

**OPTIMIZATION OF RELIABILITY  
ALLOCATION STRATEGIES THROUGH USE  
OF GENETIC ALGORITHMS**

James E. Campbell  
Laura A. Painton  
Sandia National Laboratories, MS 0746  
Manufacturing Systems Reliability Department  
Albuquerque NM 87185

**Abstract**

This paper examines a novel optimization technique called genetic algorithms and its application to the optimization of reliability allocation strategies. Reliability allocation should occur in the initial stages of design, when the objective is to determine an optimal breakdown or allocation of reliability to certain components or subassemblies in order to meet system specifications.

The reliability allocation optimization is applied to the design of a cluster tool, a highly complex piece of equipment used in semiconductor manufacturing. The problem formulation is presented, including decision variables, performance measures and constraints, and genetic algorithm parameters. 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 Mean Time Between Failures while staying within a budget. The results show that the genetic algorithm is very efficient at finding a set of robust solutions. A time history of the optimization is presented, along with histograms of the solution space fitness, MTBF, and cost for comparative purposes.

**Introduction**

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 topology that will meet system objectives. Reliability allocation should start from a base of past experience. For example, some initial design structures can often be generated with reliability estimates for the subsystems. Reliability allocation is not the same as detailed design tradeoff studies, where specific design options are evaluated for cost/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.

Reliability allocation is a subset of the more general problem of design optimization. System design optimization is difficult because there are many ways one can define subsystems and components to fulfill the functional design requirements for a system: one can alter a design by changing the structure of a configuration, the parameters of the individual components, or a mix of these. Because of the number of potential components, structural arrangements, and parameter values, system design optimization problems can quickly become large (hundreds of thousands to millions of solutions) through combinatorial explosion. Often design decisions are made at a component level without a systematic approach that examines structural linkages between parts of the overall process and uncertainties throughout the system. Financial considerations may be taken into account using a simple cost/benefit analysis of individual units, but this can lead to suboptimal decisions. A structured approach which explicitly incorporates the potential combinations through different stages of the system and their effect on other parts of the system will contribute to improved design processes and more reliable products.


This paper presents the formulation of genetic algorithms for a specific application, reliability allocation.

**Optimization Methodology**

Past work on reliability optimization has focused on Mixed Integer Nonlinear Programming (MINLP) algorithms for specific structural systems. Many of the reliability optimization models are formulated as  $N$ -stage series systems. At each stage  $i$ , there is the option to add redundancy in the form of parallel components. The optimization problem is to determine the optimal number of parallel components at each stage along with the reliability of each component to maximize system reliability subject to resource constraints such as cost or weight. This problem is often formulated as a MINLP where the integer variables denote the number of parallel components to include at each stage of the system, and the continuous variables denote the reliability of these components.

Limitations of mathematical programming approaches are that they require a rigid system

MASTER

1 DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED 

American Institute of Aeronautics and Astronautics

This work was supported by the United States Department under Contract DE-ACO4-94AL85000.

**DISCLAIMER**

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

## DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

structure, they allow for comparisons between systems of incrementally different structure but not radically different structure, and they often do not incorporate redundancy or parallelism easily. In many design optimization problems, the structural topology influences the optimal parameter settings so a simple de-coupling approach does not work. That is, one cannot optimize the structural topology and then the parameters because the optimal topology may depend upon the performance of the system at certain parameter levels. Finally, the complexity of these problems depends upon the form of the objective function. In the past, solution techniques typically depended upon objective functions that were single-attribute and linear (i.e., minimize cost). However, real problems often require multi-attribute objectives such as minimizing costs while maximizing safety and/or reliability and ensuring feasibility. In these cases, the goal is to optimize over a set of performance indices which may be combined in a nonlinear objective function.

Genetic algorithms were chosen as a solution technique because of their flexibility in overcoming some of the above limitations, and because of the natural combinatorics involved with reliability optimization. Genetic algorithms are a combinatorial optimization method. In a design, one can list possible components, upgrades or replacements, redundancies, improvements, etc. A combination of components with specified failure rates in a certain configuration is a solution. The goal is to find the combination of all possible combinations which optimizes the reliability objective. The genetic algorithm approach also has the significant advantage of incorporating uncertainty in component failure rates into the optimization of system reliability, something that math programming formulations generally do not allow.

### Genetic Algorithms

Genetic algorithms take their inspiration from the biological world.<sup>1</sup> Genetic algorithms operate by creating an initial "population" of solutions (usually represented as bit strings) 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" by replacing the value of a random bit on 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.<sup>2</sup> As in natural selection, genetic algorithms process fitness information and rank solutions according to their survival capabilities. (Note: fitness refers to the value of the objective function. For example, if the

objective was to maximize the MTBF, a solution with a higher MTBF would have a higher fitness than one with a low MTBF).

Many reliability optimization problems can be easily formulated in a genetic algorithm framework. A solution "chromosome" is a bit string of "genes," where each gene represents a failure mode or a component of the design and its associated reliability level. The "fitness" of a particular solution can be obtained by solving the reliability equations for a design involving all of the units present on the chromosome, with the component reliability given by allele levels on a gene. Genetic algorithms have been applied to combinatorial optimization problems in engineering design<sup>3</sup> and reliability.<sup>4</sup> We have successfully applied genetic algorithms to reliability improvement problems<sup>5</sup>, where the objective is to identify the types of component improvements and the level of effort spent on those improvements to maximize one or more performance measures (e.g., reliability or system availability) subject to constraints (e.g., cost) in the presence of uncertainty about the component failure rates. Reliability improvement generally occurs after a design exists and upgrades are made to improve system performance, for example in a later release or version. In this paper, we expand the previous work to address reliability allocation problems. Such problems would occur in the initial stages of design, when the objective is to determine an optimal breakdown or allocation of reliability to certain components or subassemblies in order to meet system specifications.

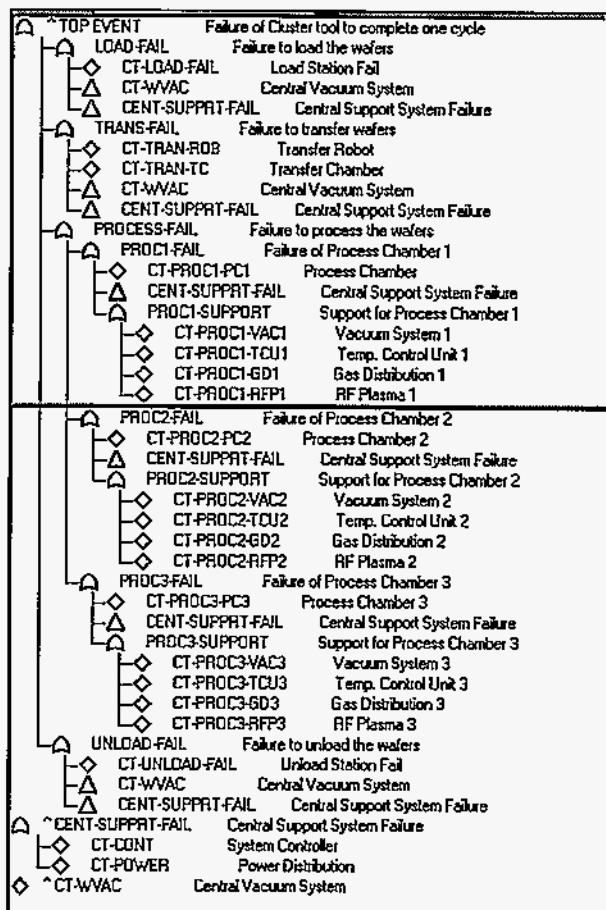
### Application

The reliability allocation optimization will be applied to a problem of cluster tool design. A cluster tool is a highly complex piece of equipment used in semiconductor manufacturing. It 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 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 1. Cluster Tool**

The reliability of the system is modeled through a fault tree. The top event (system failure) may be defined in many ways, but we choose to define it 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 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.

We have used the WinR™ software developed at Sandia National Laboratories in the modeling and optimization of the reliability allocation problem for the cluster tool. WinR™ supports the modeling of system reliability, with capabilities for nondeterministic modeling of system reliability, superior data input formatting and handling, fault tree editing and viewing, graphical displays and editing of graphics output, and optimization capabilities. Both repairable and non-repairable systems can be modeled in WinR™. WinR™ has been designed to allow for iterative reliability modeling at all stages of design, from initial conceptual design and reliability allocation, to tradeoff studies of more specific designs, to reliability improvement strategies for existing systems, and optimal spares identification.



**Figure 2. Fault tree for Cluster Tool Problem**

**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.

**Decision Variables**

The decision variables in reliability allocation are what 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 or failure mode is conceptualized. Then, for each failure mode, 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 1: Reliability improvement and corresponding cost

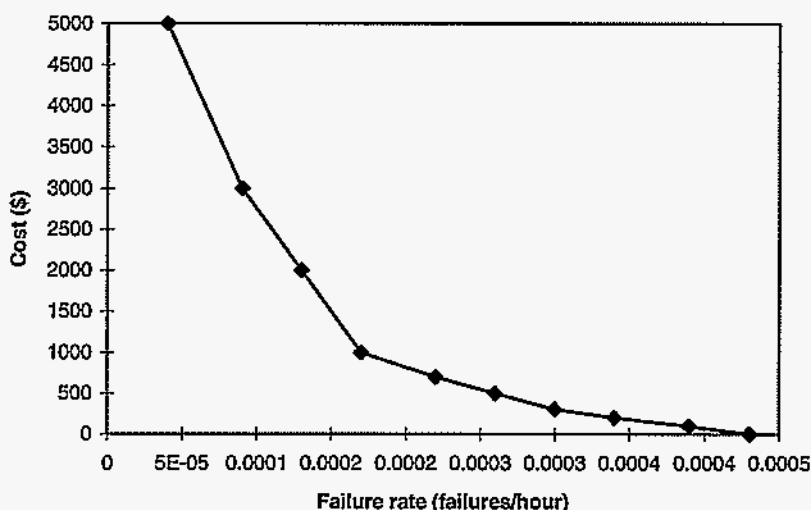


Figure 3: Effort Curve for reliability improvement

Table 1 defines a piecewise “effort curve” shown in Figure 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.

Effort curves are defined for all failure modes in the initial design. The optimization will pick those

combinations of “effort” or reliability levels for each component which optimize the objective.

**Performance Measures and Constraints**

Performance measures refer to the goals of the optimization. Typically they include goals such as maximizing system MTBF or reliability, minimizing system repair time, maximizing availability, and/or minimizing maintenance costs. WinR™ will optimize a deterministic or a *stochastic* (uncertain) objective. A

decision maker may be interested in maximizing the mean of this distribution, or a percentile. If the system is designed for customers who receive a warranty, the decision maker may decide to maximize a certain percentile, such as the fifth percentile, so that he or she can be assured that only five percent of the customers will have an MTBF less than the warranty period. A major advantage of WinR™ is that it can optimize one or more performance measures in the presence of uncertainty about the component failure rates. We have shown that genetic algorithms perform well in the face of the statistical noise or "jitter" in the output space induced by the uncertainties in the component failure rates.<sup>5</sup> In the case of reliability allocation, we are not including uncertainty in the performance measures so the optimization is deterministic.

In addition to the performance measures, it is necessary to define the constraints upon the system. These can be in terms of weight limits or other physical limits. 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 main constraint.

Genetic algorithms are an unconstrained optimization technique, that is, they do not explicitly account for constraints. There are two basic methods for incorporating constraints:

1. Only allowing solutions which satisfy the constraints
2. Penalizing the solutions which fail to satisfy the constraints by adding a penalty function to the objective. Penalty functions may be linear, quadratic, logarithmic, etc. functions of the deviations of the constraints and/or the number of violated constraints

The implementation of constraints is difficult. If one incorporates a high penalty function for constraints that cannot be violated, one runs the risk of creating a space of mostly infeasible or illegal solutions. This also occurs if any solution which violates the constraints is simply rejected. The problem is that this may lead to evaluation of many solutions before a feasible one is found, which can add significant computation time depending upon the amount of infeasibility in the space. Further, when one legal individual is found, it may drive out other solutions and a type of premature convergence may result. Moderate penalties may also lead to anomalous results, especially if individuals which violate some constraints are rated higher than ones which meet all the constraints at a lower fitness level: these techniques may find a set of solutions which excel

on one or more dimensions and have a higher objective than a feasible solution.

Penalty functions are usually constructed in an ad-hoc manner, and often require much to be known about the problem and its structure before the implementation of an optimization technique, which is not always the case. There are no easy or clear answers to the difficult issues of objective or fitness definition, standardization of fitness measures, the collapsing of many objectives into one utility measure, or constraint implementation. Recent research suggests that variable or dynamic penalty functions which change over time are most effective.<sup>6,7</sup> We have found that objective functions which gradually degrade as the performance goals get worse and/or as the constraints are violated are the most robust. Thus, we have defined our objective function as:

$$(w_1P_1+w_2P_2+\dots)*C \quad (1)$$

where  $P_1, P_2, \dots, P_n$  are performance measures (such as MTBF, system availability, etc.),  $w_1, w_2, \dots, w_n$  are the relative weights on those performance measures, and  $C$  is the cost function. The performance measures and cost function are defined by their lowest or highest acceptable limit and the objective.

In the case of the cluster tool, there is only one performance measure and that is MTBF. The "base case" design before reliability allocation has an MTBF of approximately 100 hours. The user can specify a lower limit which he or she will accept from the optimization, for example, 150 hours, along with a system objective of 200 hours. Similarly, the budget constraint for the cluster tool problem was given an upper limit of \$20,000 (no solutions with a cost above this will be accepted) and an objective of \$10,000. The value of  $P_1$  for a particular design is specified according to the graph shown in Figure 4.

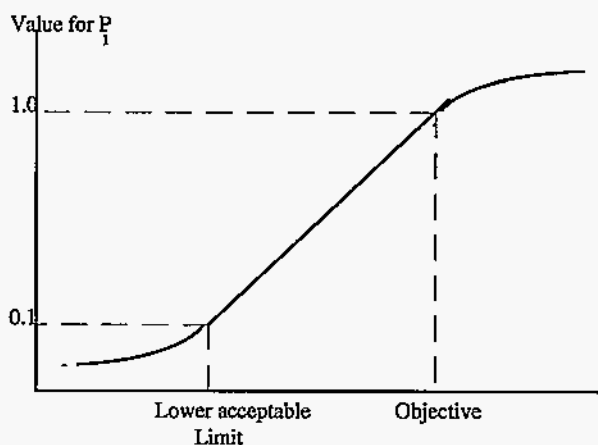


Figure 4. Fitness function for performance measure



The individual functions for  $P_1, P_2, \dots, P_n$  and  $C$  are designed to have a value of 1 if the objective for the particular performance measure or cost is obtained. If they simply meet the lowest acceptable limit, they are given a value of 0.1. Between the acceptable limit and the objective, the fitness function increases linearly. If the objective is achieved, the fitness still increases but in a decreasing fashion according to a quadratic function, and likewise, the fitness tapers off to zero if the performance or cost is below the acceptable threshold.

### Genetic Algorithm Parameters

After the decision variables and objective function (Equation 1) are defined, the combinatorial optimization problem is ready to be implemented in a genetic algorithm. We will not go into all the details of genetic algorithm implementation; Reference 2 provides an excellent overview. The reliability allocation problem had an implementation which corresponds naturally to the problem framework: the population members are solution "chromosomes" composed of "genes." Each gene represents a failure mode or a component of the design and its associated reliability level according to the effort curves given in Table 1 and Figure 3. Thus a solution might be coded as [0.8|0.5|0.4|0.7|...|0.1] where the genes are separated by |. In this example, the first gene represents the first component and the failure rate for this component is the base case failure rate multiplied by 0.8. Likewise, the second component in this design has a reliability allocation of 0.5 times its base case reliability, etc.

At each generation, every solution (population member) represented by the above coding is "decoded" to a set of components, failure modes, and failure rates and these are processed by WinR™'s fault tree solver to obtain the performance measures of interest such as MTBF, reliability, and cost. The performance measures and cost are fed to the objective function to obtain a fitness value for that particular solution. The high fitness solutions are picked by the genetic algorithm to mate to produce the next generation. The process of mating high fitness solutions leads to the "evolution" of optimal solutions in later generations.

There are some parameters which govern the performance of the genetic algorithm; these include the crossover or reproduction rate and the mutation rate. We have used a mutation rate of 0.1 and a crossover rate of 0.5. There are also other factors which influence the performance, including parent selection methods and crossover schemes. We are using tournament selection to pick the parents. That is, two population members are arbitrarily selected and the one with the higher fitness becomes a parent. It is mated with the

higher fitness solution from a comparison of two other population members. We are using two point crossover.

WinR™ allows the user to specify the number of generations and the population size. Both the number of generations and the population size control how long the optimization runs and the number of solutions examined. Larger population sizes over more generations is usually better because more solutions are examined, minimizing the possibility of the genetic algorithm getting trapped in a "local optimum" (i.e., a very good solution) and not the "global optimum" (the best possible solution). However, there is a computational tradeoff: larger populations over more generations takes longer to evaluate. At some point, the fitness value will start to plateau out or converge. We have found that running a population of size 50 to 100 for 40 generations is a good starting point for the optimization.

The performance of genetic algorithms are 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.

### Results

The cluster tool reliability allocation problem was run using a genetic algorithm implemented in the WinR™ software. This example has twelve major subassemblies or failure modes, 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 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 to best meet the system objectives. There certainly could be 1 trillion combinations of potential allocations.

The optimization was run using a population size of 100 for 50 generations. The fitness history is shown in Figure 5, and the evolution of MTBF is shown in Figure 6. In Figure 5, the solution member with the best or "maximum" fitness is tracked, along with the average fitness and the minimum fitness solution. Note that the genetic algorithm performs very well, honing in on optimal solutions by 20 generations. The MTBF of the optimally allocated solutions is more than double that of the base case (approximately 100 hours vs. 220 hours).



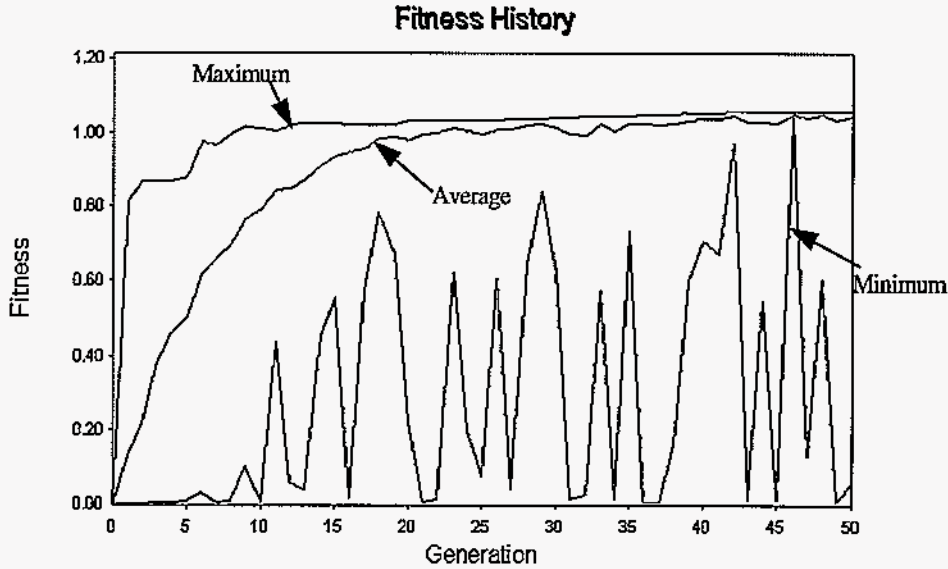


Figure 5. Fitness History

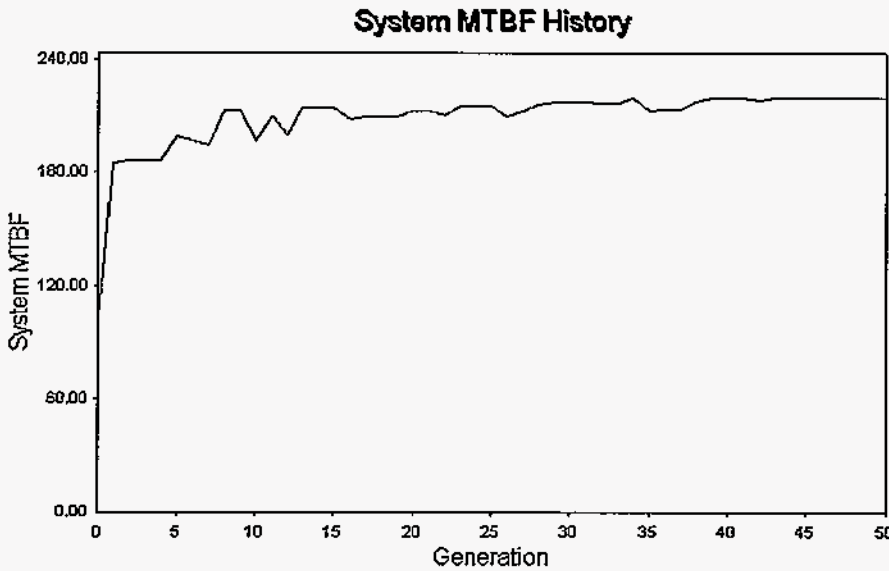


Figure 6. MTBF History

WinR™ tracks the top ten solutions over all generations. The top three solutions are shown in Table 2. Notice that they are very similar. This indicates that the solution is robust. For example, all solutions have the failure rate of the load station allocated at 60% of its base case, the failure rate of the central vacuum system at 50% of its base case, the failure rate of the central controller at 20% of its base case, etc. There are minor differences with respect to the reliability allocation for the power distribution, the unload station, and the robot. The fitness for the top solutions is the same to two decimal places, and the slight tradeoffs between cost

and increased MTBF can be seen. For example, solution 2 has an MTBF which is almost five hours greater than that of solution 3, but costs an additional \$600 more. These results can provide guidance for the target levels of reliability that subassembly should attain in a more detailed design.

Failure Mode	Solution 1	Solution 2	Solution 3
Load Station	0.6	0.6	0.6
Central Vacuum	0.5	0.5	0.5
Controller	0.2	0.2	0.2
Power Distribution	0.5	0.4	0.4
Unload Station	0.5	0.5	0.6
Robot	0.5	0.5	0.6
Transfer Chamber	0.9	0.9	0.9
Process Chamber	0.8	0.8	0.8
Chamber Vacuum	0.4	0.4	0.4
TCU	0.4	0.4	0.4
Gas Distribution	0.2	0.2	0.2
RF Plasma	0.3	0.3	0.3
<b>RESULTS</b>	<b>Solution 1</b>	<b>Solution 2</b>	<b>Solution 3</b>
Fitness	1.06	1.06	1.06
System MTBF (hours)	219.1	220.5	215.7
Availability	0.952	0.953	0.952
Improvement Cost (\$)	10.700	10.900	10.300

Table 2. Top solutions in population

To evaluate the performance of the genetic algorithm and provide some insight about the shape of the solution space, a partial enumeration of the solutions was performed by evaluating 10,000 randomly selected combinations. The results are shown in Figures 7 through 9. Figure 7 displays a histogram of MTBF values for potential combinations of reliability allocation in the solution space. The optimal solution found by the genetic algorithm (solution 1) has an MTBF of 219 hours, which is in the upper third of all solutions. Note that it is possible to attain higher reliability solutions, but not within the cost constraint. The cost histogram, in Figure 8, shows that the optimal solution lies at the lower end of the cost distribution over the solution space. Thus, the genetic algorithm found a set of solutions in a small portion of the solution space which attain the reliability goals for a very low cost. Figure 9 shows the histogram of solution fitness. This shows that most of the solutions have a very low fitness, and the optimal solution fitness value of 1.06 is at the tail end of the fitness distribution. The genetic algorithm found a set of reliability allocations extremely efficiently: by examining approximately 3000 solutions in a space of 1 trillion combinations, a tiny fraction of the solution space ( $3 \times 10^{-9}$ ). Note: the 3000 solutions was calculated as follows: at each of 50 generations, with a crossover rate of 0.5, half of the 100 population members reproduce so there are 50 new members in the subsequent generation plus the mutated ones,  $0.1 \times 100$  or 10, thus there are about sixty new members and 40 old members which have previously been evaluated in each generation.

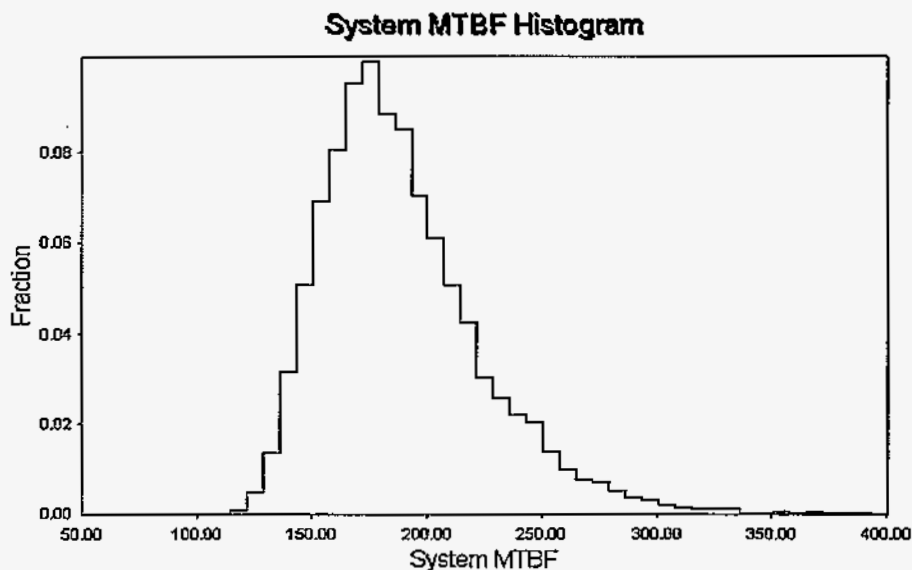


Figure 7. Histogram based on partial enumeration of the solution space

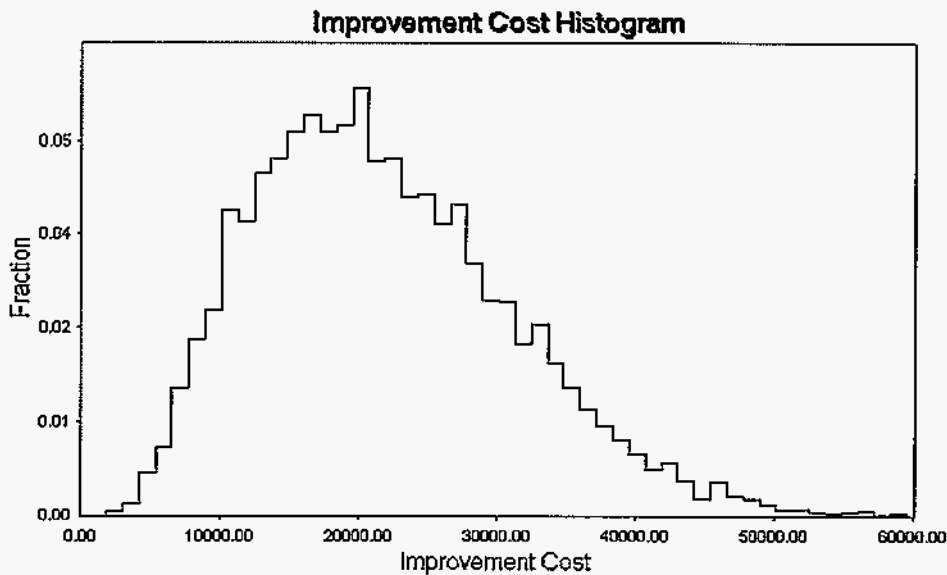


Figure 8. Cost Histogram

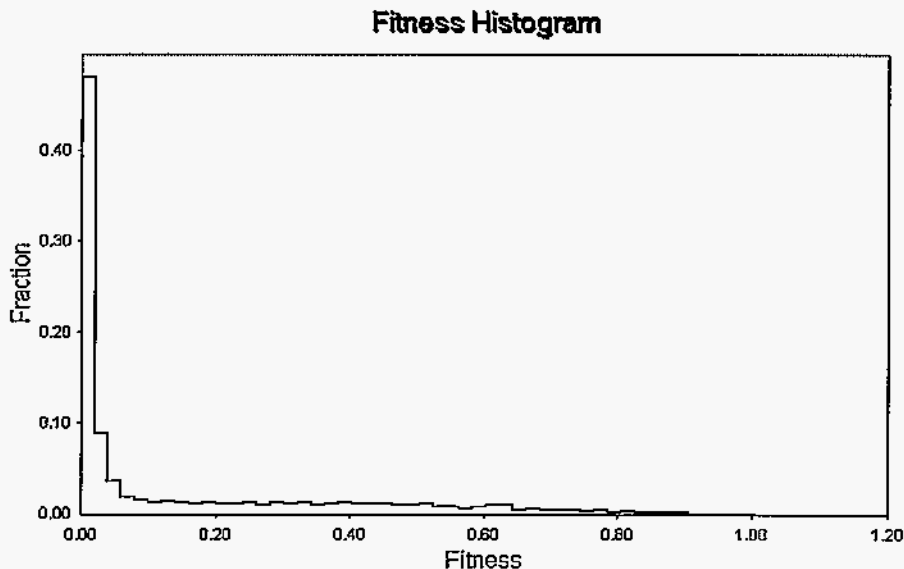


Figure 9. Fitness Histogram

### Conclusions

Reliability allocation should be one of the first steps in the design process since it can guide later tradeoff and improvement studies of more detailed designs. This paper presented a novel optimization technique called genetic algorithms and its application to the optimization of reliability allocation strategies. The reliability allocation optimization was applied to the design of a cluster tool. The results show that the genetic algorithm is very efficient at finding a set of robust solutions: 3000 solutions in a space of 1 trillion combinations were examined, a tiny fraction of the

solution space. This research builds on previous research using genetic algorithms for optimization of reliability improvement. We have shown that genetic algorithms can also successfully be used for optimization of reliability allocation, thus making them a powerful tool in an iterative design process.

### References

1. Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press: Ann Arbor. 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 Altos, CA.
4. Coit, D. W. and A. E. Smith (1994). "Use of a Genetic Algorithm to Optimize a Combinatorial Reliability Design Problem," *Proceedings of the Third IIE Research Conference*, 467-472.
5. Painton, L. and J. Campbell (1995). "Genetic Algorithms in the Optimization of System Reliability," *IEEE Transactions on Reliability, Special Issue on Design*. 44(2), 172-178.
6. Siedlecki, W. and J. Sklansky (1989). "Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and its Use in Pattern Recognition," *Proceedings of the Third International Conference on Genetic Algorithms*. J. D. Schaffer, ed. Morgan Kaufman: Los Altos, CA.
7. Smith, A. E. and D. M. Tate (1993). "Genetic Optimization using a Penalty Function," *Proceedings of the Fifth International Conference on Genetic Algorithms*. S. Forrest, ed. Morgan Kaufman: Los Altos, CA.
8. Greffenstette, J. J. (1986). "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics SMC-16*, 1, 122-128.
9. Schaffer, J.D., R. A. Caruana, L.J. Eshelman, and R. Das (1989). "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization," *Proceedings of the Third International Conference on Genetic Algorithms*. J. D. Schaffer, ed. Morgan Kaufman: Los Altos, CA.