# Using Genetic Algorithms for Naval Subsystem Damage Assessment and Design Improvements

Christopher McCubbin
Johns Hopkins University
Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
mccubcb1@jhuapl.edu

David Scheidt
Johns Hopkins University
Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
david.scheidt@jhuapl.edu

Oliver Bandte
Icosystem Corporation
10 Fawcett Street
Cambridge, MA 02138
oliver@icosystem.com

Steven Marshall
Johns Hopkins University
Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
steven.marshall@jhuapl.edu

Iavor Trifonov
Icosystem Corporation
10 Fawcett Street
Cambridge, MA 02138
iavor@icosystem.com

## ABSTRACT

Some auxiliary systems of next generation naval ships will utilize distributed automatic control. Such distributed control systems will use interconnected sensors, actuators, controllers and networking components to diagnose and reconfigure the auxiliary systems. Testing these systems will be difficult with traditional methods of fault analysis due to the interconnected and automatic nature of these subsystems. We have designed a suite of genetic algorithms to find interesting and hidden damage scenarios in a testbed of a naval subsystem. Given this knowledge, we use a genetic algorithm to improve upon the design of this subsystem.

## Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Numerical Analysis—*Optimization*

## General Terms

Experimentation, Design

## Keywords

Application, Multi-objective optimization, Design/synthesis, Coevolution

## 1. INTRODUCTION

Some auxiliary systems of next generation naval ships will utilize distributed automatic control. Such distributed control systems will use interconnected sensors, actuators, controllers and networking components to diagnose and reconfigure the auxiliary systems. Auxiliary systems are designed with redundant capabilities to assure survivability and robustness in the presence of battle damage. However, because damage to the automatic control system can result in loss of auxiliary system control, survivable auxiliary systems require that control systems be designed with redundancy greater than or equivalent to the redundancy inherent in the auxiliary system.

Building on previous efforts[1], We have used evolutionary testing to evolve challenges, i.e. component level damage, to an onboard ship system on which is layered a set of intelligent agents that make inferences about the system's state and reconfigure the system. The goal was to challenge the system in an effective manner, seeking out counterintuitive scenarios or minor damage that results in significantly impaired system performance. This was then followed by evolutionary testing of the design itself. Thus, the ultimate goal was to devise a prototype design tool and approach for the creation of distributed control systems that will result in control systems, and consequently auxiliary systems that are more robust and survivable. Key design elements that were investigated were the location of sensors. Future investigations would include the physical location of control system components, the number of sensors, and the scope, or "responsibility," of high level intelligent control agents.

In this paper we describe the design of our testbed, our experiments, and results.


*GECCO'07,* July 7–11, 2007, London, England, United Kingdom.


## 2. SYSTEM DESIGN

This section describes the testbed that was used to perform the optimization of damage scenarios and system design.

### 2.1 Overall System Design

A diagram of the overall system design can be seen in Figure 1.

The plant model, sensor model, diagnosis model, and control model make up the core of the fitness function for the genetic algorithms. These components are described in detail in sections 2.2, 2.4, and 2.3.

Two separate genetic algorithms are displayed in the system diagram. One, the Damage GA, consists of the components labelled "Damage Fitness Evaluator", "Damage Genetic Algorithm Stim", "Failures", and "Spatial Model". The other GA, referred to as the Design GA, consists of the components labelled "Sensor Placement

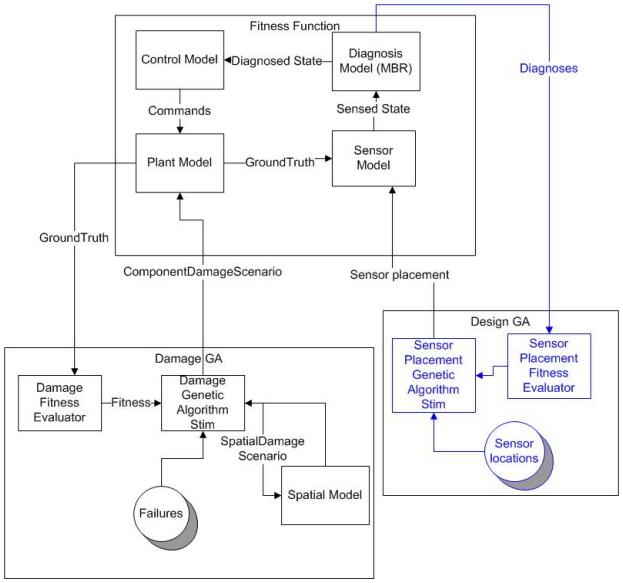


**Figure 1: Oak 3 System Block Diagram**

Fitness Evaluator", "Sensor Placement GA Stim", and "Sensor Locations".

The general idea here is that the fitness function represents a simulation of a target system plus the control software that operates on that system. The damage GA and design GA are attempting to use this function to measure the worst damage event possible, or the best design of the sensor system, respectively. How this was actually calculated is detailed in sections 2.7 and 2.6.

### *2.1.1 Component high-level descriptions*

The components of the fitness function each serve a different role in simulating the target system. The Plant Model acts as a ground-truth simulator of the system's physical components. The Plant Model can simulate the effects of damage on the system and can accept and implement low-level system commands. The Sensor Model models the sensors that are detecting the state of the physical system. The Diagnosis Model and Control Model represent the target system's control system. In our setup we give the control system one shot to determine the state of the system from limited sensor data and attempt to reconfigure the system to a better state.

The components of the GA's attempt to come up with optimal damage or sensor placements based on the methods of Genetic Optimization. Their operation is described in sections 2.6 and 2.7.

## 2.2 Plant Model

To generate ground truth for the diagnosis model, we utilized a model of the Chilled Water demonstration system built in Epanet2 [6].

The Chilled Water Reduced Scale Autonomy Demonstrator (RSAD) is the target hardware-in-the-loop simulator used for this research. The NSWC CD Philadelphia, Auxiliary Machinery Controls & Automation Branch constructed the RSAD as a model of a reconfigurable fluid system test platform to demonstrate technologies for unmanned operation. The RSAD fluid system hardware consists of valves, sensors, chillers, heat exchangers which act as chilled-water consuming services, and pumps. In normal operation we will attempt to configure this system to provide chilled water to as many important services as possible. A picture of the RSAD can be seen in Figure 2.

**Figure 2: Reduced Scale Autonomy Demonstrator (RSAD)**

## 2.3 Diagnosis Model

Once ground truth has been provided by the plant model, the ground truth is filtered using a sensor model, and then sent to the diagnosis model. This filtration process reduces the ground truth to what can be determined using the current sensor configuration being evaluated. The diagnosis model then attempts to form a complete picture of the system state using only the sensor readings. The diagnosed system state can then be compared against the ground truth to form a metric of diagnosis success, which is then utilized by the GA for evaluating the ability of the current sensor configuration to correctly assess the faults in the system. To perform diagnosis on the sensed system state, we used the Livingstone 2 Model-Based Reasoning Engine (MBRE)[7].

## 2.4 Control Model

The Control Model, as described in section 2.1, looks at the Diagnosed State produced by the Diagnosis engine and attempts to create a set of low-level commands that will produce the best system configuration possible. How well the Control Model operates is fairly dependant on how good the diagnosis is.

The current control model is a planner that is capable of producing reconfiguration plans for water supply systems that include pumps, resources that need to be supplied, and a system of valves and redundant supply lines. It takes as input a description of the condition of the system, including pipe, valve and pump states. It then attempts to create a configuration of these components that supplies as many services as it can given the state of the system. The planner also has the capability to supply more important services first, and the capability to only supply one of a number of redundant services if needed.

## 2.5 Genetic Algorithms

John Holland first proposed genetic algorithms in the 1970s as a search technique that exploits the idea of biological evolution as it occurs in nature [5]. The classical genetic algorithm is traditionally used for search and optimization along the line of one objective as defined by the fitness function. However, our problem requires the optimization of two or more objectives at the same time, while often these multiple goals are at odds with each other.

One way to address such a multi-objective problem with a traditional GA would be to incorporate both goals in the fitness function. As an alternative, multi-objective genetic algorithms[4] can simultaneously optimize multiple objectives. For our purposes we

chose to try both a single-objective fitness function and compare results with the multi-objective algorithm NSGA2 [3], which represents the current state of the art in multi-objective evolutionary optimization. It implements a standard Pareto frontier based ranking algorithm and a binary tournament selection algorithm.

## 2.6 Damage Optimization GA

The goal of the damage GA is to discover damage scenarios for the plant that can render vital services inoperative. We worked on several different variations of this general idea. We looked at damage scenarios that address specific elements chosen across the entire plant, and scenarios where failures are dependant on the natural failure rate. In addition, we guide the evolutionary search along three different dimensions: severity of the damage, likelihood of the damage, and the complexity of the scenario. Depending on the different combinations of these elements we developed several single-objective and multi-objective GAs that are described in detail in the following subsections. In all cases, a damage scenario is simply a set of system components that the plant model would be instructed to consider broken as part of its initial condition configuration. The severity of damage scenarios would then be evaluated based on how many and which of the operational loads in the system remain functional.

To avoid obvious solutions we limit the total number of damaged components and only look at damaging valves, pipes, chillers, and pumps, leaving the actual services and loads that characterize the operational state of the system out of reach. If we let our GAs damage services and loads directly, it would be trivial to find severe damage scenarios by just marking the services themselves as broken. Similarly, if our GAs have the freedom to damage as many components as possible, they would quickly find that more damaged components means more severely incapacitated system. Truly interesting scenarios are these that have great consequences but are caused by deceivingly small problems.

### *2.6.1 Damaged Components GA*

The damaged components GA (DCGA) is a single-objective GA that searches for a set of N broken components that would constitute the damage scenario. The size of the set is a parameter that is specified by the user. As already explained, the smaller the size of the set, the more interesting the damage scenario would potentially be. The challenge for DCGA is to discover severe damage scenarios that involve only a few components.

#### *Fitness function.*

To guide the evolutionary search DCGA uses a fitness function that evaluates damage scenarios and calculates a score for each on of them. DCGA's fitness function does that by setting the scenario that is being evaluated as the initial condition for the plant model simulation and runs it. Next, the sensor model and the diagnostic model are run to obtain the perceived state of the system which is fed to the control model. The control model in turn issues a list of commands in response to any changes from the nominal state of the system. Finally, these commands are applied by the plant model and the final state of the system is produced after another simulation.

The fitness function inspects the final state of the system and calculates a number that reflects the severity of the state by looking at which services have been rendered inoperative - whether they were turned off by the control model, or burned out due to overheating. Different services have different importance to the operation of the system and the fitness function assigns different points to each service. In addition, some services are redundant, so having two of the same kind working at the same time is not much different than having only one when it comes to the operational state of the system. Yet, having one of two redundant services down is rewarded differently than having both services open since this is an intermediate step towards solutions that might have both services down.

For convenience DCGA tries to minimize the fitness function across its population. To achieve that, the fitness function assigns points for every service that remains operational after the infliction of the damage scenario that is being evaluated. The points for each service range from 1 to 15.

The redundant service pairs are scored together, and slightly higher score is used when both services are working. The score values were chosen with the following considerations in mind: a vital load should be equivalent in fitness to more than one other vital load, but less than all of them summed together; and redundant loads should affect the fitness the same as one turned-off non-redundant load if both are turned off. If one or the other of the redundant loads is on, the fitness should be equivalent to a turned-on vital service. If both are on, the fitness should be equivalent to one vital plus one nonvital load. The actual numeric fitness value is calculated by finding which services are operational and adding up the corresponding scores.

### *2.6.2 Damaged Components GA Experiments*

In our initial tests, we concentrated on finding damage scenarios where N = 3. In other words, we look for combinations of 3 components that can incapacitate the system as severely as possible. Table 1 lists the other GA parameters that we used for our simulation runs.

| Parameter | Value |
|---|---|
| Random seeds | 1, 10, 20, 30, 40 |
| Population size | 30 |
| Generations | 25 |
| Elite size | 5 |
| Mutation probability per gene | 0.05 |
| Crossover probability per gene | 0.10 |

**Table 1: DCGA simulation run parameters**

According to our results DCGA was able to find damage scenarios which completely incapacitate the system in each of the 5 runs that we conducted.

| Scenario | Damaged components |
|---|---|
| 1 | PH1S,V002R,V101B |
| 2 | PG2S,PP201B,V002S |
| 3 | PE3S,V002R,V231S |
| 4 | PA4S,V122R,V234R |
| 5 | PA1S,V121R,V204S |
| 6 | ACP101,V122S,V204S |
| 7 | V006R,V121R,V122S |
| 8 | V003S,V006S,V231S |
| 9 | V122R,V127S,V230R |
| 10 | V121S,V127R,V231R |

**Table 2: DCGA sample damage scenarios**

Table 2 lists some sample damage scenarios discovered by DCGA that incapacitate the entire system. Because DCGA is free to explore any combination of three components the broken elements in these scenarios are scattered across the entire system. Even though such scenarios are unlikely, they may constitute a

good benchmark for constructing robust designs of the chilled water system.

### 2.6.3 Natural Failures GA

The natural failures GA (NFGA) addresses the problem of finding a severe as well as a likely natural failure scenario. Each component in the system has a certain probability of failing from natural tear and wear, which is measured in terms of the mean time between failures (MTBF). Table 3 lists the MTBF for every type of component. All components that are of the same type share the same probability of failure.

| Component type | MTBF in days |
|---|---|
| Pipe | 180 |
| Valve | 30 |
| Service | 15 |
| Load | 60 |
| Chiller | 20 |
| Pump | 20 |

**Table 3: MTBF in days per component type**

When using NFGA the user can specify N - the number of damaged components just as in DCGA. The difference is that in addition to any damaged components, there is an arbitrary set of components that are considered failed due to natural causes. The combined set of damaged and failed components constitutes the damage scenario that is evaluated.

If the user sets N to zero, then NFGA would explore the space of natural failure scenarios. It would try to discover a set of components that fail naturally at the same time to bring as many services down as possible. The difference from DCGA is that NFGA considers the total likelihood of the natural failures occurring at the same time. The more natural failures the chromosome accumulates, the less likely the scenario becomes as per Table 3.

One of the purposes of NFGA is to explore this tradeoff - severity of the scenario versus total likelihood of such a scenario occurring in reality. The N parameter can be used to balance that tradeoff. The larger N is, the more components NFGA can damage without incurring a loss in likelihood. The more damaged components there are, the fewer natural failures we need to produce a severe damage scenario altogether, and thus the more likely it would be.

#### *Fitness function.*

The addition of failure to the chromosome and NFGA's ability to fail as many components as necessary introduce the need to balance the total loadScore with the likelihood of the failures to occur in the first place. Since NFGA has to minimize the score but maximize the probability of the failure to occur, the formulation shown in Equation 1 was used.

$$Fitness = \frac{1 + loadScore}{\prod_{i=1}^{Allfailures} \frac{1}{MTBFComponent_i}} \quad (1)$$

The loadScore term represents the total damage severity and is calculated as described earlier. The larger the probability of all failures in the chromosome to occur at the same time, the smaller the fitness value will be. In Equation 1 the effect of the probabilistic term is more pronounced relative to the loadScore since dividing by small numbers such as the failure probabilities results in values that are larger than the maximum loadScore. Yet, bringing the loadScore down as a result of having severe damage scenarios reduces the overall value significantly as well.

### 2.6.4 Natural Failures GA Experiments

We use NFGA to discover likely natural failure scenarios. For our experiments we set the number of damages N to 0 and explore purely natural failure scenarios. We conducted a set of experiments similar to the ones we made with DCGA. We ran NFGA over five different seeds and observe the results. Our preliminary runs indicated that NFGA requires more generations and a larger population than DCGA, so here we use a population size of 100 individuals and run for 100 generations. The mutation and crossover probabilities remain similar as from their perspective the chromosome structure is virtually the same.

Figure 3 shows the progression of the minimum fitness for each generation for the NFGA run with a seed equal to 1.

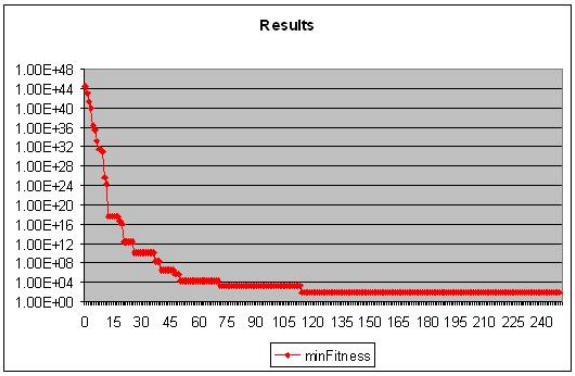


**Figure 3: NFGA progress for seed 1**

For all five seeds we observed a quick improvement in the fitness during the first half of the run and the invariable discovery of the same best solution - an individual with no natural failures. Table 4 lists some of the damage scenarios that NFGA discovered and their respective fitness values.

| Scenario | Damaged components | Fitness |
|---|---|---|
| 1 | None | 58 |
| 2 | PP101B | 1160 |
| 3 | ACP201 | 1160 |
| 4 | V005R | 1530 |
| ... | | |
| 9 | V002S,V005S | 49500.0 |
| 10 | PD4S,V002R,V201B | 162000.0 |

**Table 4: NFGA sample damage scenarios**

This result is counterintuitive at first since our expectation is to find a damage scenario and NFGA gives us a scenario where all components are healthy. A closer look at our fitness function however reveals that NFGA actually did an excellent job.

The reason lies in the fundamental problem that plagues traditional GAs which try to solve multi-objective problems. The purpose of NFGA is to find severe damage scenarios, which are also very likely. To achieve this we put together a fitness function which incorporates both elements - likelihood and damage severity as specified by Equation 1. Yet, NFGA simply found the most likely but least damaging scenario - that of no natural failures and therefore a maximum loadScore of 57.

The more failures, the larger the total sum becomes. Similarly, the larger the loadScore is, the larger the total sum is as well. Yet, for the loadScore to decrease, at least some failure must occur, but

even the most likely failure (MTBF 20) more than compensates for any decrease in loadScore and brings the total fitness value above what it used to be.

The initial improvement is due to the fact that there are so many failures that the scenarios are very unlikely and the fitness value is very high. In addition however, due to the many failures the loadScore is zero - indicating the great damage severity of the scenarios. NFGA proceeds to reducing the amount of natural failures gaining large improvements in total fitness value without incurring any loss from a reduction in severity - whether half or a quarter of all components are failed, the loadScore will still be down to zero. DCGA showed us that only 3 out of 70 components are enough to bring the loadScore down to zero.

This continues up to the point when the total number of natural failures is low enough that removing more failures would actually result in part of the chilled water system remaining operational. Even then however, removing an additional failure reduces the total fitness more than the gain in loadScore increases it as explained above. For that reason NFGA settles with no failures at all and a maximum loadScore of 57, bringing the total fitness to 58.

NFGA is a shining example of the problem of traditional GAs which try to solve multi-objective problems. One can try to balance the fitness function a little better and increase the penalty that comes from the loadScore term when the total number of failures begins to decrease too much, but invariably NFGA will converge to the one specific area of the solution space, which is strictly described by the fitness function. What we really need to expose is the Pareto frontier[2] of non-dominated solutions that show the actual trade-off between the likelihood of a failure scenario and its severity.

### 2.6.5 Natural Failures MOGA

Here we translate NFGA into its multi-objective variant (NFMOGA). The motivation behind this is that we would like to address the issues described in section 2.6.4. NFGA optimizes two objectives - the severity of the damage scenario and its total likelihood. Because the fitness function combines the two metrics NFGA suffers from convergence towards specific parts of the solution space without exposing the Pareto frontier of non-dominated solutions that truly describe the tradeoff between the two objectives.

#### *Objective functions.*

The two objectives that NFMOGA optimizes are the severity of the damage scenario and the likelihood of all indicated natural failures occurring at the same time. We split them in two separate objective functions. For the severity of the damage scenario we use the familiar loadScore, while for the likelihood objective we use Equation 2 below.

$$Likelihood = \log \frac{1}{\prod_{i=1}^{Allfailures} \frac{1}{MTBFComponent_i}} \quad (2)$$

In this formula we take the logarithm with base 10 of the original metric in order to rescale the product. The likelihood objective needs to be minimized in order to get a high probability of all failures occurring at the same time

### 2.6.6 Natural Failures MOGA Experiments

The experimental set-up for NFMOGA follows the same structure as with our other GAs. We use five different seeds and observe the results. We run NFMOGA for 200 generations and use the same mutation and crossover probabilities that we had for NFGA since the chromosome is the same. Here the standard elitism mechanism is irrelevant because NSGA2 [3] achieves that effect intrinsically.

Figure 4 displays NFMOGA's progress for seed 10. The set-up of this figure is different from what we have seen so far. It shows the Pareto frontier of non-dominated solutions for each generation as a set of connected points. Every point represents a specific individual evaluated along the two objectives that NFMOGA optimizes. The y-axis shows the value for the Likelihood objective, while the x-axis shows the value of the loadScore.

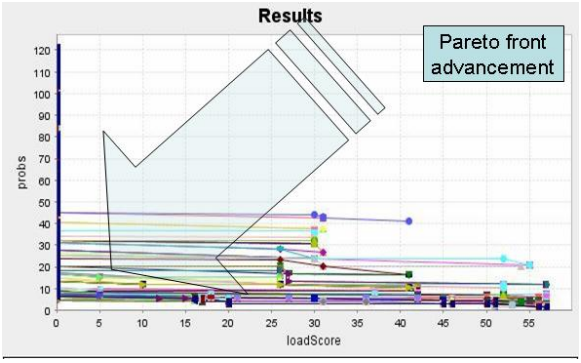


**Figure 4: NFMOGA progress for seed 10**

The goal of NFMOGA is to minimize both objectives, so we see a shift of the Pareto frontier towards the origin as generations progress. Not surprisingly we see most of the individuals spread out along the Likelihood objective and with a zero value for the loadScore objective. The reason for that is the same as it was for NFGA. It is likely that the chromosomes contain more than enough failures to bring the whole system down, especially in the initial generations. This trend is consistent for each of the five seeds that we used.

The difference from NFGA however is that here the prevalence of severe damage scenarios does not hinder the discovery of solutions where the loadScore is not zero. The multi-objective mechanism ensures that the two goals are optimized independently and we can see the range of the tradeoff between Likelihood and loadScore more clearly. Table 5 lists some of the non-dominated scenarios discovered by NFMOGA after 200 generations.

| Scenario | Damaged components | loadScore | Likelihood |
|---|---|---|---|
| 1 | NONE | 57.0 | 0.0 |
| 2 | V122R | 57.0 | 1.4 |
| 3 | V006S | 56.0 | 1.4 |
| 4 | PP201A,V003S | 51.0 | 2.7 |
| 5 | V103S,V603S | 46.0 | 2.9 |
| 6 | V005R,V006S | 45.0 | 2.9 |
| 7 | PC3S,V231R | 20.0 | 3.7 |
| 8 | V121R,V127R,V234S | 16.0 | 4.4 |
| 9 | PB3S,V006R,V231S | 16.0 | 5.2 |

**Table 5: NFMOGA sample damage scenarios**

We ordered the scenarios in Table 5 to show the tradeoff between the two objectives. Clearly with the increase of the Likelihood (smaller values) the severity of the damage decreases (loadScore increases). NFMOGA does a much better job than NFGA at exposing this relationship and helps for a more thorough understanding of the problem.

## 2.7 Design Optimization GA

One of our goals for this project is to demonstrate the use of an evolutionary algorithm for exploring different designs of the chilled water plant system. The first things that come to mind when discussing evolutionary algorithms are search and optimization, however design could be viewed as a kind of optimization as well. As long as we can provide a genetic representation of our designs and a way to evaluate them, we can use an evolutionary algorithm to explore and optimize different solutions.

### *2.7.1 Flowmeter Sensor Location MOGA*

The flowmeter sensor location MOGA (FSLMOGA) searches for robust configurations of the flowmeter sensors in the system, while at the same time it tries to minimize the total number of sensors used. The motivation is to provide an automated mechanism that can reduce the cost of the diagnostic system while it preserves its quality by exploring the tradeoff between these competing objectives.

The sensor model of the system is designed with 16 flowmeter sensors - one for each service in the plant model. FSLMOGA explores different variants of this by trying to use fewer sensors. It essentially removes some of the flowmeter sensors while it keeps the others and searches for the ones that are not vital to the good operation of the diagnostic model.

#### *Genotype and genetic operators.*

The genetic representation for FSLMOGA needs to encode for the presence or absence of 16 different flowmeter sensors. For this purpose we chose to use 16 boolean-valued genes in our chromosome - one for each sensor. A 'true' value in a gene means that the corresponding sensor is present, while a 'false' value means that it is absent. Table 6 shows an illustration of the FSLMOGA chromosome.

| **Chromosome** | **Gene**$_1$ | **Gene**$_2$ | **Gene**$_3$ | **...** | **Gene**$_{16}$ |
|---|---|---|---|---|---|
| Value | true | false | true | ... | true |

**Table 6: FSLMOGA genotype**

There is no limit to how many sensors may be removed at a time. The exploration ranges from setups where there are virtually no flowmeter sensors to systems where all 16 sensors are present.

#### *Objective functions.*

FSLMOGA is trying to minimize the number of flowmeter sensors used, while at the same time it maintains the 'robustness' of the system. We evaluate the robustness of a configuration by running it against a portfolio of selected damage scenarios and evaluating the loadScore metric for each one of them. Next, the difference between the maximum possible score and the resulting loadScore is taken, and the average across all damage scenarios is found. Equation 3 describes the resulting robustnessScore.

$$robustness = \frac{\sum_{i=1}^{Scenarios}(maxScore - loadScore_i)}{numScenarios} \quad (3)$$

This objective practically tries to maximize the familiar loadScore metric across a set of damage scenarios. A sign of a healthy system would be a high loadScore, where as many services as possible are operational. By subtracting the loadScore from the maximum possible score, we convert the problem from maximization to minimization for convenience. The damage scenarios that constitute the portfolio are chosen by the user. They can certainly be evolved with one of our damage genetic algorithms like NFMOGA for example.

The maxScore constant here is equal to 58 instead of 57 when one of the main services is turned off.

The second objective of FSLMOGA is to minimize the number of flow meter sensors used. This objective is directly at odds with the robustnessScore goal, as reducing the amount of sensors translates in poorer diagnoses for the diagnostic model, and therefore smaller loadScore. Equation 4 shows the sensorsScore metric.

$$sensorsScore = Number\ of\ sensors\ used \quad (4)$$

### *2.7.2 Design Optimization GA Experiments*

Here we run FSLMOGA for five different seeds to see if it can discover robust designs with fewer flowmeter sensors. For these experiments we put together a portfolio of damage scenarios that were discovered by another GA that optimized explosion events with an explosion radius of 20 units. Tables 7 and 8 list the damage scenarios that went into the portfolio, while Table 9 lists the values for the standard GA parameters that were used.

| **Scenario** | **X** | **Y** | **Z** | **loadScore** | **TruthScr** |
|---|---|---|---|---|---|
| 1 | 68.5 | 55.6 | 48.9 | 0.0 | 41 |
| 2 | 79.1 | 79.2 | 50.5 | 31.0 | 60 |
| 3 | 35.0 | 43.0 | 49.0 | 31.0 | 63 |
| 4 | 71.4 | 46.4 | 29.1 | 46.0 | 49 |
| 5 | 71.5 | 59.5 | 42.5 | 46.0 | 42 |

**Table 7: Damage scenarios portfolio**

| **Scenario** | **Components** |
|---|---|
| 1 | V003S,PB5S,PB5R, V103S,PA4S,PA4R,V103R |
| 2 | PA1S,PA1R,V003S,V003R, PB5S,PA7S,PB5R,PA4S, V002S,PA7R,V002R,PA4R |
| 3 | PB5S,PA7S,PB5R, PA7R,V103R |
| 4 | V003S,PA4S,PA4R |
| 5 | V003S,PB5S,PB5R,V103S |

**Table 8: Damage scenarios portfolio components**

We chose these scenarios with several considerations in mind. First of all, we picked scenarios that affect the system differently. Some are more confusing, while others are more severe. The total robustnessScore for the portfolio is 27.2 if all scenarios are evaluated with the full set of 16 flowmeter sensors.

| **Parameter** | **Value** |
|---|---|
| Random seeds | 1, 10, 20, 30, 40 |
| Population size | 30 |
| Generations | 30 |
| Mutation probability per gene | 0.10 |
| Crossover probability per gene | 0.10 |
| Explosion radius | 20 |

**Table 9: FSLMOGA simulation run parameters**

Our preliminary runs showed that a smaller population size and a fewer number of generations would be sufficient to see reasonable

convergence towards the true Pareto frontier. In addition, we decreased the mutation and crossover probabilities. The chromosome is built from 16 genes, so these probabilities ensure at least one or two mutation / crossover events per mating which is the typical setting for a standard GA. Figure 5 shows the progress of the Pareto frontier for seed 40.

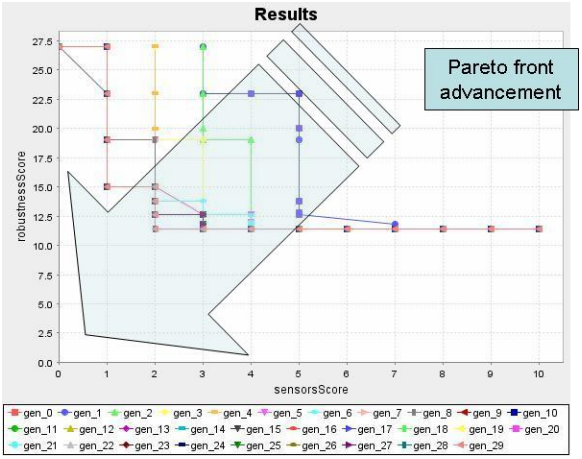


**Figure 5: FSLMOGA progress for seed 40**

The advancement of the set of non-dominated solutions from generation to generation is clearly visible. The shape of the Pareto frontier that FSLMOGA discovers is the same across all five seeds and the solutions that it includes range from a total of 12 flowmeter sensors to none.

According to the results the system can maintain the same robustness that it has with 12 sensors with only 2 sensors. This is an impressive reduction. Furthermore, the robustnessScore is brought down to 11.4 for 2 sensors from 27.2 with 16 sensors.

We begin to see an increase in the robustnessScore if we go down to one or no sensors at all, but even then it does not exceed 27.2, which is a very counterintuitive result. Practically, the system performs just as well without any sensors as with all of them, and performs best with just a subset. This is due to the fact that the accuracy of the diagnosis produced by the diagnostic model depends not only on how many and which specific sensors are available, but also on the specific damage scenario and the default state of the system. Because of the interaction of these three factors, sometimes sensors may provide misleading information and it would be more accurate to just take the default state of the system instead of the measured state. FSLMOGA discovers these inefficiencies for the specific scenarios in the portfolio and discards the misleading sensors, preserving only the essential ones.

For each of the five different seeds FSLMOGA discovers the same set of 2 flowmeter sensors that bring down the robustnessScore to 11.4 just by themselves. It is always the sensors on SP23 (RS02) and SP107 (RS22S) that are kept. Table 10 compares the loadScore and the sensedTruthScore for each of the scenarios in the portfolio when they are evaluated with a system that works only with these two sensors versus a system that works with the full set of 16 sensors.

We see a dramatic increase in the loadScore as well as the sensedTruthScore for three of the scenarios. A loadScore value of 57 means that the system is fully operational, and all services are being supplied with chilled water. It is an interesting result that by removing the right set of sensors such a large improvement can be gained.

| Scenario | loadScr16 | truthScr16 | loadScr2 | truthScr2 |
|---|---|---|---|---|
| 1 | 0.0 | 41 | 57.0 | 59 |
| 2 | 31.0 | 60 | 31.0 | 60 |
| 3 | 31.0 | 63 | 31.0 | 63 |
| 4 | 46.0 | 49 | 57.0 | 61 |
| 5 | 46.0 | 42 | 57.0 | 60 |

**Table 10: Damage scenarios comparison for a system with 16 sensors vs. a system with 2 sensors**

The most compelling example is scenario 1. By removing all sensors but the two FSLMOGA discovered we gain an improvement from a completely incapacitated system to all services operational. This implies that the damage scenario itself did not really destroy the system, but rather resulted in a confusing diagnosis, which in turn prompted the control model to shut all services down. Now that FSLMOGA removed the confusion produced by some of the sensors, the control model takes actions that are appropriate for the situation and the damage is handled without any consequences for the operation of the system.

This is evident by the increase in sensedTruthScore. The sensedTruthScore is a measurement of the accuracy of the diagnosis produced by the diagnostic model. A higher value means that fewer components in the system were misdiagnosed. The maximum value for the sensedTruthScore is 64.

### *2.7.3 FSLMOGA vs. damage GA experiment*

FSLMOGA was able to discover a design for the system that has an improved ability to handle the damage scenarios in the selected portfolio. The natural question that this result provokes is whether this new design is truly better. It could be the case that FSLMOGA simply optimized for the specific set of damage scenarios that were part of the portfolio and any other damage scenario would be devastating. Even worse, it may turn out that reducing the amount of flowmeter sensors to 2 may turn some previously benign damage scenarios into serious threats.

Such concerns are well founded as any optimization or search technique always carries some risk of overfitting the solutions it produces. One way to test whether this is the case is to actually go back to the damage GA and have it discover new damage scenarios for the chilled water system when it is equipped only with the two sensors suggested by FSLMOGA, namely the sensors on SP23 (RS02) and SP107 (RS22S). This time the GA failed to discover a solution that incapacitates the whole system for each of the five different seeds. In addition, the damage GAcould not bring the sensedTruthScore below 55 either. This implies that the design that was discovered by FSLMOGA is fairly robust and is better than the original one.

## 3. SUMMARY OF ANALYSIS AND FUTURE WORK

In this project our objectives were to demonstrate the applicability of evolutionary algorithms to intelligent ship control systems testing and design. For our purposes we developed several genetic algorithms (GA) and multi-objective genetic algorithms (MOGA) that we used to discover damage scenarios for a chilled water plant system and to search for improved configurations of the sensor layout in the system.

As a first step we developed the damaged components GA (DCGA). This algorithm was used to search for damage scenarios that are unrelated in space and incapacitate the chilled water system as severely as possible. Our experiments demonstrated that

DCGA can easily discover sets of three components that if damaged simultaneously can bring the whole system down. Although unlikely, such scenarios are excellent examples of counterintuitive situations that seem benign at first, but actually have the potential to be terminal for the operations of the entire system.

Next we added the notion of likelihood to DCGA in what became the natural failures GA (NFGA). It took into account the likelihood of any given component failing due to natural causes and produced natural failure scenarios that were as likely as possible, but also as severe as possible. Given these two separate objectives, NFGA combined them in its fitness function, which practically established a specific relationship between them. This forced NFGA to discover solutions that were likely, but not very damaging to the system.

To remedy NFGA's problem we were motivated to transform it into a multi-objective GA, which we called NFMOGA. It split the likelihood and the severity of a natural failures scenario in two separate objectives that were optimized independently. This helped us expose the actual tradeoff between the two objectives and to examine a Pareto frontier of non-dominated solutions. We exposed the clearly inverse relationship between the likelihood of a scenario and the level of severity that it has.

This led us to our final step for which we created the flowmeter sensor location multi-objective GA (FSLMOGA). It is our take on creating an EA that explores different design options for the sensor placement in the chilled water plant. FSLMOGA used a portfolio of damage scenarios generated with a GA to evaluate its designs. During our experiments with FSLMOGA we discovered that it is possible to use only two out of sixteen sensors, which perform better than a system equipped with the full set of sensors installed.

FSLMOGA optimized the average operational level of the system across the portfolio of damage scenarios as well as the total number of sensors used in the system. We managed to get significant gains in survivability while reducing the total amount of sensors down to two out of sixteen.

To validate these results we conducted a final experiment where we took the two-sensor design produced by FSLMOGA and we fed it back into the damage GA in an attempt to discover new damage scenarios that can break it. While our results revealed that there are damage scenarios that can bring a good portion of the system down, the damage GA failed to discover a damage scenario that completely incapacitates the system. In addition, the damage GA was unable to discover damage scenarios that lead the diagnostic model to misinterpret the state of more than 10 components.

This led us to believe that there are some inefficiencies in the diagnostic system, which can be remedied by removing some sensors and preserving others that are key to its proper operation. In addition, the process of going back and forth between a 'damage' GA and a 'design' GA resembles an arms race process that can be used to evolve ever better designs that can handle progressively adapted damage scenarios. Such a mechanism can be automated into a coevolutionary process which evolves damage scenarios and system designs in parallel as separate species.

Competitive coevolutionary algorithms have been developed for many different applications and are suggested as an avenue of future work for this project. We believe that there is much to be learned about the robustness of the chilled water plant in a coevolutionary environment, especially if the system is rid of its inefficiencies and more accurate simulation models are employed.

By using a sequential approach of going back and forth between two separate algorithms like the damage GA and FSLMOGA the user runs the risk of settling at local optima for the design and the possible damage scenarios. On the other hand, by maintaining two populations in parallel and evolving them simultaneously, a coevolutionary algorithm has a much greater ability to explore the full specter of possibilities and arrive at globally optimal solutions both for designs and damage scenarios. The intrinsic interaction between the two and the diversity offered by maintaining entire populations instead of small portfolios remedy the inherent bias and premature convergence issues of a sequential approach.

# 4. ACKNOWLEDGMENTS

The authors would like to thank the Office of Naval Research for funding portions of this research under ONR Contract N00014-05-C0061.

# 5. REFERENCES

[1] C. Anderson, E. Bonabeau, and J. M. Scott. Evolutionary testing as both a testing and redesign tool: a study of a shipboard firemain's valve and pump controls. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 1, pages 1089–1097, USA; Piscataway, NJ, 2004. IEEE; IEEE Neural Network Soc.

[2] A. Belegundu and T. Chandrupatla. *Optimization Concepts and Applications in Engineering*. Prentice-Hall, Inc, 1999.

[3] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast and elitisit multi-objective genetic algorithm: Nsga-ii. *KanGAL Report 200001, Indian Institute of Technology*, 2000a.

[4] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1):1–16, 1995.

[5] J. H. Holland. *Adaptation in natural and artificial systems*. MIL University of Michigan Press, Ann Arbor, 1975.

[6] L. A. Rossman. Epanet, users manual, 1994. howpublished: Risk Reduction Engineering Laboratory, Office of Research & Development., U. S. Environmental Protection Agency, Cincinnati, Ohio.

[7] B. Williams and P. Nayak. A model-based approach to reactive self-configuring systems. In *AAAI*, 1996.