Evolutionary testing as both a testing and redesign tool: a study of a shipboard firemain's valve and pump controls

Carl Anderson, Eric Bonabeau, John M. Scott
Icosystem Corporation, 10 Fawcett St., Cambridge, MA 02138
Email: {carl, eric, johnscott}@icosystem.com

***Abstract*-Given the large number of possible fault scenarios, complex control systems are usually impossible to test exhaustively. One promising approach, however, is evolutionary testing (ET) in which a genetic algorithm (GA) evolves critical test situations. We evolved challenges (pipe rupture and water demand) to the valve and pump controls of a shipboard firemain system. ET found minor events that collectively produced significant systen failure, and also identified a modification to the ship design that improved system performance. ET can also be used to map the boundary of search space for a given, user-defined criterion. The potential of this powerful, generic technique not just for testing but also for system redesign is thus emphasized.**

I. Introduction

Evolutionary testing could play a far more important, prominent role in the testing and design of future complex control systems. In such complex systems, which may involve many components, each with multiple states, and with correlated or interdependent activity among those components, the number of possible fault scenarios is so large that systematic, exhaustive testing is usually impossible.

One approach that is often employed is to test each component individually to assess their relative importance or sensitivity. However, when interdependencies exist among components, and especially when such interdependencies are unclear or unknown, it is all too easy to overlook some scenario of individually insignificant events that interact and cascade to "break" the control system—consider the massive power outage in north America in 2003 that affected a staggering 50 million people, a result of several, separate and relatively minor events. So, if systematic testing is infeasible and another obvious approach, random testing, is inefficient, what other options are available?

One powerful approach, one that shows great promise but is woefully undertudied, is evolutionary testing [1–6]. In evolutionary testing, evolutionary computation, such as a genetic algorithm [7], is used to *evolve* system challenges, thus harnessing the power of a distributed, parallel search. The goal is to "hill climb" *poor* system level performance and so seek out combinations and scenarios that would otherwise have been overlooked and likely would never have occurred to the engineers or scientists that created the control system.

ET was used to test the local controls that open and close valves and simple controls that turn pumps on and off on a ship's firemain—a hydraulic network of pumps, valves, pressurized pipes and sprinkler heads. Our goal was to study the potential of ET for testing hydraulic systems, as well explore and advance the ET approach and tools more generally. Keeping the valve and pump control logic fixed, the genetic algorithm was used to evolve two types of challenges:

- ***extrinsic challenges***: pipe rupture, e.g. a result of enemy attack, and
- ***intrinsic challenge***s: water drawn off the system, e.g. for ballasting.

Ideally, the valve controls should isolate sections of ruptured pipe, thereby stemming local flooding, while maintaining flow to the sprinkler heads and any junctions where water may need to be drawn off, e.g. to fill ballast tanks and prevent list. Also, pumps should shut off if they are pumping into "dead ends" or to ruptured pipe.

There are several important interdependencies in the system. For instance, sprinkler head outflow is a function of hydraulic pressure which, in turn, is a function of water demand that may occur in far-off sections of the network. Also, valve closure can both divert and increase flow to different sections of the ship while curtailing it to others. Thus, this is an interesting problem and model in its own right. However, one motivation of this study was the methodology itself.

As is demonstrated, there are other ways in which one can harness the power and utility of ET and, importantly, how engineers can get directly involved and easily tailor the ET process to suit their specific needs or interests. We introduce a novel aspect of of using ET, "threshold search," in which the engineer can customize the GA's search to seek component-specific failure modes, such as excess pump speed, and highlight weak spots in the system design. Further, we demonstrate that engineers can indeed use the GA's results to help suggest where the system should be modified, and in our simple example, it

0-7803-8515-2/04/$20.00 ©2004 IEEE

does lead to enhanced system performance. Hence, ET could be a powerful tool in the development cycle, not just in *testing* extant designs but also helping engineers in the *redesign* process.

II. Model And Methodology

The model is a relatively realistic scale model of the *USS Arleigh Burke* (DDG–51), the lead ship of the ARLEIGH BURKE class of a guided missile destroyers.

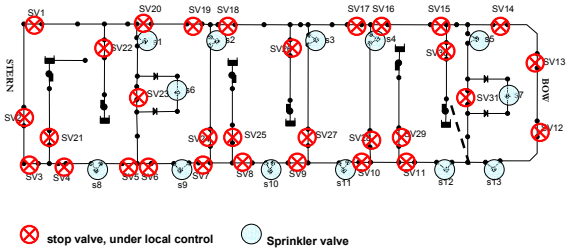

Figure 1. Schematic of the firemain system, with 31 stop valves (⊗) and 6 pumps, each under local control, and 13 sprinkler heads. The bow is to the right. Potentially, all pipes between these stop valves can be ruptured (= 63 locations). (Dashed line: see section III.D.) Scale: approximately 100 ft. × 400 ft.

A. Modeling the USS Arleigh Burke

Starting with the *Arleigh Burke*'s firemain schematic given in Wilkins [8], public domain sources such as the U.S. Navy's website were used to supplement this with more specific information such as the ship's dimensions, typical pump pressures, firemain diameters etc.

The analysis was restrictd to 13 sprinkler heads (shown as s1–s13 in figure 1) evenly spread across the firemain. The manually operated valves shown in Wilkins' schematic [8] were not implemented but 31 autonomous stop valves were (shown as SV1–SV31 in figure 1), each with local, controlling logic, closely matching the locations in the real schematic.

There are six pumps, each operating with the following head-flow pump curve: head (ft.) = 533.34 – 0.0001334 × flow $(\text{gpm}^{-1})^2$. This produces around 230 pounds per square inch (psi) pressure during "normal" operation, comparable to the real ship.

1) Network challenges: two types of network challenges were modeled and evolved:

- *pipe rupture*, a simulated hole with an emitter coefficient of two (that is, outflow (gpm) = 2 × pressure $(\text{psi}^{-1})^{0.5}$) that may occur in any of 63 locations, each representing a between-valve pipe;
- *water demand*, a constant outflow in gallons per minute (gpm) from a junction, which may occur at any of 49 locations around the ship.

2) Valve and pump controls: the valve and pump control logic consists of a simple but reasonable scenario: if a pipe ruptures, every "local" stop valve closes that is necessary to isolate that broken pipe from the network. Thus, in the simplest case, if a pipe ruptures, two stop valves, one at either end of the pipe, close to prevent any flow into that rupture location which would cause local flooding (and possible listing of the ship). In more complex cases, e.g. where a rupture occurs close to a T-junction, three or four stop valves may need to close. In addition, control logic is also implemented on the pumps. If the other control logic closes a stop valve on the riser from the pump, it is futile to attempt to pump water into a "dead end," thus the pump automatically shuts off. For simplicity, all sprinklers are kept open and water flow through the whole network was monitored.

3) Implementation: To implement the underlying hydraulic model and the valve and pump controls, EPANET [9], a public domain, pressurized pipe network simulator from the U.S. Environmental Protection Agency, was used.

To introduce our evolved challenges to the model ship, an existing C++ library, OOTEN (Object-Oriented Toolkit for EPANET; [10]) was further developed, which acts as a "wrapper" to an underlying EPANET simulation, facilitating run-time changes to network components as well as network data acquisition, e.g., pipe flows and pressures, and valve states.

B. The Genetic Algorithm

Each genotype, i.e., a scenario parsed to an EPANET simulation, consists of two genotypes: an *r-chromosome*, a bitstring that encodes rupture (or not) at each of the 63 pipes, and a *d-chromosome*, which encodes water demand (or not) at each of the 49 possible locations. Individual i's r- and d-chromosomes are denoted as r_i and d_i respectively and the j^{th} gene of these two chromosome as $r_{i,j}$ and $d_{i,j}$.

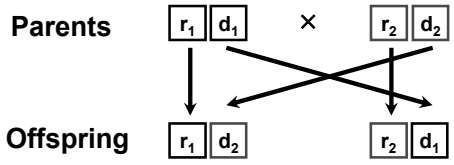

Figure 2. Chromosomal crossover: parents r_1d_1 and r_2d_2 produce two offspring r_1d_2 and r_2d_1.

1) Genetic Operators: Unusually, the GA uses three genetic operators:

Chromosomal crossover: when two parents r_1d_1 and r_2d_2 are crossed, they produce offspring r_1d_2 and r_2d_1 (figure 2).

Genic crossover: with system performance as the minimand, if left unconstrained, the GA would quickly evolve the worst case scenario: all pipes ruptured and all water siphoned ($r_{i,j} = d_{i,j} = 1\ \forall$ i, j). Thus, the number of pipe ruptures, $N_r \langle\langle 63$, and the number of water demands, $N_d \langle\langle 49$, were held constant. As uniform crossover cannot guarantee constant N_r and N_d, only on overage, the traditional GA crossover operator had to be modified.

Consider two parent r-chromosomes, r_a and r_b. For each locus, j $\in$ {1, 2, …, 63}, the gene values of the two parents are compared. If they match, that is, $r_{a,j} = r_{b,j}$, then swapping is futile: offspring cannot differ from their parents with respect to gene j. Therefore, only those locations in which they differ ($r_{a,j} \neq r_{b,j}$) were considered. There are thus two possibilities for each of these locations: 1) $r_{a,j} = 0$ and $r_{b,j} = 1$, which is termed a –type locus. 2) $r_{a,j} = 1$ and $r_{b,j} = 0$, a +type locus. Importantly, whatever the length of the chromosome, the parity of N_r and N_d, and the number of locations with matching gene values, the number of –type and +type loci will be *equal* (so long as N_r is the same for both parents). Therefore, all loci are randomly paired (without replacement) so that one locus of the pair is a –type locus and the other locus is a +type locus (thus, this can be considered a form of random genetic linkage). Finally, for each of these ±loci pairings, with probability 0.5 the gene values are swapped (0→1 and 1→0; figure 3).

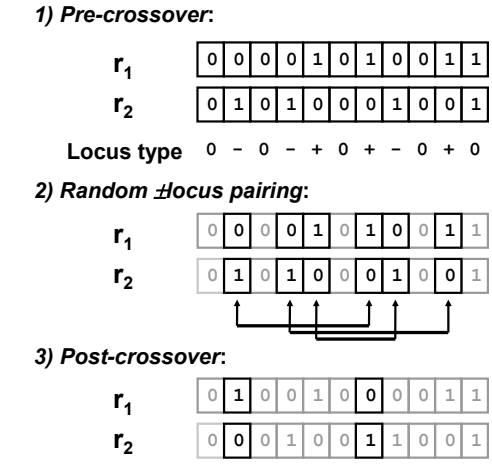

Figure 3. Genic crossover of two r-chromosomes. Step 1) the locus types are identified (see text). Step 2) each -type locus is randomly paired with a +type locus. Step 3) for each of these ±loci pairings, with probability 0.5 all four gene values are swapped (0→1 and 1→0) thus maintaining constant N_r and N_d. Here, only the 2nd / 7th locus pair were swapped.

Mutation: the objective of the mutation operator is to take an existing rupture and preferentially rupture another nearby location instead. That is, a symmetric proximity matrix $\mathbf{M_r}$, whereby each entry $\mathbf{M_{ri,j}}$ specifies the physical distance (in 2D Euclidean space) between two rupture locations *i* and *j*. An existing rupture on the r-chromosome is selected at random, i.e., some j where $r_{i,j} = 1$. A vector

$$W = 1/\mathbf{M}_{\mathbf{r}i,j} \Big/ \sum_{i=1}^{49} 1/\mathbf{M}_{\mathbf{r}i,j}$$

specifies normalized weightings that are inversely proportional to the Euclidean distance between rupture locations i and j. *W* is used to select a new rupture location, say *k*, probabilistically. We then set $r_{i,j} = 0$ and $r_{k,j} = 1$, thus preferentially rupturing a nearby location instead of j (figure 4).

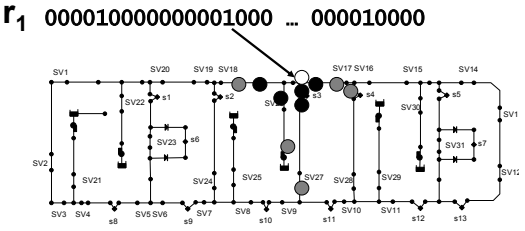

Figure 4. Mutation of genotype 1's r-chromosome. Suppose that the chosen gene (2nd gene from left with value 1) encodes a port beam location (open circle). Mutation preferentially ruptures a nearby location in place of the focal rupture. Thus, the black circle locations are far more likely to be chosen for rupture than those slightly farther away (grey circles). The actual probabilities are inversely proportional to the physical distance from the selected rupture (open circle).

Similarly, we mutate the d-chromosome using an appropriate proximity matrix, $\mathbf{M_d}$, that gives the physical distances between the water demand locations. It is primarily for this reason—that ruptures and demands involve different proximity matrices—that we implemented two chromosomes per genotype.

2) Fitness Functions: two metrics were used, chosen because they are continuous, simple and sufficiently variable over state space, both of which are maximands (i.e., the GA seeks worst performance):

Dead water length: the total length of pipe, in feet, with zero flow. This includes pipe that is isolated from the pumps due to closed stop valves as well as "dead ends," i.e., sections with no outlet.

Maximum pump flow: the maximum flow, in gallons per minute, of the six pumps.

3) Creating the next generation: each generation, a population of *n* genotypes is produced, each run on a separate EPANET simulation, and their fitnesses calculated. Elitist selection was used to create the next generation: *e* elites were chosen to pass unaltered to the next generation while the remaining *n* – *e* slots were filled by selecting individuals from the current generation

probabilistically, weighted by their fitness, and then applying the genetic operators.

For simplicity, weightings were calculated using a normalized selection pressure. Suppose the i^{th} genotype has fitness f_i. The range of fitnesses $R = \max\{f\} - \min\{f\}$. A linear transformation was performed such the minimum is zero (that is, $\min\{f\}$ was subtracted from each f_i, so that all fitness range from 0 to R). [If one is *minimizing* the fitness metric, an additional step here must be performed here: subtract all fitness values from R.] Finally, $c \times R$ was added to each fitness where $c \in (0, \infty)$ and is a constant (hence all fitness range from cR to $(1 + c)R$). Selection occurs probilistically from these transformed fitnesses where c determines "selection pressure."

After an individual had been selected for the next generation, the three genetic operatiors were performed with a fixed, independent probability, but never on the e elites.

III. Results

Table 1 specifies the parameter values, essentially chosen abitrarily, usually used in the simulations.

TABLE I. USUAL PARAMETER VALUES USED

Parameter	Value or Range
Population size, n	50
Elite pool size, e	10
Number of generations	100–300
Number of ruptures, N_r	1–5
Number of demands, N_d	0–2
Pr.(chromosomal crossover)	0.5
Pr.(genic crossover)	0.5
Pr.(mutation)	0.5
Selection pressure, c	1
Emitter coeffcient	2
Water demand (gpm^{-1})	500

A. Validation of the Genetic Algorithm

The GA had first to be validated. The most obvious, rigorous and effective manner to do this was systematic testing. That is, for a given N_r and N_d , run all possible rupture-demand scenarios and so find the worst case scenario(s). The GA was then run to test that it evolved this same worst case (or cases), and also to check its covergence rate.

First, the relative importance of each of the 63 rupture locations was checked (i.e., $N_r = 1$; $N_d = 0$; figure 5). Thus, for example, several pipe ruptures on the starboard beam (center of the lower margin of figure 5), caused around 110 feet of pipe to have zero flow. It is clear that there is a great deal of variation around the ship with some locations having relatively little effect, e.g. those on the starboard beam, while others have very large effects, e.g. portside bow and quarter. There is not one single worst rupture location but four that are equally bad.

Significantly, while evolving a single rupture ($N_r = 1$; $N_d = 0$), the GA found and retained *all* of these four worst rupture locations in its population, an important advantage of having a relatively large elite pool in this GA.

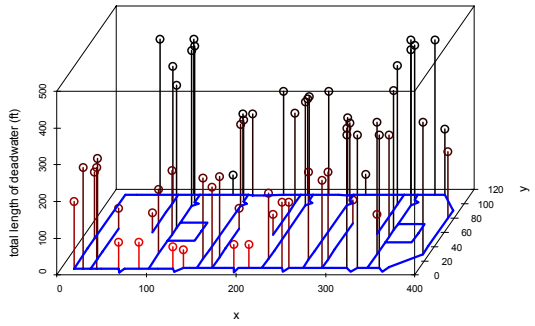

Figure 5. Dead water length (in feet; z-axis) when a pipe was ruptured in each of 63 locations (x- and y-axes). Projected onto the floor of this figure is a schematic of our hydraulic network.

All dual ruptures were then systematically explored ($N_r = 2$; $N_d = 0$). Once again, there is a huge amount of variation, in this case generating 16 "equally worst" combinations. The GA quickly optimized the dead water length to find the same worst case scenarios. The population at the end of 100 generations contained 6 of the 16 worst scenarios.

Finally, all triple ruptures were investigated ($N_r = 3$; $N_d = 0$). Here, the state space starts to become large ($63 \times 62 \times 61 = 238266$ combinations) and while the simulation took several hours, the GA found the same maximal dead water length in a few minutes—it took a median of just 49 generations (35 replicates) to achieve this, thereby only searching ≈1% of state space. Hence, it was demonstrated that without tuning any of the parameters, the GA found not one but multiple scenarios that were truly worst case, hence validating its convergence behavior.

B. Attributes of Worst-Case Triple Ruptures

Are these worst three-point cases a combination of the worst single point ruptures? In other words, is damage and effects additive or are there some more interesting and complex interactions? The results were very surprising. There were 12 equally worst scenarios: one rupture from a port quarter group of four, plus one from a starboard beam group of three plus one port-bow location, r56 (figure 6).

While the four locations in port quarter group consisted of the four *individual* worst rupture locations (i.e., for simulations where $N_r = 1$), the second group consisted of locations that could be considered far more insignificant. That is, as individual rupture locations these were ranked only 28^{th}–30^{th} in the list of 63 (they are the 13^{th} worst *group* of equally bad ruptures). The third location constituted the 5^{th} worst group which was 11^{th} worst of the 63.

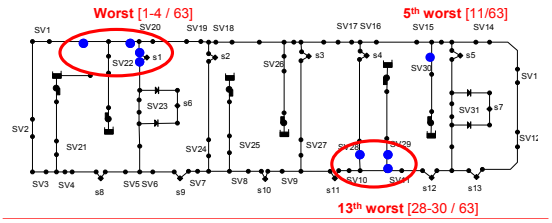

Figure 6: Structure of the 12 worst three-point ruptures. Each case consists of one rupture from the top-left (port quarter) group, one from the lower group (starboard beam) and the rupture location in the top right, r56 (port stern). While those in the top left were all locations that, individually, were the worst single point ruptures, those in the lower group were low down in the list: ranked only 28–30th worst of the 63 locations. The forward most rupture was ranked 11th of 63, comprising the 5th worst group of rupture values.

In summary, all of the worst three point ruptures involve locations that, individually, could easily be overlooked and might be deemed insignificant (those of the starboard beam group). In other words, these results suggest that with a spatial, interconnected system such as this ship's firemain system, an analysis of individual locations, and controls to deal with them, may be insufficient, thereby validating the evolutionary testing approach.

C. Favoring low-ranked components

These cases that involve low-ranked individual ruptures are clearly the important scenarios that engineers are most likely to overlook using traditional techniques. How then might one favor such results in the GA?

A simple yet powerful multiobjective function was designed that balances maximizing deadwater length but at the same time maximizing the sum of the ranks of the individual ruptures. That is, from the systematic investigation of individual ruptures (section III.A), a rank was assigned to each rupture location, with lower value ranks meaning *higher* deadwater length. As there are 63 rupture locations, the ranks range from 2.5 (the top four locations with equal deadwater length that cause most damage) to 63 (the single location that causes least damage). Let v_i represent the rank of rupture location i. For any given rupture scenario, i.e., a set of locations {j}, the ranks of those rupture locations:

$$S = \sum_{\{j\}} v_j$$

were summed. The GA's new fitness function is then:

$$f = S^{\alpha} \times \text{dead water length}.$$

That is, when α is high, S^{α} becomes the dominant term and the GA preferentially returns scenarios that inolve high v_i locations, but *at the expense of reduced dead water length*. Conversely, low α favors maximizing dead water length, and hence will likely invlove low v_i locations but *at the expense of lower S*. Importantly, for a fixed α, this is a true multiobjective function in which the GA essentially evolves the "biggest bang for the buck": the highest deadwater length for the least significant (highest v_i) locations; moreover, this tradeoff can be easily tuned by varying α.

Table II demonstrates this tradeoff and the utility of using this multiobjective fitness function. Each row represents the final result after a GA run of 100 generations. The table gives the final fitness value, *f*, the deadwater length (DWL), sum of ranks, *S*, and the individual ranks for the fittest genotypes, ie the 5 ruptures. It is clear that this multiobjective function works are required: *S* and α are positively correlated, and DWL and *S* are inversely correlated. Also, the individual ranks of the ruptures in the fittest genotype are high, e.g. involving ruptures with ranks 15–58 for α = 1, but ranging from 46–63 for α = 3. α = 0 is the control, i.e. the "normal" single-objective fitness function such that *f* = deadwater length, and contains one of the most significant ruptures (rank 2.5).

D. Threshold Search

A different, very powerful, and as far as we are aware, novel aspect of evolutionary testing borne out from our GA is what we term "threshold search." In many

TABLE II. FITNESS (*f*), DEAD WATER LENGTH (DWL) AND SUM OF RANKS (S) VERSUS. α. FOR EACH α, INDIVIDUAL RANKS THAT COMPRIZE THE FITTEST GENOTYPE ARE LISTED.

α	***F***	***DWL***	***S***	Individual ranks (v_i's)				
0	1751	1751	98	2.5	11	19.5	29	36
1	237971	1141	208.5	15	39	46	50.5	58
1.5	3.43×10^{6}	1086	223.5	15	46	50.5	54	58
2	5.30×10^{7}	807	256.5	39	46	50.5	58	63
2.5	8.90×10^{8}	689	267.5	36	50.5	58	61	62
3	1.42×10^{10}	650	267	46	50.5	58	62	63

situations of evolutionary testing where the objective is to "break" the system, we may be interested in mapping out the boundaries of the state space: what regions of state space exist in which this system works, but beyond which the system fails?

More detailed, an engineer may be interested in mapping out three regions of interest. The first, in which all pumps work within particular user-defined safety limits; second, a broader region in which pumps may operate, say at maximal speed or pressure (e.g., for emergency situations and short periods only), and finally a third region in which conditions exceed component limits. Thus, for sake of argument, we might decide upon several thresholds of interest with respect to the pumps:

- > 1000 gpm: overheating
- > 1250 gpm: warning (emergency operations only)
- > 1500 gpm: dangerous (untenable operation)

Now, we can set these thresholds in the GA and allow it to search state space. Whenever it encounters a scenario whose fitness exceeds the relevant threshold, it keeps a copy of the genotype (and associated fitness) in a list. At the end of the simulation, we have a crude map of the boundary for a user-desired threshold. To prolong the GA's search and to avoid convergence, we can easily set high values (say, 1) for the crossover and mutation probabilities. The key point here is that the GA is doing more than searching state randomly; in a sense, it is efficiently feeling its way around the user-defined boundary wall.

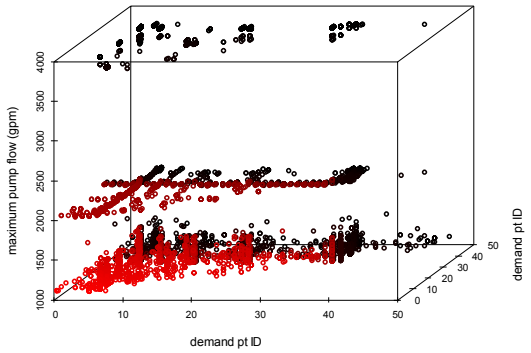

Figure 7. Maximum pump pressure (z-axis) versus ID of the 2 demand location (x- and y-axes) for a threshold ssearch with threshold 1000gpm (see text). Here, one can see significant structure in the results, as well as the large maximal pump flows.

We ran a simulation of 2 ruptures, 2 demands of 2000 gpm each, and a single threshold of 1000 gpm for pump flow, with the GA fitness metric as maximum pump flow, and values of 1 for the three crossover and mutation probabilities. The surprising results are shown in figure 7. First, there is significant structure in the results (middle "layer" of figure 7) that shows where water should *not* be drawn off during situations where ship damage may be occurring. Second, the upper "layer" at 4000gpm are results in which the 2 ruptures isolate a pump *without shutting it off* and from which the 2 demands draw water.

E. Evolutionary Testing in System Redesign

Our analysis of three point ruptures above identified a problematic pairs of ruptures: any of the starboard beam group with rupture location r56 (figure 8). Together, they cause a huge forward section of the ship's firemain to become isolated. A glance of the schematic suggests that an extra riser from the forward pump (pump #6) to the starboard beam should improve the situation. Thus, a riser was added as indicated by the dashed line in figure 1. However, this alone will not alleviate this worst case situation.

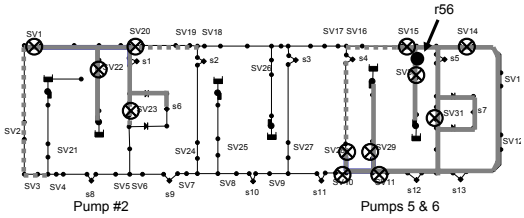

Figure 8. The worst case three point ruptures all cause three pumps to shut off (pumps 2, 5 & 6), 12 stop valves to close (indicated by ⊗) and large sections of pipe to become isolated (solid line) or otherwise have zero flow (dashed line). Rupture location r56 is indicated (● and arrow; see text).

This is because when r56 is ruptured, stop valve S30 closes which, in turn, causes pump #6 to shut off, thereby stemming any flow into the new riser. Thus, the control for pump #6 must also be removed.

It is clear that this new design must outperform the old ship in this worst three point rupture case. However, how did it fare over *all* possible 238266 three point rupture scenarios?

A systematic analysis reavealed that, on average, this new riser did indeed significantly decrease deadwater length (N = 238266 pairs; one-tailed, paired Wilcoxon rank sum test. $V = 43920654$; $p < 10^{-15}$). A histogram of differences is shown in figure 9 indicating a mean reduction of nearly 14 feet of dead water.

As there is more water in the system with the same number of pumps, there is a 10 % mean increase in maximum pump pressure which, statistically, is significant. However, as this is only from 78.4 to 86.5 gpm, which is well with normal operating limits (≈1500 gpm maximum), the additional riser and modified pump controls do improve overall performance, at least across all possible three point ruptures but likely across other damage scenarios too.

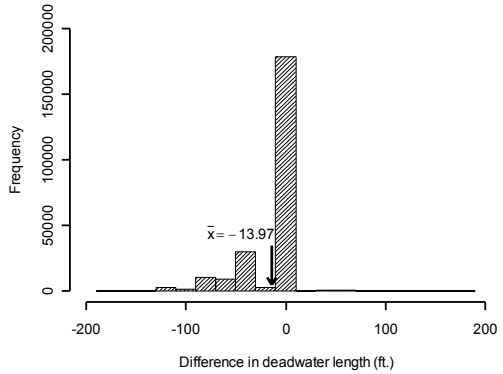

Figure 9. Difference in deadwater length between "normal" ship configuration and that with the extra riser and no pump 6 control. Negative values indicate better performance with the new ship design (mean = –13.97 ft., median = 0 ft.; N = 238266).

In summary, it was shown that the GA can very quickly evolve multiple, worst case three point rupture scenarios. This helps the engineer to identify problematic features of the system design and so aid in the redesign process. Although systematic analysis was used to demonstrate that this modified design was indeed a significant improvement, the deadwater length for the worst case scenario was slightly lower with the new design. Thus, an engineer could have used the GA to check that the maximum values were lower and thus test that the new design was superior.

F. Correlated Damage

Thus far, the rupture locations have been unconstrained spatially, and the GA generally produces worst case scenarios that involve rupture locations that are spread across the ship. (This makes sense as one can imagine a "sphere of influence" around each rupture location, that is, a set of pipes and stop valves that are affected by that rupture. If ruptures are too close, then these sets of influences will overlap and the fitness metric will be submaximal.) However, damage may be correlated. That is, if a torpedo damages a bulkhead, not one but several pipes may rupture in the vicinity. Thus, the GA was modified to take this spatial correlation into account.

The GA was set to evolve a single rupture and 2 demands (N_r = 1; N_d = 2). The way that the model transfers the information stored in a genotype to an EPANET simulation is to write out a list of ruptures to a file. Before this file was passed to EPANET, however, some additional ruptures were added that were physically close to the one that the GA had selected. That is, to evolve a cluster of n ruptures, the single rupture specified by the genotype was taken and the $\mathbf{M_r}$ proximity matrix (Section II.B.1) was used to add the physically closest n – 1 rupture locations to the rupture list. Therefore, one can easily evolve the position of a cluster of ruptures, of a desired size, without modifying the GA.

TABLE II. EFFECT OF SPATIALLY CLUSTERED DAMAGE: DEADWATER LENGTH (COLUMNS 2 & 3) FOR CLUSTERED (CL) AND NON-CLUSTERED (NON) DAMAGE, AND SIMILARLY FOR MAXIMUMUM PUMP FLOW (GPM; COLUMNS 4 & 5).

#	Deadwater length		Maximum pump flow	
ruptures	CL	NON	CL	NON
1	424.0		80.3	
2	494.7	845.9	80.3	147.2
3	561.0	1158.2	96.5	260.1
4	561.0	1470.0	96.5	263.2
5	672.3	1751.2	118.2	376.5

Table II shows evolved results for 1–5 ruptures, plus two unclustered water demands, that were either clustered (columns 2 & 4) or unclustered (columns 3 & 5; excepting that for a single rupture) and where the fitness function was either deadwater length (columns 2 & 3) or maximal pump flow (columns 4 & 5). Clearly, for both metrics, non-clustered damage caused significantly greater damage than clustered damage.

Moreover, the makeup of worst clusters (recall that the GA evolves worst case scenarios) is especially insightful. As cluster size increases, the clusters do not grow in one location but "move" around the ship. In other words, the worst three point cluster is not a simple matter of adding one nearest rupture to the worst two point rupture, but is a whole new three point cluster on an entirely different part of the ship (figure 10). (As might be expected, these worst cases clusters each straddle at least one stop valve, thus maximizing damage.) In some sense, one can consider that, under attack, the most vulnerable section of the ship is not fixed but dependent upon the magnitude of the damage, and thus highlights the significant challenge for the designer and the significant utility of evolutionary testing.

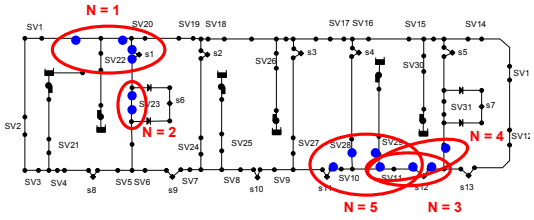

Figure 10. Location of the worst clusters as they grow in in size. For single ruptures (N_r = 1) any of the four in the aft quarter group are equally worse. For 2 ruptures (N_r = 2), the worst cluster is aft midship. For $N_r \geq 3$, the worst clusters are port bow, but still move with increasing N_r.

IV. Discussion

The GA, and more generally, the evolutionary testing approach, clearly worked well. Not only did it find true worst case scenarios, at least for the N_r-N_d values tested,

but it did so in a fraction of the time and a fraction of the state space compared to systematic testing. (Of course, like all evolutionary approaches, we cannot guarantee optimality.) Moreover, in this particular model there were multiple worst case scenarios which the GA found. That is, the GA did not hone in on any one single scenario but rather found and retained many different worst case scenarios in its population, presumably a result of the relatively large elite pool size[1].

Although the model is rather simple, especially the valve and pump controls, it is significant that the GA found three point rupture cases that consisted of ruptures that, individually, could be deemed insigifnicant, or at least were relatively low in the rankings. Moreover, the multiobjective fitness function successfully biased the GA's search to seek out such "obscure" results and its bias was easily tunable by the single parameter α. This is a key facet in the power and utility of evolutionary testing: the ability to seek out surprising, counterintuitive yet significantly large fault scenarios. If such results exist in a simple hydraulic model like this, what might one find in real, more complex systems such as chemical factories, nuclear reactors, and space shuttle controls, all systems that really must work to specifications under all circumstances.

For a given design with a given fitness function, the GA will literally find fault with the system where possible. However, from the engineer's perspective to make this a more powerful tool, he or she should be able to tailor the GA to meet their specific needs more closely. One obvious way is through the choice of fitness function. On top of that however, one can overlay "threshold search."

Threshold search allows the engineer to incorporate component specifications, local building codes and regulations and so on to constrain, filter, and mold the search. This could be especially useful in a team environment where several engineers, each with a different perspective and particular area of expertise, must work together to design a system; for instance, a piping engineer may use the GA test that the model's water pressures are safe, while another may focus on pump speed. At its simplest, threshold search collates cases whose fitness exceeds some user-defined threshold, thereby allowing engineers and other experts to get more involved in the testing process.

Moreover, threshold search harnesses the GA's search capabilities to best advantage to reveal underlying structure of the fitness landscape in such a way that can be directly mapped back to the physical structure of the ship—patterns of failures and key failure combinations almost jump out of the data. Thus, it would be trivial for an engineer to identify the key demand point locations highlighted in figure 7 and modify damage control protocols so that water is never drawn off these locations, or at least never when other damage is likely to occur.

By finding worst case scenarios, the GA can help the engineer to pinpoint weak spots in the design. They can then use their expertise to redesign the system and of course use the GA again to retest the new design. Thus, evolutionary can form an integral part of the system development process. However, the evolutionary testing process outlined here can be easily be extended to increase its utility in such a design process. For instance, suppose that the engineer has identified a problematic aspect of the design but due to component interdependencies may not fully understand *why* that design aspect is problematic. One approach is to fix those particular problematic component states—say rupture locations r56 and a starboard bow location—and use the GA to evolve scenarios around that configuration (in other words, to fix certain gene values of all genotypes). Our ship model is simple enough to understand why how these two ruptures work together but in other systems this may not be so. Such "knowledge embedding" has proved useful in other interactive evolutionary computation studies [11,12].

In some systems, it may be possible to automate the redesign process. If so, it might be feasible to link a GA that tests a system—that is, one that maximizes a particular fitness function—with a second system, that may even be a GA itself, that modifies system design, perhaps to minimize the same fitness metric. Open-ended search or evolutionary design [13,14] could be perfect for such a second system, and linked in such a manner could generate a powerful co-evolutionary arms race [14–17] that evolves worst case test scenarios while simultaneously evolving system design counter strategies. The systems would have to be suitably constrained (for instance, the ships's captain would not want a large pipe running across the bridge floor), but in many circumstances could conceivably work very well.

Finally, for many complex systems, engineers may not necessarily know in advance what sort of fault scenarios may arise in their systems—hence, catastrophic events, such as the 2003 power outage occasionally occur—even with complete knowledge of each components and all interaction types. Thus, interactive evolution [15,18–20] may prove useful. That is, the engineer is presented with a panel of variants and chooses one or more for mating in the next generation, thus biasing and directing the GA towards certain areas of state space. Further, knowledge embedding [11,12], as described above could be incoporated to investigate state space around certain system configurations. Overall, it is clear that evolutionary testing has enormous potential to become an invaluable testing and design tool for designers of complex control systems.

[1] When filling the elite pool, we did not check for multiple copies of the same scenario. This simple modification would likely have worked even better and retained a greater proportion of the multiple worst cases.

ACKNOWLEDGMENTS

This work was funded by the Office of Naval Research (contract Number N00014-03M-0356). CA thanks Dr. Jacobus van Zyl of Rand Afrikaans University for his invaluable help with OOTEN.

REFERENCES

[1] A.C. Schultz, J.J.Grefenstette, and K.A. De Jong, "Learning to break things: adaptive testing of intelligent controllers," in *Handbook of Evolutionary Computation G3.5*, pp. 1–10, 1995

[2] A.C. Schultz, J.J Grefenstette, and K.A. De Jong, "Applying Genetic Algorithms to the Testing of Intelligent Controllers," presented at *Applying Machine Learning in Practice, IMLC-95, July 1995*.

[3] J. Wegener, H. Sthamer, and A. Baresel, "Application fields for evolutionary testing," in *EuroSTAR 2001: Ninth European International Conference on Software Testing, Analysis and Review*, 2001.

[4] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," in *Information and Software Technology, Special Issue on Software Engineering using Metaheuristic Innovative Algorithms*, 43(14), pp. 841–854, 2001.

[5] F. Mueller, J. Wegener, "A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium RTAS '98*, pp. 144–154, 1998.

[6] O. Buehler, and J. Wegener, "Evolutionary Functional Testing of an Automated Parking System," in *The 9th International Conference on Information Systems Analysis and Synthesis: ISAS '03, Orlando, Florida, USA, July 31, August 1-2, 2003*.

[7] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

[8] D.C. Wilkins, and J.A. Sniezek, The DC-SCS Supervisory Control Systems for Ship Damage Control: Volume 1- Design Overview. Knowledge Systems Lab Report UIUC-BI-KBS-2000-03. Beckman Institute, University of Illinois, Urbana-Champaign. Dec., 2000.

[9] EPANET homepage: http://www.epa.gov/ORD/NRMRL/wswrd/epanet.html.

[10] OOTEN webpage: http://general.rau.ac.za/civil/wrg/ooten.htm.

[11] H. Takagi, "Active user intervention in an EC search," in *Int'l Conf. on Information Sciences (JCIS2000), Atlantic City, NJ, USA*, pp. 995–998, 2000.

[12] N. Hayashida, and H. Takagi, "Visualized IEC: Interactive Evolutionary Computation with Multidimensional Data Visualization," IEEE Int'l Conf. on Industrial Electronics, Control and Instrumentation (IECON-2000), pp. 2738–2743, 2000.

[13] E. Bonabeau, "Don't trust your gut," *Harvard Business Review*, May, pp. 116-123, 2003.

[14] J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MIT Press, 1992.

[15] R. Dawkins, "The Evolution of Evolvability," in *Artificial Life*, *SFI Studies in the Sciences of Complexity*, C.G. Langton, ed., Reading, Mass: Addison-Wesley, pp. 201–220, 1987.

[16] L. Van Valen, "A new evolutionary law," *Evolutionary Theory* 1, pp. 1–30, 1973.

[17] W.D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," in *Artificial Life II*, C.G. Langton et al., eds., Addison Wesley, pp. 313–324, 1990.

[18] R. Dawkins, *The Blind Watchmaker*, Harlow, England,1986.

[19] W. Banzhaf, "Interactive evolution," in *Handbook of Evolutionary Computation*, T. Bäck, D.B. Fogel, and Z. Michalewicz, Eds., Oxford University Press, C2.9, pp. 1–6, 1997.

[20] H. Takagi, "Interactive evolutionary computation—cooperation of computational intelligence and KANSEI," in *Proceedings of the 5th International Conference on Soft Computing and Information / Intelligent Systems (IIZUKA '98)*, World Scientific, pp. 41–50, 1998.