational cost and would limit the scalability of DOME.

A more feasible solution would be one where a local Monte Carlo manager would make service requests to remote Monte Carlo managers, then wait for notifiication from the remote managers that the local manager may proceed.

This requires that Model1 knows if and where remote aliases and/or remote targets exist. (This capability is currently supported in DOME.) The first model to see a change would start the simulation. For example, suppose a change in A occurred. It would start running its first iteration. Before running R1, it would see that B had a remote alias. Thus it would send the Monte Carlo iteration to Model2, and request it run whatever relations were necessary to make B up to date. (To accomplish this, B's generated sample arrays would need to be accessible in both models, to both B and B_alias. This should not be a problem, as an alias and target generally share data.)

Model2 would see that B_alias is independent in its scope, thus it would give permission to Model1 to continue. Model1 would run R1, then proceed to R2. Again, it finds D has a remote alias so it would request Model2 update it. Model2 sees it needs to run R3 to accomplish this, so it runs it with the appropriate sample of B_alias. Permission would

be granted for Model1 to continue, and it would run R2 with the appropriate samples calculated from R1 and R3.

This may seem like a high volume of network requests and computation for each iteration, but the network speed experienced with DOME has been small enough that computation almost always dominates. Also, the Model1 simulation and Model2 simulation can be occurring in parallel as long as they see that no remote aliases or targets are required to continue.

Thus this last solution seems the most feasible and consistent with the DOME distributed model. Further research is needed to implement and test the feasibility and speed of this implementation.

4 MAPPING DESIGNER VIEWS TO PROBABILISTIC DISTRIBUTIONS

4.1 Background

An important problem in probabilistic design is generating density functions based on the available information. Density functions are most appropriately defined on sets of experimental data, but early in the design stage this is not available.

Four general methods are available to codify random variables into distributions (Siddall 1983):

1. Analyzing the data to yield a non-parametric or numerically defined distribution.

2. Fitting the available data to a standard analytical distribution.

3. Determining the maximum entropy distribution.

4. Subjective codification of judgment.

Methods one through three assume that at least some experimental data is available. This is problematic since early in the development process, there is not statistical data for the product's parameters as it has not been produced yet.

Method four is the most practical when little or no data is accessible. This method involves taking the experience and knowledge of the engineer and drawing upon it to arrive at density functions. One would need to use his experience of similar products and manufacturing processes to determine probabilistic information. This method also leads to difficulties as engineers are often not well trained in the area of probability.

4.2 Survey and Results

In order to better understand how engineers perceive probability, a survey was given to a diverse group of engineers. (The survey can be seen in full in appendix B.) Surveys were given to students of engineering, professors of engineering, automotive industry engineers, and electronics packaging engineers. The results for each group were similar except for some trends that will be mentioned later.

There were different categories of questions. One group was concerned with addressing typical probability problems that an engineer may encounter. The survey tried to force the engineer to record the graphic visualization of how he perceived the problem in his mind.

> A material from a supplier is specified as having a yield strength of 1,000 ± 50 psi. (950 psi minimum, 1050 psi maximum.) You are buying five pieces. You are creating an engineering model and need to guess at what those pieces yield strength will be.

The respondents could pick from a variety of answers (or choose none of the above). The answers were graphical representations of possible resulting distributions. A sample of answers is shown in Figure 4-1.

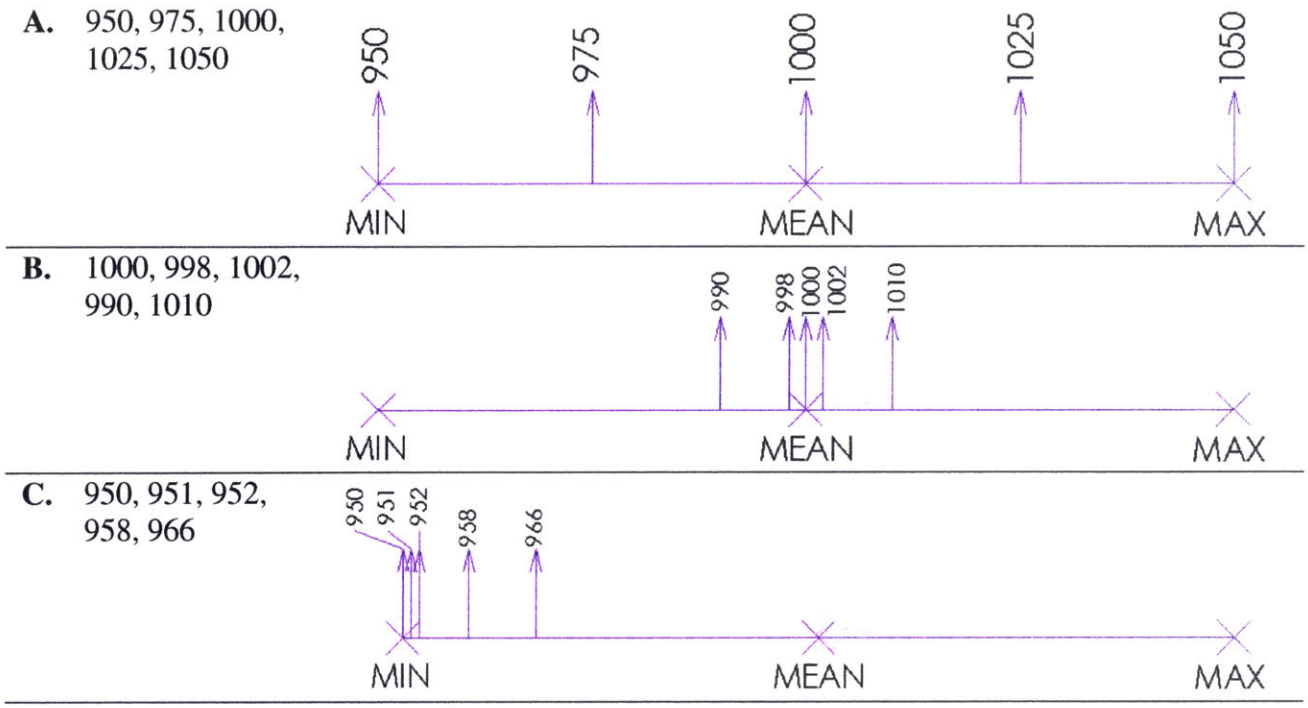


Figure 4-1[5] Sample of answers from survey

Another subset of questions asked the engineer to specify probabilities for tolerance ranges. This was meant to test how well they could create distributions based solely on their intuition. (See appendix B, question 3.)

A third subset gave the engineer the opportunity to sketch his own answers without being biased by the survey author's preconceptions. (See appendix B, question 5.) The following conclusions could be drawn from the answers supplied by those surveyed.

1. Engineers generally perceive probability within tolerance ranges as Gaussian.

Despite the fact that engineers usually specify parameter uncertainty in terms of tolerance ranges, they perceive the distributions within these ranges to be Gaussian. As a percentage of all the questions for all of those surveyed, 93% of answers resembled normal distributions. Of these, 85% were Gaussian centered at the midpoint of the

tolerance range. The other 7% were either uniform or some aggregation of Gaussian and uniform.

2. Engineers are poor at direct specification of probability distributions when no empirical data is available. This varies based on an engineer's background and discipline.

In response to the second question set, specifying probabilities to tolerance ranges, in general the respondents did very poorly. For the specification of the same normal distribution, the error in the standard deviation averaged 211% percent for all test takers. There was a large deviation in this error amongst the test takers as well. The smallest error was 18%, while the largest was 362%. The conclusion drawn was that although engineers perceive design parameters as probabilistic, they need tools to aid them in the creation of corresponding distributions.

Interestingly, the more experienced engineers working in industry performed better on this question. The two most experienced engineers surveyed achieved two of the smallest errors of all surveys, one of them being the 18% error. One possible reason for this is engineers deal with probability on a daily basis and get better at it with experience, even though they may not realize it.

3. Engineers with more industry experience perceive that most often a distribution's mean is not centered at the midpoint of the specification range.

Another interesting point was that the engineers working in industry (those not in the academic environment) gave more non-standard answers. The answers still resembled standard probability distributions, but they were more likely to be skewed to the left or right of the midpoint of the tolerance range. Often they were even concentrated at the endpoint of the range.

[5] Not all answers for the question are shown here. This is merely to illustrate the format of the survey answers. See appendix B, question 1 for a sample of a complete question and answers.

A number of possible reasons could be given for this trend. Those working in industry may not be as familiar with probability theory and thus understand less the tendency of large samples to be normal. Another possibility is that in industry often other parties control what will happen to their specified values. For example one engineer commented in the survey that the resulting distribution depends on which supplier he is dealing with.

Industry also deals more with the random nature of manufacturing on a daily basis. One electronics packaging engineer felt he could not even answer the questions without knowing more about the manufacturing process to be used.

In academics, problems tend to distill complexity of real situations to idealized representations. It is possible that this caused the answers received from the academics to be over-simplified.

4. Engineers experienced in industry feel that distributions were highly driven by cost.

The engineers in industry also felt cost was very relevant to their distributions. An automotive engineer commented that "the distributions will skew to whichever end of the spec yields the least expensive part". This is especially true in the automotive world where production volumes are very high.

5. Engineers feel making engineering tasks easier drives distributions.

It was also found that distributions skew to the direction that makes engineering tasks simpler. These tasks could be manufacturing or design related. One example would be the reuse of an existing product design that was not intended for the specification yet satisfies it with the expected value towards the endpoints of the specification range.

4.3 Application to DOME

Based on these conclusions, specifications for the tool were developed. As normal distributions were the principal answer given, they would be the center point of the tool. However uniform and beta distributions would also be available. The beta distributions

would provide the capability to skew to the left or right of the midpoint of the tolerance range.

Since it was found that engineers have difficulty specifying a distribution through the typical means, a tool must be provided that allows engineers to specify in terms of parameters they understand, namely percentages and tolerance ranges.

As many felt that the manufacturing process was critical, the tool should allow the user to build a distribution based on the manufacturing process to be used. Based on these surveys a tool that could provide these services would greatly increase the user's capability to add data quality information to his models.

Due to the lack of experimental data available, an adaptation of the method of subjective codification of judgment was used. Based on the survey, a graphical user interface was devised to help engineers create normal distributions. Three methods of creating a distribution were provided.

4.3.1 Direct Specification

First of all the engineer could create the distribution by explicitly specifying the defining parameters of a distribution, e.g. the mean and standard deviation for a normal distribution. This method would be used when the engineer already has empirical data on or experience with the parameter. For example, in Figure 4-2Figure 4- the engineer specified a mean of 10.0 and a standard deviation of 1.0.

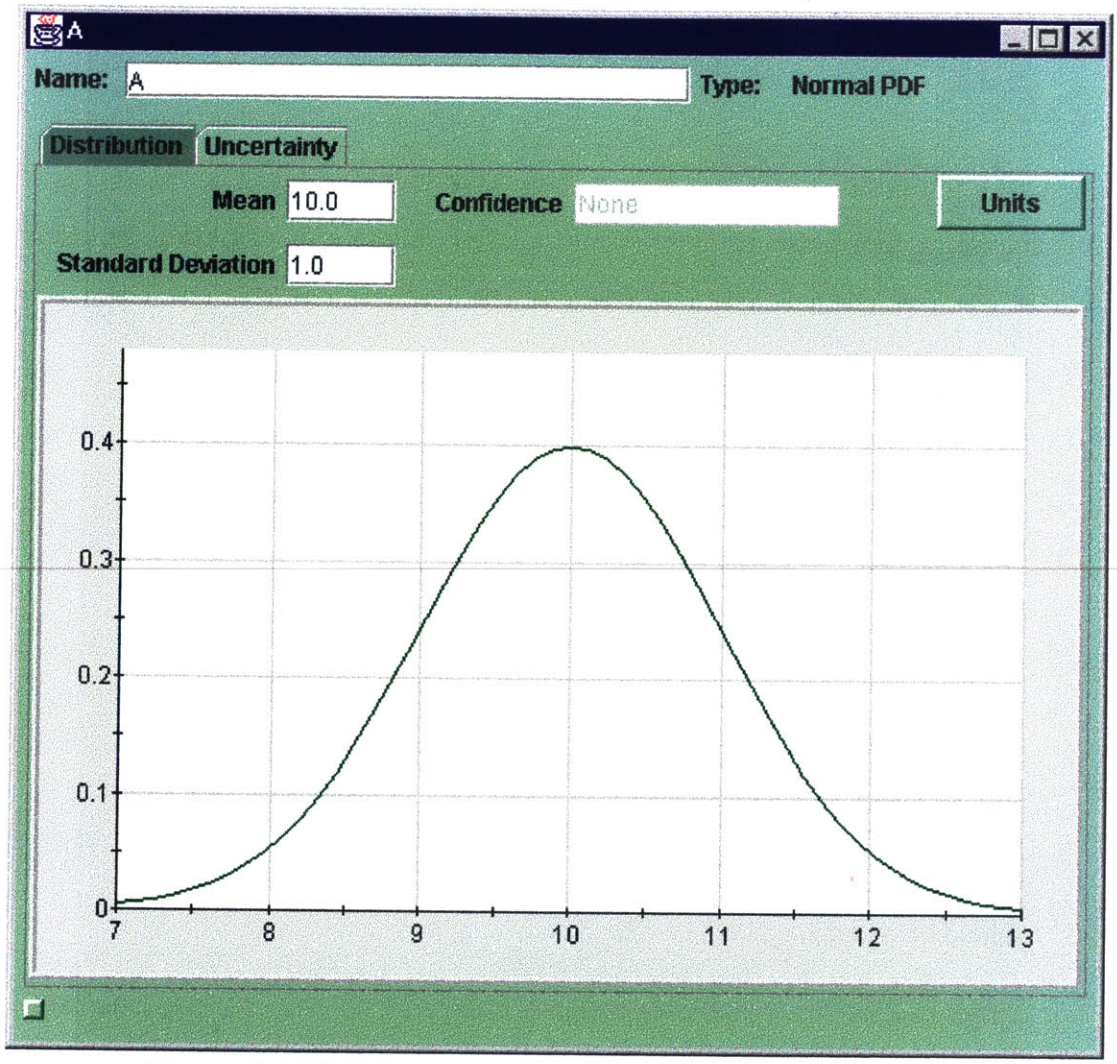


Figure 4-2 DOME normal probability density function GUI

4.3.2 Qualitative

The second method allowed the engineer to state his intuitive confidence in terms of percentages and tolerances. This was based on subjective distribution specification as explained in section 4.1. In Figure 4-3 the engineer specified that he was 90% confident that the parameter would fall within +/- 1.0 of the mean.

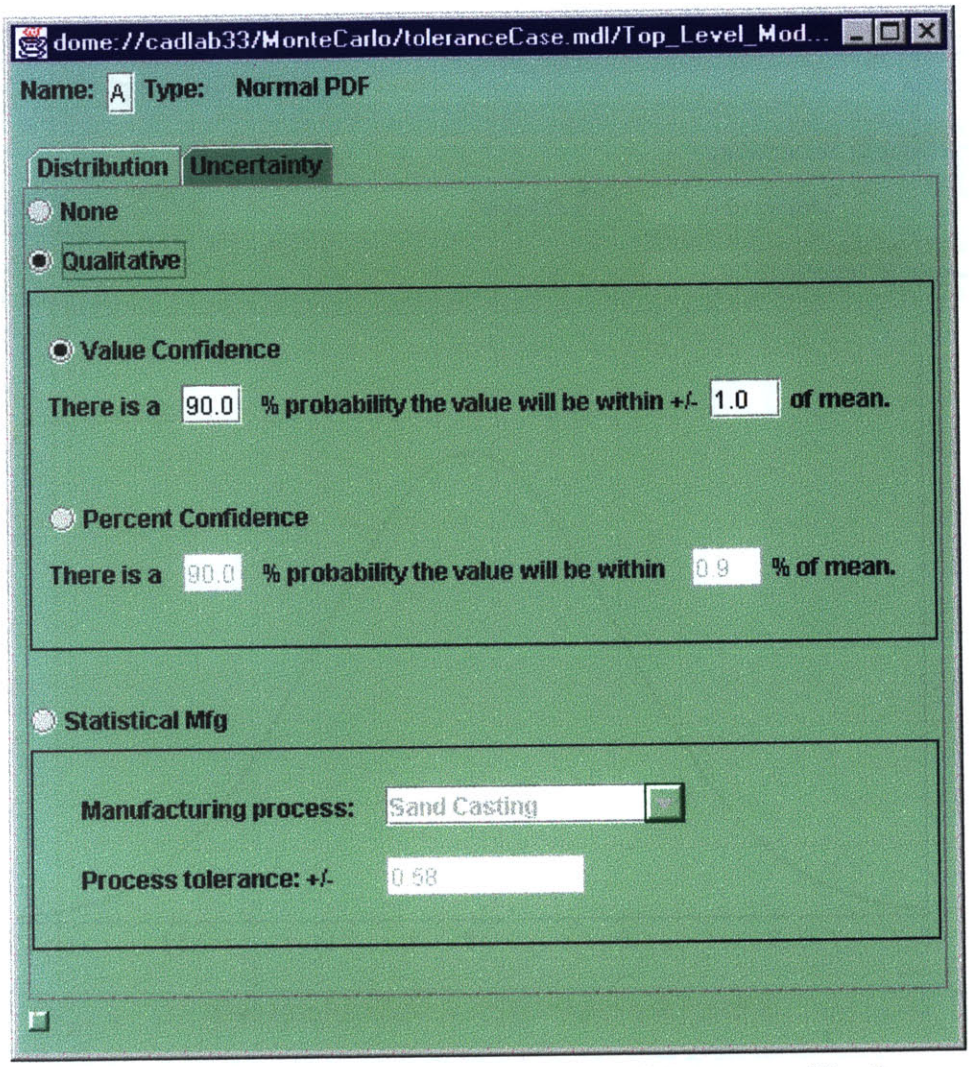


Figure 4-3 DOME GUI for qualitative confidence specification

This change affected through the GUI starts an algorithm that determines the standard deviation based on these specifications. The solution is defined as the standard deviation that results in the integral of the probability density function over the tolerance range equaling the specified confidence. For example, in this case the algorithm finds the standard deviation such that 90% of the area under the curve lies between 9.0 and 11.0. The results of the algorithm are shown in the following figure.

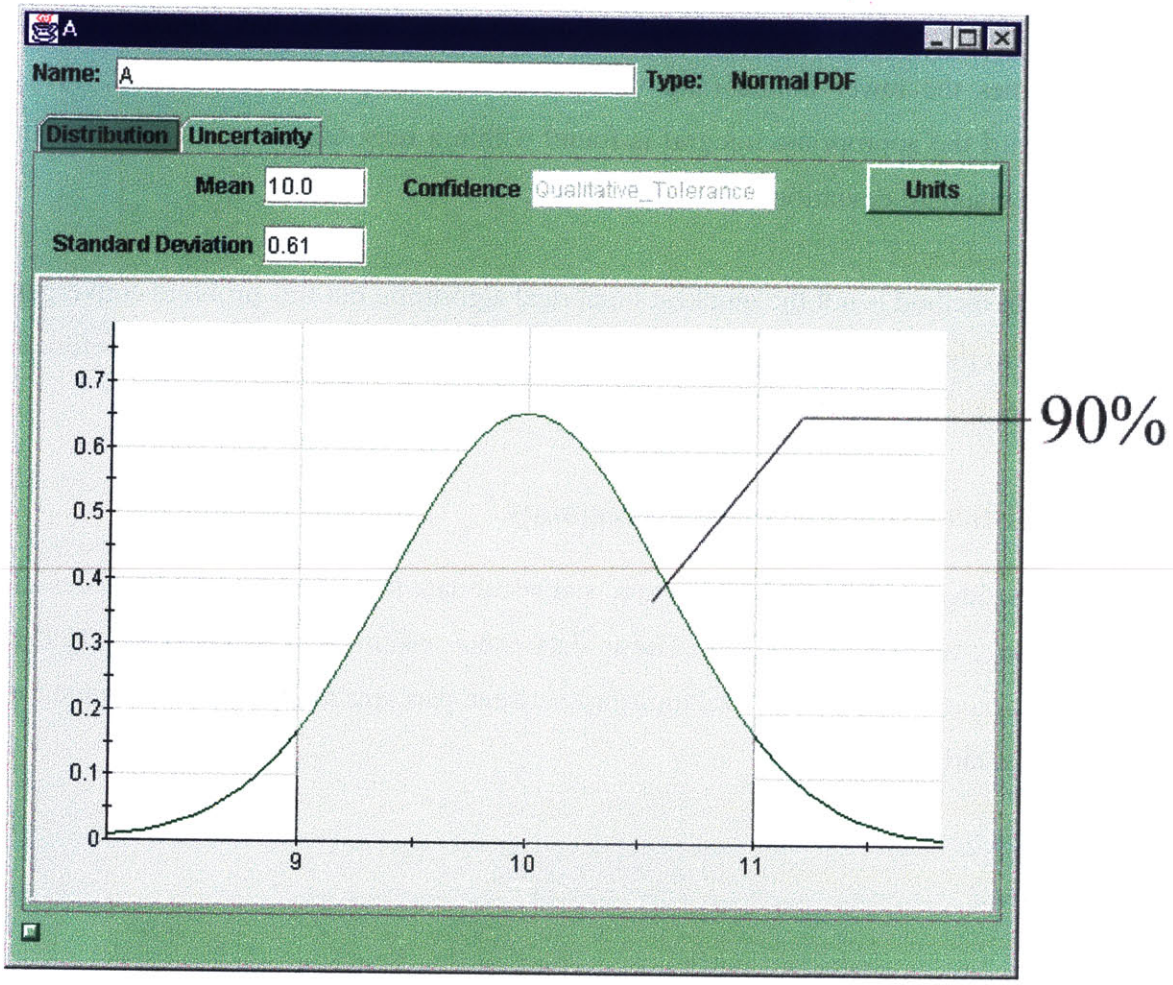


Figure 4-4 Qualitative specification result

This was accomplished through the bisection method (Chapra *et al* 1988). Two guesses were made for the standard deviation, one being an upper bound and the other being a lower bound. It is verified that the solution is in between these. This is done by finding the integral from 9.0 to 11.0 based on both standard deviations and then checking that one integral is greater than 0.90 and one integral is less than 0.90. This confirms that the solution lies in between the two bounds. If it does not, the upper bound is increased and the lower bound is decreased until the bounds are sufficient.

Once appropriate upper and lower bounds are found, they are bisected. A guess for the standard deviation is made that is directly in between the upper and lower guesses. The new integral is calculated for this guess. It is determined from this integral whether the solution is in between the upper bound and the new guess, or the lower bound and the

new guess. If the solution is in between the upper bound and the new guess, the new guess becomes the lower bound (and vice versa) and the process is repeated. This continues until the appropriate integral is found within a reasonable error. Note that in Figure 4-4 a standard deviation of 0.61 was found.

The bisection method is not the quickest numerical algorithm, but this problem converges quickly due to the simple nature of the normal probability density function. Therefore it was sufficient for this problem.

4.3.3 Statistical Manufacturing Uncertainty

The final confidence method provided uses statistical data from manufacturing processes to determine standard deviation. These data were taken from Kalpakjian (1995). Kalpakjian derived a logarithmic relationship between part size and process tolerance for most manufacturing processes.

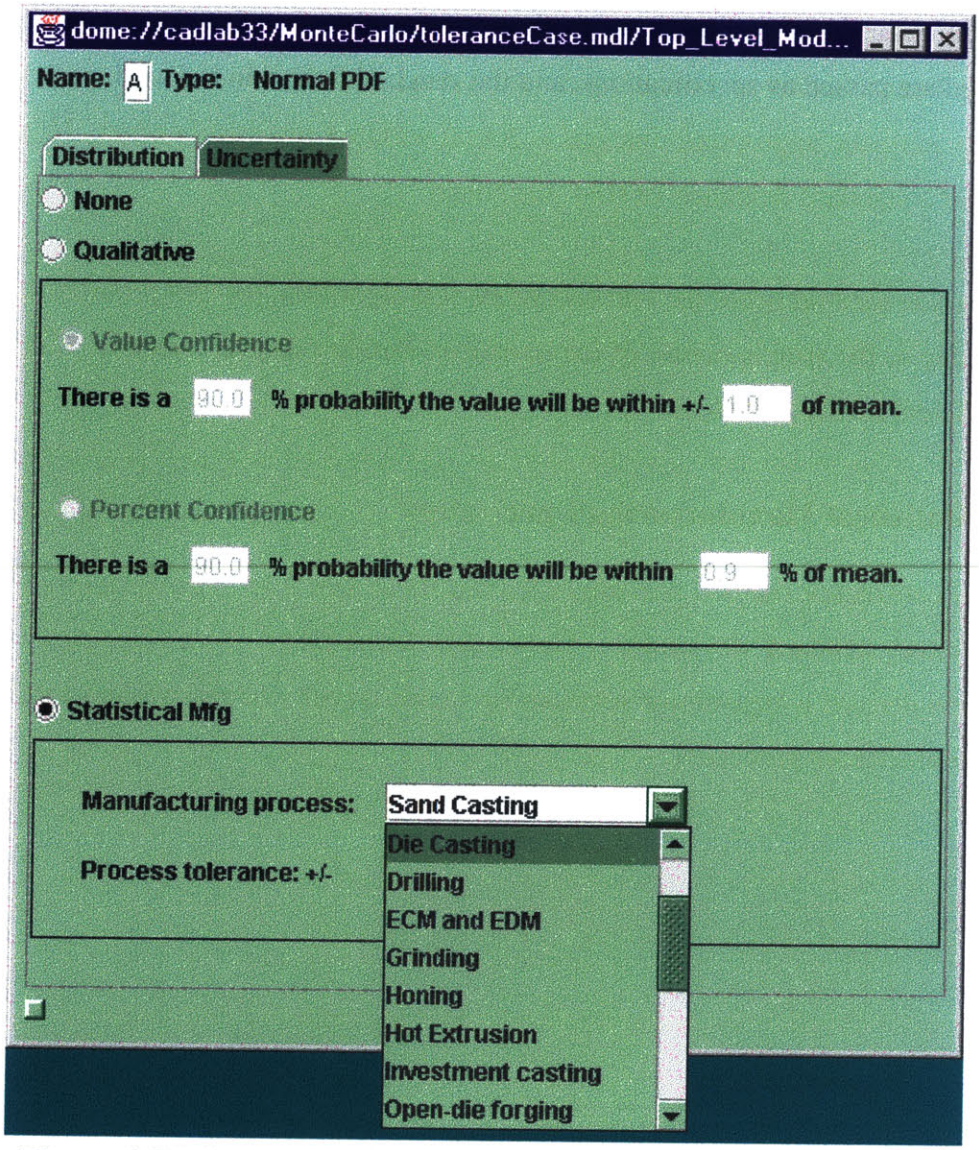


Figure 4-5 DOME GUI for manufacturing confidence specification

The relationship constants were read from a file that lists the manufacturing processes, their logarithmic relationship slope (m) and their logarithmic intercept (b). The tolerance could then be calculated by equation (4.1):

$$\log_{10}(tolerance) = m \log_{10}(dimension) + b \qquad 4.1$$

The algorithm references this file and gets the slope and intercept based on the process chosen. Processes could easily be added to this file using the same format. Thus the process tolerance is determined. Using this tolerance the same bisection algorithm is

called that is used in the previous method. A confidence of 99% and the tolerance calculated are passed as specifications, and the standard deviation is found.

5 CONFIDENCE TOOL APPLIED TO A SIMPLE ENGINEERING MODEL

5.1 Model and engineering problem description

A very common application of probabilistic engineering tools is in tolerance analyses. Parts and assemblies can have many tolerances aggregating to result in overall or interfacing dimensions with very large tolerance ranges. Determining the probability of being towards the endpoints of the aggregated tolerance range can help avoid over-constraining tolerance ranges.

The problem chosen for the case study is one of assembly (adapted from Shigley *et al* 1989).

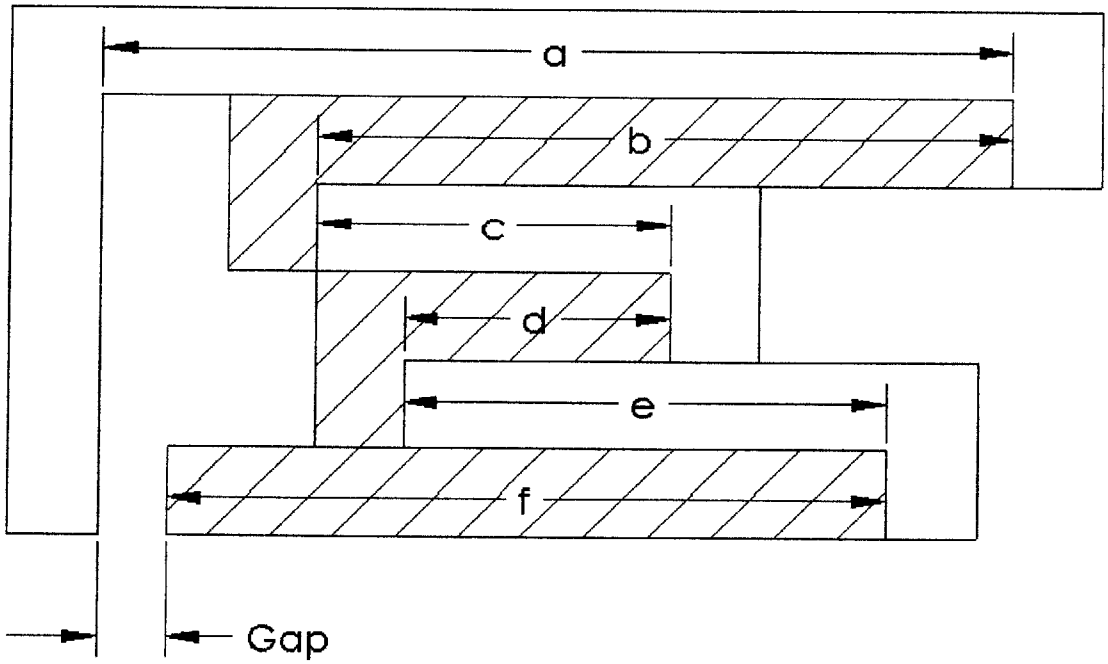


Figure 5-1[6] Assembly tolerance problem

The figure shows six parts that will mate in assembly. Dimensions a through f affect the gap dimension that should be greater than zero to allow assembly.

[6] Adapted from Mechanical Engineering Design, Shigley *et al* (1989)

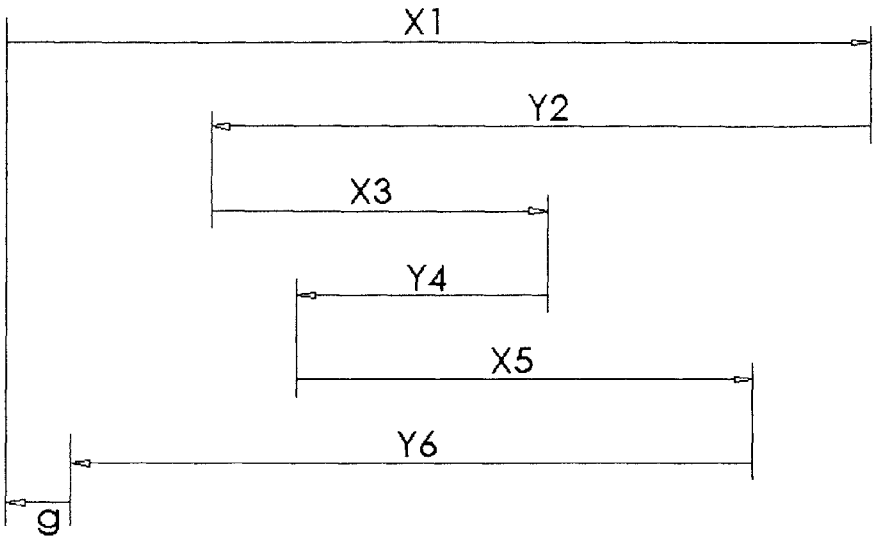


Figure 5-2 Assembly dimensions shown as vectors

Figure 5- shows how certain dimensions drive the final gap (here shown as g) to the left and some drive it to the right. The final nominal value of the gap is the sum of the right tending dimensions minus the sum of the left tending dimensions as seen in 5.1:

$$\bar{g} = \sum \bar{x} - \sum \bar{y} \qquad 5.1$$

Three models were created using some of the different methods previously described. A worst case analysis, three sigma analysis, and an analysis using the DMCT were done. A conventional Monte Carlo simulation was not done as the relative benchmarking data was easily approximated.

The cost of producing the assembly could be simplified as shown in 5.2:

$$\text{cost/year} = (a)(\$1/\text{part})(n \text{ parts/year}) \qquad 5.2$$

Parts per year was given as 1,000,000. Thus the cost could be simplified to 10^6*a. The objective was to minimize the cost by minimizing dimension a. The dimensions were given as follows.

Table 5-1 Dimension names, values and tolerances

Dimension	Mean	Tolerance
a	10	±0.7
b	8	±0.6
c	5	±0.1
d	2	±0.1
e	6	±0.2
f	7	±0.9

Thus the only dimension to be changed was a.

The nominal values analysis is done using a simple spreadsheet shown in Figure 5-.

Right tending	
a	10
c	5
e	6
Total, right	**21**
Left tending	
b	8
d	2
f	7
Total, left	**17**
Mean, clearance	**4**
Cost	**$ 10,000,000**

Figure 5-3 Nominal analysis

The worst case analysis was done as a spreadsheet. It took an engineer knowledgeable in the spreadsheet program 7 minutes to build.

	Mean	tolerance, upper range	tolerance, lower range
Right tending			
a	8.7	0.7	0.7
c	5	0.1	0.1
e	6	0.2	0.2
Total, right	19.7	1	1
Worst case; smallest possible	18.7		
Left tending			
b	8	0.6	0.6
d	2	0.1	0.1
f	7	0.9	0.9
Total, left	17	1.6	1.6
Worst case; largest possible	18.6		
Mean, clearance	2.7		
worst case clearance	0.1		
Cost	**$ 8,700,000**		

Figure 5-4 Worst case analysis

Note that the modeler left a clearance of 0.1, even though worst case was assumed for all parameters. This is typical of many worst case models. Although the likelihood of worst case actually occurring is zero or near zero, the engineer does not feel comfortable leaving a clearance of 0, so he over-compensates again.

The three sigma analysis was also done as a spreadsheet. It took 10.4 minutes to build, also by an engineer knowledgeable in the spreadsheet program used.

	Nominal	Tolerance	(Tol)^2
		[+/-]	
a	7.84	0.7	0.490000
b	-8	0.6	0.360000
c	5	0.2	0.040000
d	-2	0.1	0.010000
e	6	0.1	0.010000
f	-7	0.9	0.810000
	Average	0.433333333	
Interference	1.84	2.6	
			1.720000
			0.286667
			0.535412613
RMS 3 sigma tolerance Analysis		Tolerance	Clearance
	Sigma 1	0.968745947	0.871254053
	Sigma 2	1.40207928	0.43792072
	Sigma 3	1.835412613	0.004587387
Cost	**$ 7,840,000**		

Figure 5-5 Three sigma analysis

The analysis using the DMCT was done in the DOME environment. It took 10 minutes to build by a user proficient in DOME[7].

[7] For one who had never before used DOME, it would take a few minutes (more or less depending on the user) to gain familiarity with how to add objects and the functions supported. At that point he would also be able to build the simulation in 10 minutes.

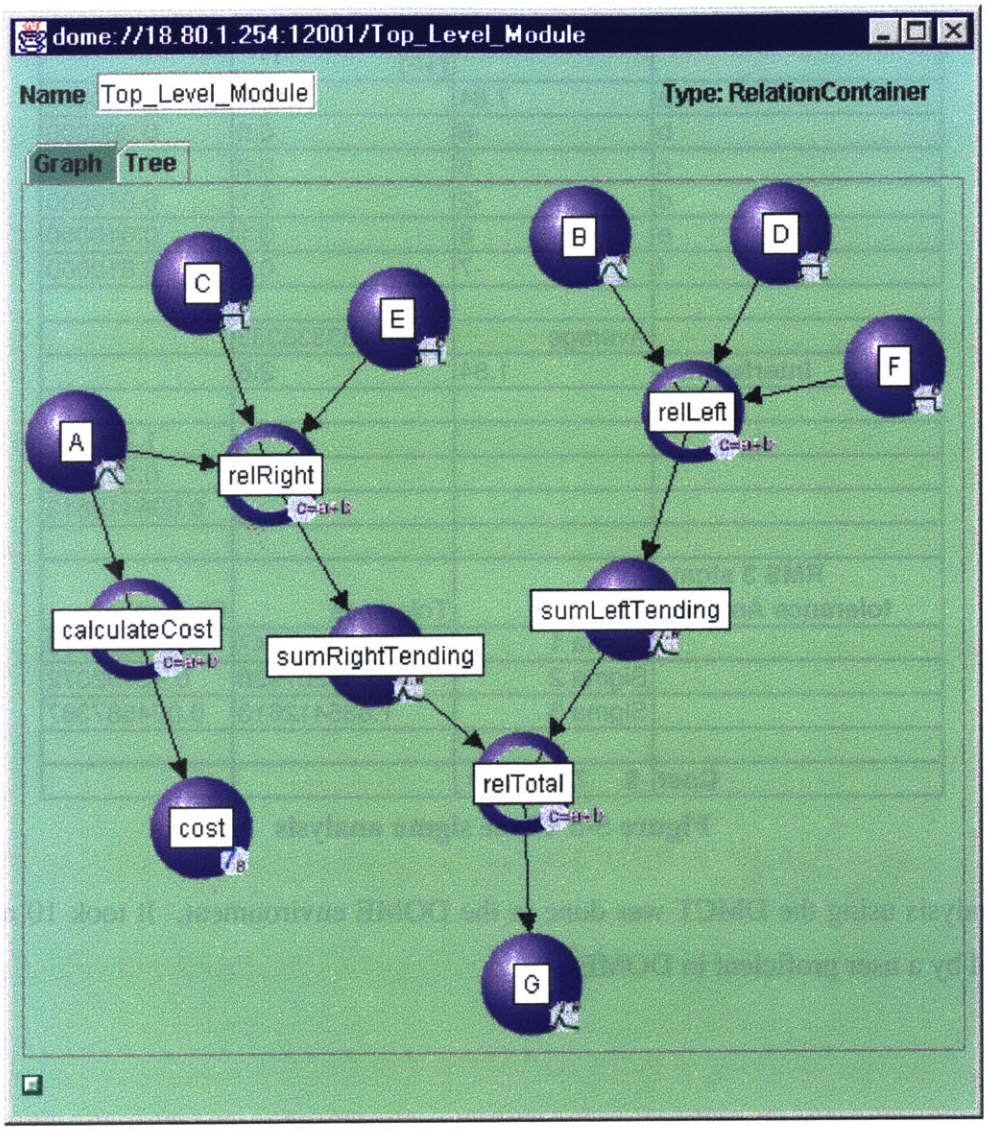


Figure 5-6 DOME DMCT analysis

Note that the causal visualization in this environment is superior to the spreadsheets. The arrows show the direction of dependency as defined by the user in relations.

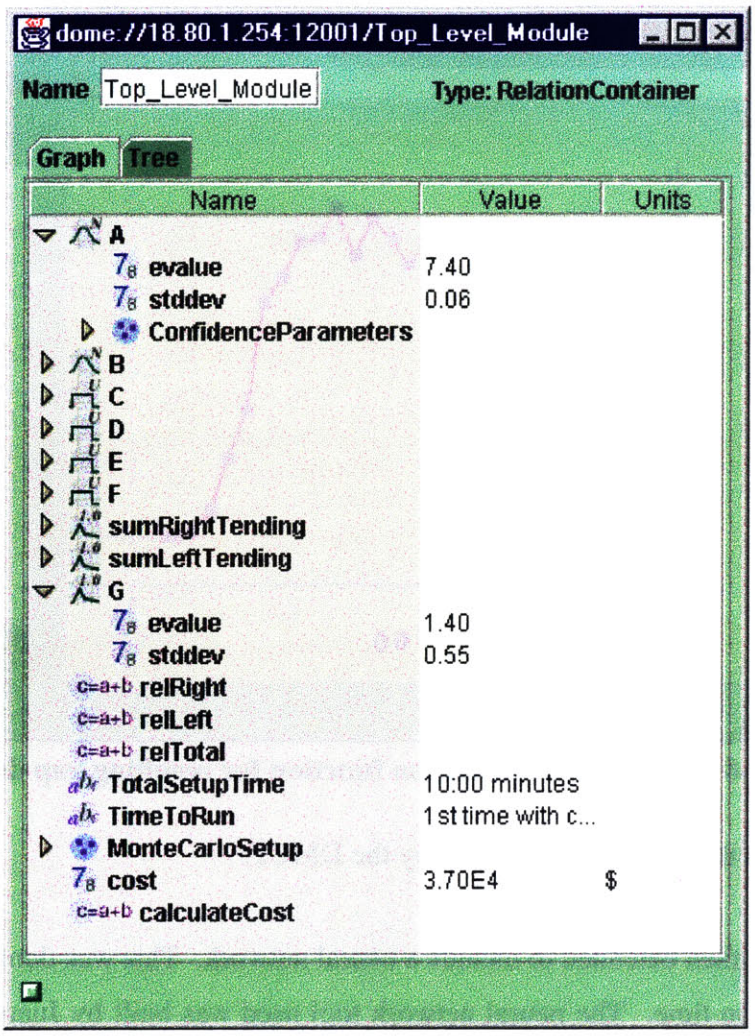


Figure 5-7 DOME tree view

The DOME view shown in Figure 5- (called tree view) is convenient to view the results and inputs at a glance.

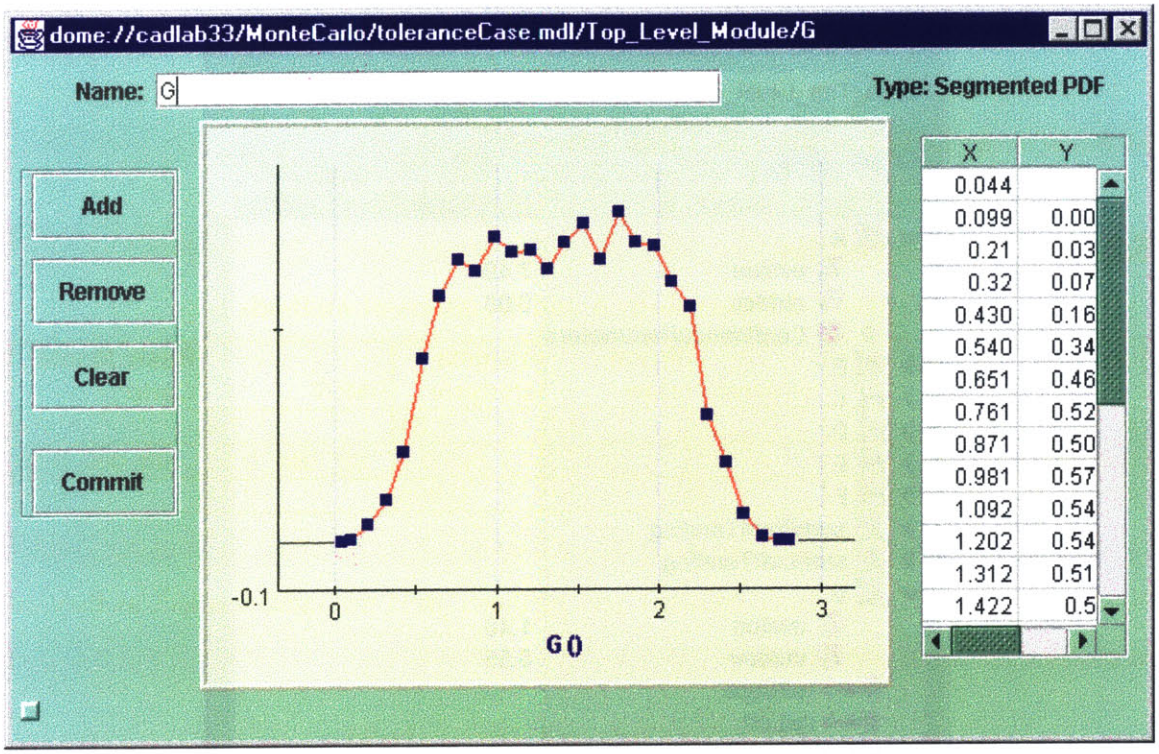


Figure 5-8 Probability distribution function for resulting gap dimension

Figure 5- shows the gap result as solved by the DMCT.

This model was then extended to include a neural network. This was done to improve the model calculation time. The neural network tool used was built by Juan Deniz and was based on his work (Deniz 2000). It took less than a minute to add and set up the neural network with the mean and standard deviation of a as inputs and the gap minimum as the output.

After the network was set up it needed to collect data sets for the inputs and output and then train based on these exemplars. For this model five exemplars is sufficient. The network runs the model automatically, generating these exemplars based on the parameters provided by the user. This process took 10 minutes (five exemplars at two minutes each). Thus the total setup time was 20 minutes (10 for the DMCT model plus 10 for the neural network setup).

After the network was trained, the model run time was almost instantaneous (see Table 5-3, run time for DMCT with neural net). The results for the neural network were nearly identical to the DMCT results (see Table 5-3).

5.2 Confidence results

Table 5-2 Dimensions A and G, different methods

	Nominal	Worst case	3 sigma analysis	DMCT	Basic Monte Carlo
A	10	8.7	7.84	7.4	7.4
tolerance	+/- 0.7	+/- 0.7	+/- 0.7	std. dev. 0.55	std. dev. 0.55
minimum clearance, G	4	0.1	0.005	0.04	0.04

Note that the DMCT and basic Monte Carlo methods yield the same results[8], as they are in essence the same simulation.

Table 5-3 Costs and Time to build

	Nom-inal	Worst case	3 sigma analysis	DMCT	Basic Monte Carlo	DMCT w/ neural net
cost/year (in millions)	$ 10.0	$ 8.70	$ 7.84	$ 7.40	$ 7.40	$ 7.40
DMCT Tool Cost Comparison	$ 2.60	$ 1.30	$ 0.44	0	0	0
Cost savings (in millions)	-	$ 1.30	$ 2.16	$ 2.60	$ 2.60	$ 2.60
Time to Build (minutes)	5	7	10.42	10	120	20
Time to Run (minutes)	0.02	0.02	0.02	2	2	0.02

The cost savings incurred from using a Monte Carlo simulation are shown in Table 5-3. Cost savings are 100% over the worst case method, and about 9% over the 3 sigma analysis.

Comparing the basic Monte Carlo to the DMCT, the cost savings are the same as the cost was simplified to be purely driven by manufacturing cost. However, Monte Carlo development time is 12 times that of the DMCT, so savings would come in the design stage.

[8] Due to the random nature of a Monte Carlo simulation, the results may vary each time it runs. This is still the case for the DMCT. However if enough iterations are executed, the differences are negligible.

The cost savings are also identical comparing the DMCT to the DMCT with the neural network. The difference is the increased model build time, and nearly instantaneous run time for the DMCT with the neural network.

5.3 Assessment of usefulness

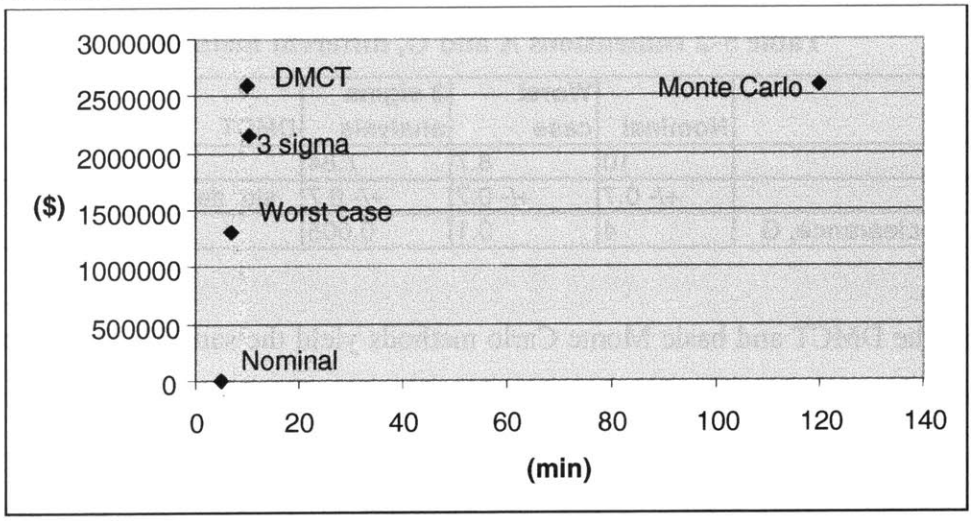


Figure 5-9 Development time vs. cost savings

This figure shows graphically the results from section 5.2. Note that on the time axis the DMCT is far superior to the Monte Carlo, but on the cost axis they are identical.

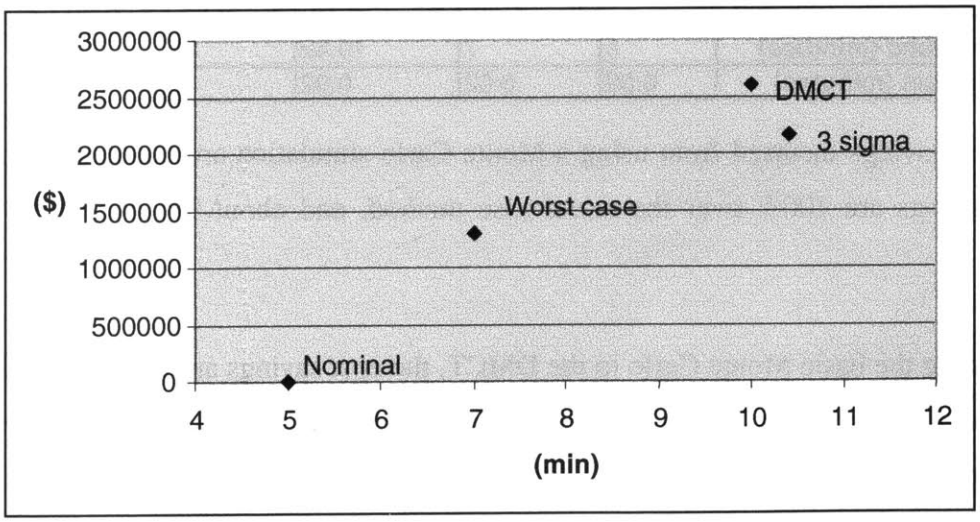


Figure 5-10 Development time vs. cost savings, conventional Monte Carlo not shown

In this figure the basic Monte Carlo is removed to focus on the other tools.

A more difficult parameter to measure is the level of expertise necessary for implementation. To implement a basic Monte Carlo simulation, the developer must be familiar with probability theory and the idiosyncrasies of Monte Carlo simulations. For a three sigma analysis, the model builder again must know probability theory. For a worst case analysis, the modeler must make guesses at the parameters that will result in a poor design. As previously discussed this is difficult for large and complex models.

The DMCT requires no additional expertise other than that required to create the nominal model. Only the basic mathematical relationships between the model parameters need to be known. Also, no special knowledge in probability theory is necessary.

Combining the use of neural networks with the DMCT allows a user to get instantaneous model approximations based on the trained network. The neural networks make optimization methods possible on probabilistic models.

6 CONCLUSION

6.1 Summary

This thesis presented the need for assessing designer confidence in an environment that is increasingly model-oriented and distributed. Two main barriers to accomplishing this were presented as creating probability distributions based on designers' perceptions, and the propagation of said distributions in a dynamic and distributed model building architecture.

Current methods and research from these two areas were presented. It was shown that mapping designer views of confidence to distributions was a problem best addressed by subjective codification of judgment. A graphical user interface was presented that allowed specification of distributions in terms familiar to designers and engineers.

Methods to propagate probability distributions were presented. The three methods focused on were worst-case analysis, three sigma analysis and conventional Monte Carlo simulation. As these methods increase in accuracy, they also increase in complexity and implementation time. The Dynamic Monte Carlo Tool was presented as an alternative to a conventional Monte Carlo simulation. This tool manages the complexity automatically, thus giving the accuracy and robustness of a Monte Carlo Simulation without the implementation difficulties.

A case study benchmarking the different methods was shown. Time to build the models as well as cost of the resulting designs were measured. The DMCT resulted in significant reduction in time over a conventional Monte Carlo simulation and significant manufacturing cost savings over the other methods.

6.2 Future work

6.2.1 Probabilistic collocation integrated with the DMCT

Although the DMCT significantly improves model implementation time, model run time is not (as compared to a conventional Monte Carlo simulation). If the model does not require outside program simulations (see section 3.2.1.2) then the run time is sufficiently short. For the case study in section 5, the run time was 0.2 minutes for the worst case and three sigma analyses and two minutes for the DMCT and Monte Carlo analyses. A model with a two-minute run time is useable, but iterative model searches and optimizations become tedious or impossible.

For this reason the tool was used in collaboration with surrogate modeling techniques, specifically neural networks. Another technique that should be benchmarked would be probabilistic collocation methods (section 3.1 and Webster 1996). Similar to neural networks, this method would combine the ease-of-use of the DMCT with the simulation speed of the approximation method.

This could most likely be done in an automated fashion, as polynomial approximation is a fairly standard iterative procedure. The model would run a few times to get a first order approximation, then depending on the accuracy of the approximation it would increase the approximation order or run the model using the polynomial.

Once this has been implemented, the accuracy and speed of probabilistic collocation used with the DMCT could be compared to that of neural networks.

6.2.2 DMCT in a distributed model

The DMCT was designed to be extendable to a distributed architecture, although the full implementation is outside the scope of this thesis. Further research could draw on section 3.2.2 to complete that implementation.

APPENDIX A: GRAPH ALGORITHMS

Elementary graph algorithms were used to determine model structure and relation order.

A.1 Building the Adjacency List

Figure A-1 outlines the procedure of building a model that represents the model structure (known as an adjacency list).

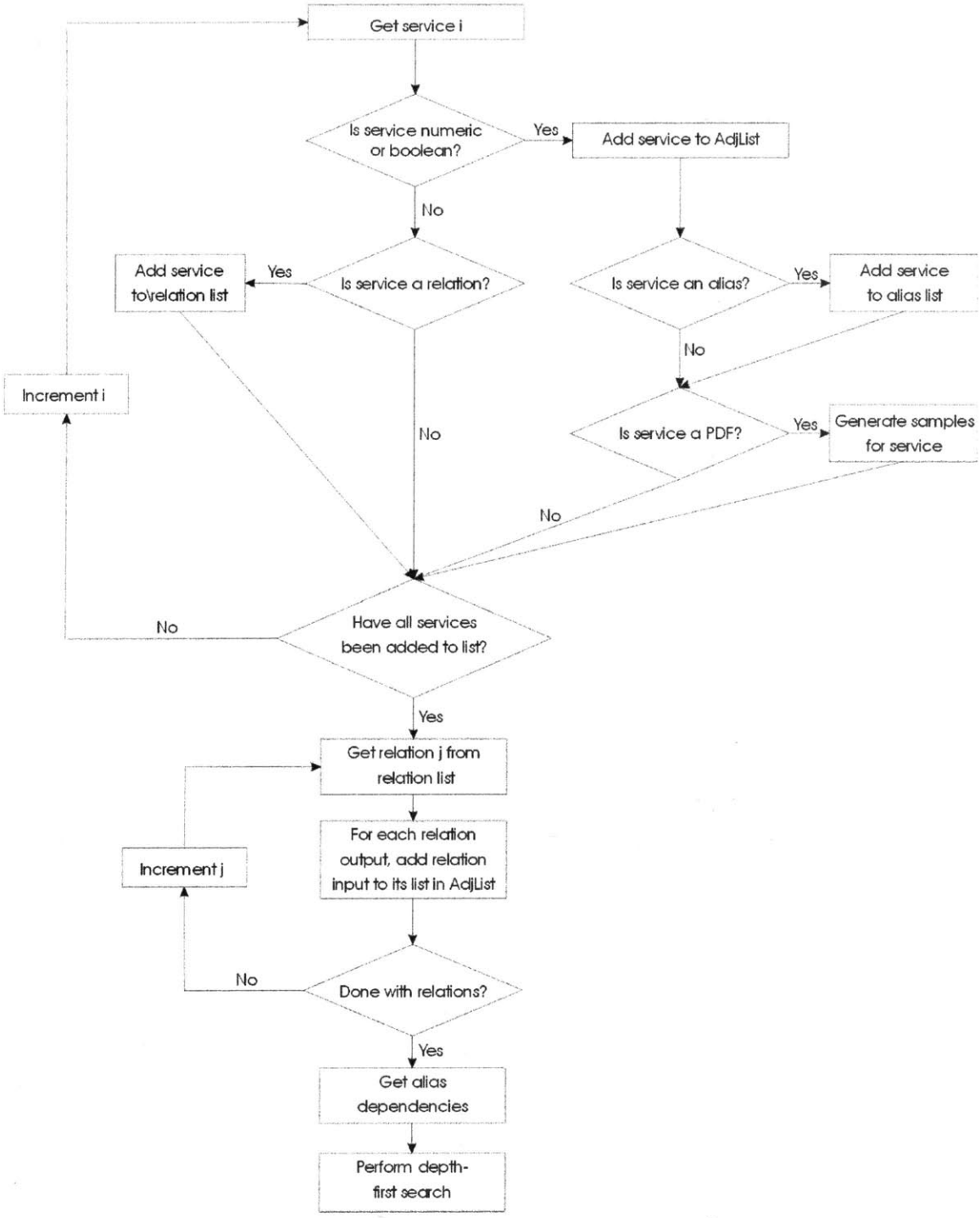


Figure A-1 Building the adjacency list[9]

Take the model example previously shown:

[9] The figure refers to the adjacency list as "AdjList".

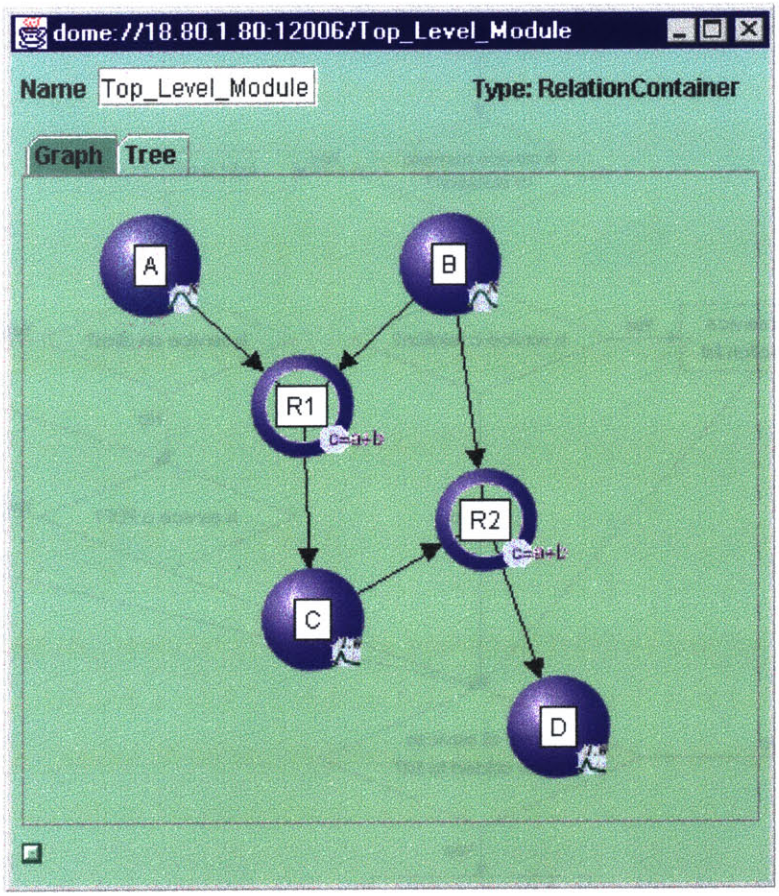


Figure A-2 Example model

The DMCT begins searching the model at the top level to load all of the contained services. It checks to see if they are of numeric or boolean type, and if they are it loads them into an adjacency list (a 2 dimensional array). If they are of relation type, it puts those into a list as well. If the numeric is an alias, it adds it to a list of aliases (to be discussed later). If the numeric is a PDF, it generates samples for it. For the above model, after getting all services, the adjacency list appears as in Table A-1.

Table A-1

A	B	C	D

Once all the model services have been added, it steps through the relation list. It steps through each output for every relation. It finds the output in the adjacency list, then adds all inputs of the relation to the column associated with the output. For example, in the above model relation "R1" is defined as "C = A + B". Also stored (for reasons that will

be apparent later) is the relation associated with the input. After searching the relation's inputs and outputs, the adjacency list would look as follows.

Table A-2

A	B	C	D
		A: R1	
		B: R1	

The next relation is "R2", which is defined as "D = B + C". After searching this relation:

Table A-3

A	B	C	D
		A: R1	B: R2
		B: R1	C: R2

The adjacency list now has all relationships as defined in relations. At this point the alias list is used to find alias dependencies. (See section 3.2.2 for a description of aliases and targets.) If a node is an alias, its target is added as one of its inputs. Also, the alias is added to the list of the target's inputs, as alias dependencies are bi-directional[10].

After alias dependencies have been found the adjacency list is complete. The model relationships are completely represented by this structure.

A.2 Depth-First Search

Next the model is searched using the depth-first search to determine the correct way to solve the model. The depth first search is a way of "visiting" every node in the system and checking each relationship systematically. This algorithm can be thought of as a way of "climbing the tree" to the top, and then coming back down, checking the relationships as you go.

For the example shown in Figure A-2, the algorithm starts with A in the adjacency list. It sees that there are no "branches" above it. In other words, there are no dependencies in the adjacency list. So the next node is checked. The search continues in this fashion

[10] Bi-directionality here refers to the fact that both the alias and the target can affect changes to the other. Neither module is a master.

until C, where branches that are higher (inputs) are found. So the search climbs up their (A's and B's) branches. Since A and B have already been visited, it climbs back down the tree, adding the associated relations as it goes. So R1 would be added at this point. In the same manner, it climbs up D's branches and adds R2 on the way back down.

Climbing the branches and adding relations in this manner ensures that the relations will be solved in a consistent and correct way.

APPENDIX B: SURVEY

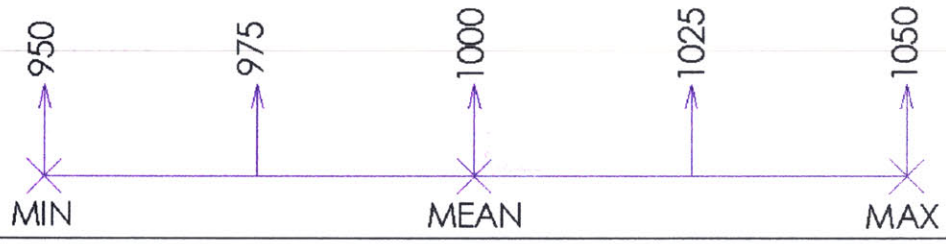
Thank you for your time. There are no right answers. Answer these according to your experience.

Question 1:

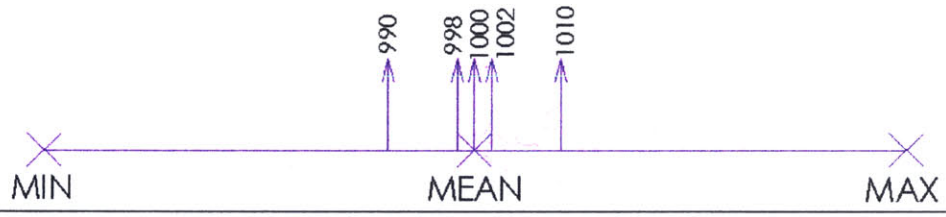
A material from a supplier is specified as having a yield strength of 1,000 ± 50 psi. (950 psi minimum, 1050 psi maximum.) You are buying five pieces. You are creating an engineering model and need to guess at what those pieces might come in at.
Circle the letter corresponding to the list of values that you think they will most likely be:

Note: The graphs in the survey are for visualization of the numbers listed relative to the minimum, mean, and maximum of the tolerance range.

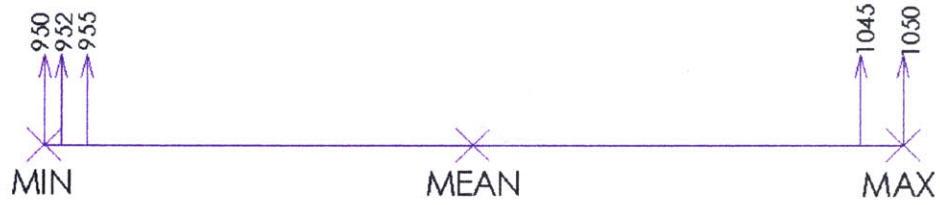
A. 950, 975, 1000, 1025, 1050



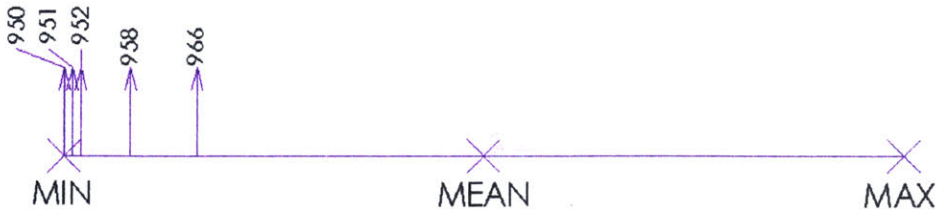
B. 1000, 998, 1002, 990, 1010



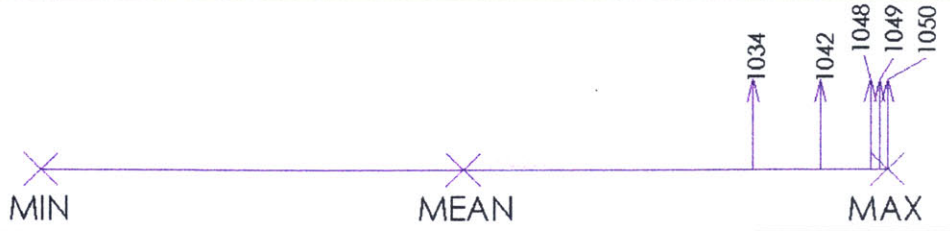
C. 950, 952, 955, 1045, 1050



D. 950, 951, 952, 958, 966

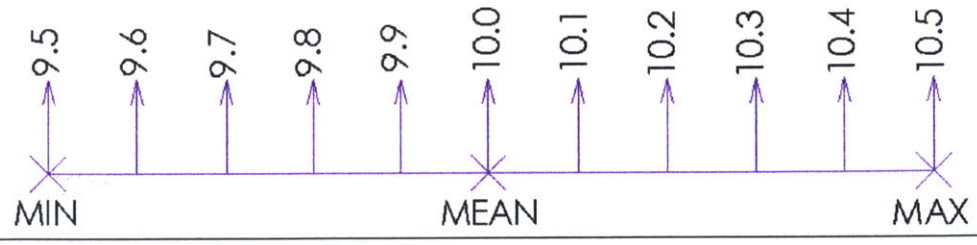


E. 1034, 1042, 1048, 1049, 1050

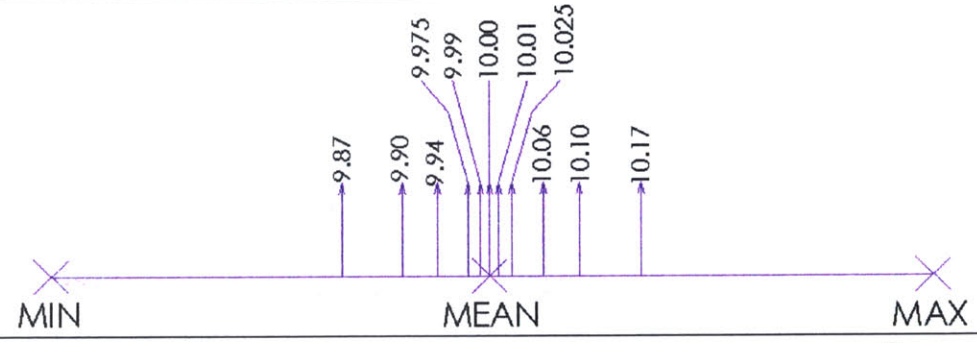


Question 2:
You design a piece to be 10 ± 0.5 thick (9.5 min, 10.5 max). You make eleven. Circle the letter corresponding to the list that you think the pieces will most likely come in as.

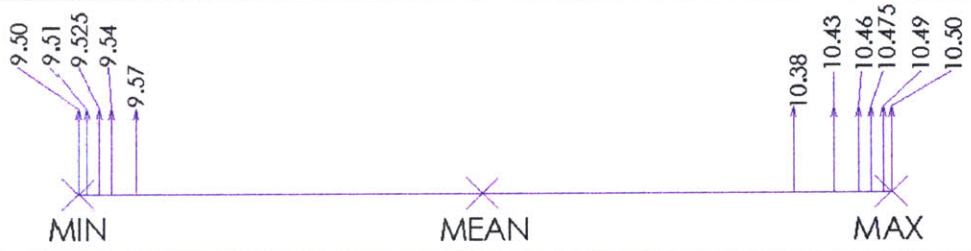
A. 9.5, 9.6, 9.7, 9.8, 9.9, 10.0, 10.1, 10.2, 10.3, 10.4, 10.5



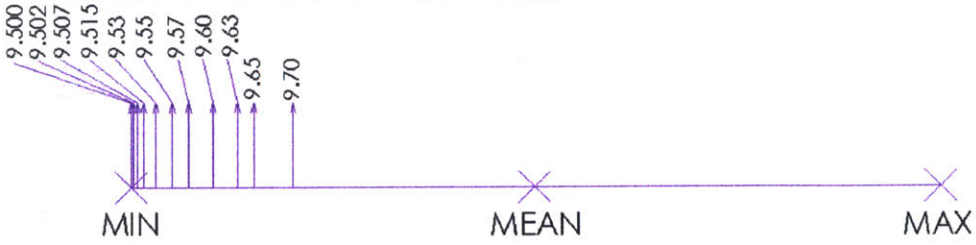
B. 9.87, 9.90, 9.94, 9.975, 9.99, 10.0, 10.01, 10.025, 10.06, 10.10, 10.17



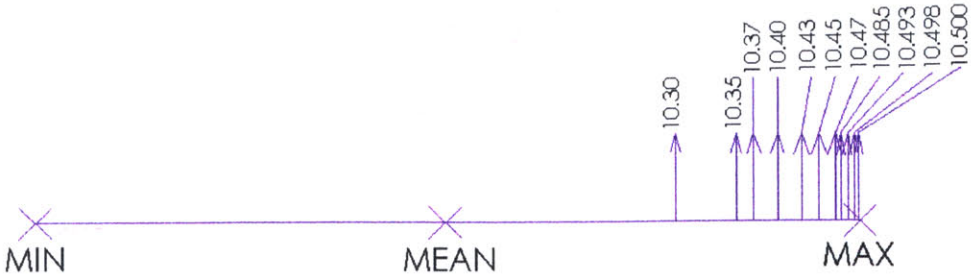
C. 9.50, 9.51, 9.525, 9.54, 9.57, 10.38, 10.43, 10.46, 10.475, 10.49, 10.50



D. 9.500, 9.502, 9.507, 9.515, 9.53, 9.55, 9.57, 9.60, 9.63, 9.65, 9.70

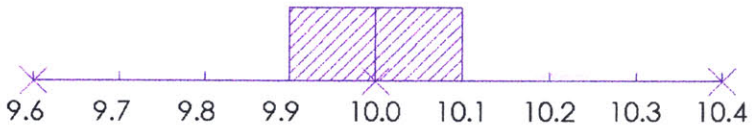
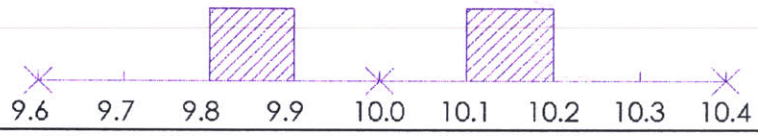
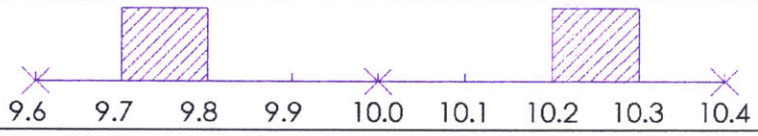
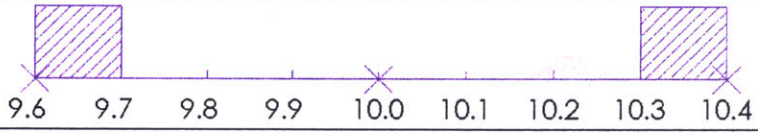


E. 10.30, 10.35, 10.37, 10.40, 10.43, 10.45, 10.47, 10.485, 10.493, 10.498, 10.500



Question 3:
You design a piece to be 10 ± 0.4 thick (9.6 min, 10.4 max). You make 1000.

Estimate what percentages would fall in the ranges listed below:
Note: It's OK to put 0%. Percentages should add up to 100%. Write answer in far right column.

		Write percentage
Between 9.9 and 10.1		
Between 9.8 and 9.9, 10.1 and 10.2		
Between 9.7 and 9.8, 10.2 and 10.3		
Between 9.6 and 9.7, 10.3 and 10.4		
Outside of tolerance		

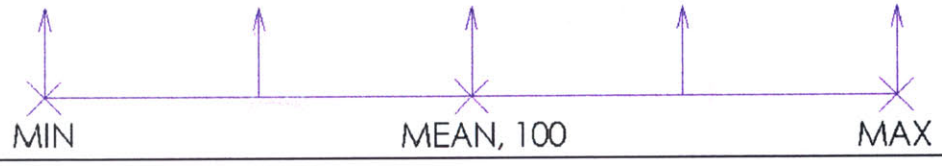
Question 4a:
You are designing a part. You want it as thick as possible for strength reasons, but you're constrained by the systems engineer who will tell you how much space you have. He tells you there will be a minimum of 90 mm and a maximum of 110 mm. You need to create your engineering model now, though.

Make some guesses at what it will come in as. Circle your answer.

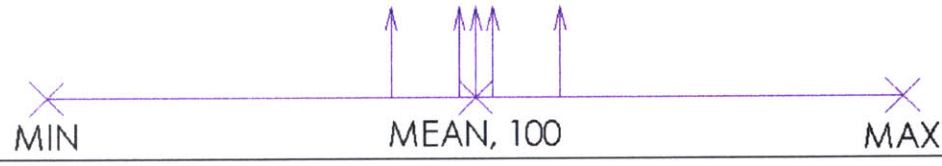
A. 90% chance in 99 to 101
B. 90% chance in 95 to 105
C. 90% chance in 90 to 110

Question 4b:
Given the 90% range you specified in question 4a. How will that 90% be distributed within the maximum and minimum? (Circle your answer)

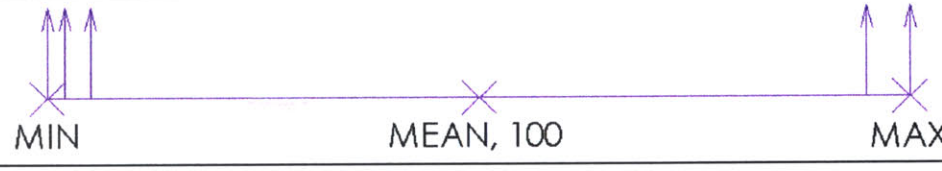
A.



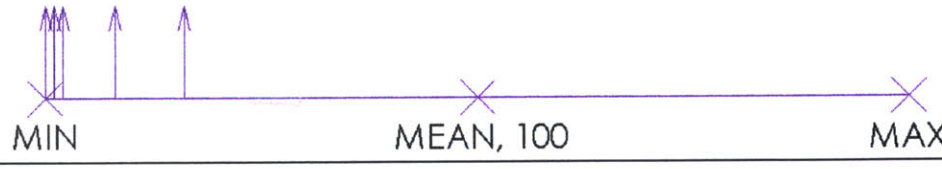
B.



C.



D.



Question 5:
Something you've designed and analyzed has an estimated safety factor between 2.5 and 3.0. Draw a plot of the safety factor distribution over that range.

Question 6:
Something you've designed and analyzed has an estimated weight between 9.0 and 10.0 pounds. Plot the weight vs. likelihood of occurrence. In other words, put the weight on the x axis, and the probability of that weight on the y axis.

Question 7:
Something you've designed and analyzed has an estimated volume between 10 and 12 cubic inches. Draw the probability distribution for the volume . If you don't know what a probability distribution is, guess.

Question 8: (optional)
If you have any feedback on this survey, please provide it here. Or you can just talk to me.

REFERENCES

Abrahamson, S., Wallace, D. R., Senin, N., Borland, N., Integrated Engineering, Geometric, and Customer Modeling: LCD Projector Design Case Study, *Proceedings of the ASME DT Conferences*, DETC/DFM-9084, 1999.

Chapra, S. C., Canale, R. P., *Numerical Methods for Engineers*, McGraw-Hill, 1988.

Deniz, J. C., Learning Theory Applications to Product Design Modeling, Master of Science Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2000.

Eggert, R.J., Preliminary Robust Design Using Advanced Spreadsheet Modeling Techniques, *Proceedings of the 1995 Design Engineering Technical Conferences*, **2** (1995) pp.43-53.

Haugen, E. B., *Probabilistic Mechanical Design*, John Wiley & Sons, 1980.

Kalos, M. H., *Monte Carlo Methods*, John Wiley & Sons, 1986.

Kalpakjian, S., *Manufacturing Engineering and Technology*, Addison-Wesley Publishing Company, 1995.

Masters, T., *Practical Neural Network Recipes in C++*, Academic Press, 1993.

Regnier, E. and Hoffman, W. F., Uncertainty Model for Product Environmental Performance Scoring, *Proceedings of the 1998 IEEE International Symposium on Electronics and the Environment*, (1998) pp. 207-12.

Sedgewick, R., *Algorithms in C++*, Addison-Wesley, 1992.

Shigley, J. E. and Mischke, C. R., *Mechanical Engineering Design*, McGraw-Hill Inc., 1989.

Siddall, J. N., *Probabilistic Engineering Design*, Marcel Dekker, Inc., 1983.

Tatang, M. A. and McRae, G. J., *Direct Treatment of Uncertainty in Models of Reaction and Transport,* Massachusetts Institute of Technology, 1994.

Wallace, D. R., Abrahamson, S., Senin, N., Sferro, P., Integrated Design in a Service Marketplace, *Computer-aided Design*, **32** 2 (2000) pp.97-107.

Webster, M., Tatang, M. A. and McRae, G. J., *Application of the Probabilistic Collocation Method for an Uncertainty Analysis of a Simple Ocean Model*, MIT Joint Program on the Science and Policy of Global Change, Report No. 4, 1996.