Automated Test Case Selection for Flight Systems using Genetic Algorithms

Kevin Barltrop[1], Brad Clement[2], Greg Horvath[3], and Cin-Young Lee[4]
Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA, 91009

Without rigorous system verification and validation (SVV), flight systems have no assurances that they will actually accomplish their objectives (e.g., the right system was built) or that the system was built to specification (e.g., the system was built correctly). As system complexity grows, exhaustive SVV becomes time and cost prohibitive as the number of interactions explodes in an exponential or even combinatorial fashion. Consequently, JPL and others have resorted to selecting test cases by hand based on engineering judgment or stochastic methods such as Monte Carlo methods. These two approaches are at opposite ends of the search spectrum, in which one is narrow and focused and the other is broad and shallow. This paper describes a novel approach to test case selection through the use of genetic algorithms (GAs), a type of heuristic search technique based on Darwinian evolution that effectively bridges the search for test cases between broad and narrow spectrums. More specifically, this paper describes the Nemesis framework for automated test case generation, execution, and analysis using GAs. Results are presented for the Dawn Mission flight testbed.

I. Introduction

Finding the fatal flaws or vulnerabilities of complex systems requires thorough testing. In the traditional approach for validating such systems, an expert selects a few key high fidelity test scenarios that he or she believes will most likely uncover problems. Each of the cases is crafted and evaluated by hand. Sometimes, the test engineer adapts his strategy as he goes along, using interesting results from one test case to guide the selection of new cases. The usefulness of these tests in finding flaws can be limited by the biases and assumptions of the expert in the selection process.

Another approach augments the expert selection process with scripting to walk through many scenarios, traversing values of various test parameters. Evaluation can be automated with test result scoring to prioritize review team attention according to features found in the test results. Unfortunately, this approach results in wasting valuable test time on families of similar cases with little new information gained. Furthermore, because the test team must wade through a large volume of results, there is less opportunity to adapt the approach to what is discovered along the way.

In this paper, we describe the application of genetic algorithms to automated test case selection to exploit the advantages of both adaptive expert case selection and automated test space exploration by evolving test scenarios that expose the vulnerabilities of a system under test (SUT) according to models and scoring functions defined by the test team. The test team controls the scope of test space coverage through what they choose to include in the model, and controls the search priorities through the definition of the fitness function to guide the evolutionary search. Furthermore, the starting point for the search is manually specified, allowing the system to cover the ground initially defined as important by the test engineers, continuing the search into other areas as well, and adapting the search to examine more closely those areas with tell-tale signs of stress.

This paper is organized into the following sections: (I) Introduction, (II) Genetic Algorithm Background, (III) Detailed Approach, (IV) Results, and (V) Conclusion.

[1] Senior Systems Engineer, Systems Engineering Section, M/S 321-320.
[2] Senior Member Artificial Intelligence Group, Planning & Execution Systems Section, M/S 301-260.
[3] Staff Software Engineer, Flight Software and Data Systems Section, M/S 301-270.
[4] Senior Software Systems Engineer, Flight Software and Data Systems Section, M/S 301-285K.

II. Genetic Algorithm Background

Genetic algorithms (GAs) are search algorithms that use heuristics based on biological evolution to guide the search. The GA search begins by initializing a set of solutions, referred to as the population, as the starting point for the search. Once a population exists, the search enters an iterative loop in which: (1) each solution in the population, referred to as an individual, is evaluated to determine its fitness, (2) the progenitors of the next generation (iteration) are selected from the current population with a bias towards individuals with higher fitness, (3) the selected progenitors are varied to create new individuals over which to search, and (4) the newly generated individuals are transmitted to the next generation (basically a pass through operation). These steps are repeated until some termination criteria are met. The iterative nature of GAs is illustrated in Figure 1. For a more detailed primer on GAs, refer to (Goldberg, 1989).

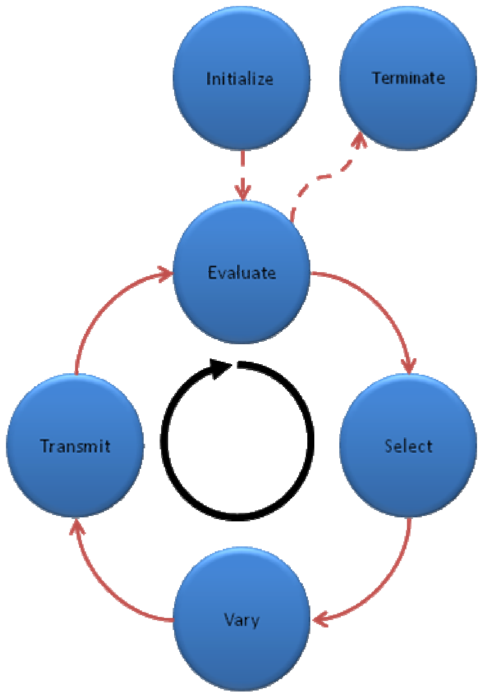


Figure 1. The genetic algorithm in action.

III. Detailed Approach

The following section describes the details of how genetic algorithms can be applied to automated test case selection. These descriptions are intermingled with discussions on canonical GAs and how the specific application domain (automated test case selection) required the development and selection of new GA approaches.

A. Termination

As most testers know, a system can never have too much testing. This is simply a reflection that today's complex systems cannot be exhaustively tested due to resource constraints, which often translates to constrained schedules and limited time. Since this task was mainly as a technology demonstrator, we chose to simply terminate the GA after specified amount of time, in contrast to canonical GAs in which specific convergence criteria are usually specified (e.g. no new solutions have been found for 15 generations).

B. Initialization

Given no *a priori* information about a search space, the initial population is typically chosen uniformly at random over the entire search space. This allows the GA to start from the most diverse possible set of solutions from the outset, and then focus in on good solutions as they are found. This is the recommended approach for the system testing domain as well. However, since there will no doubt be system experts available, GAs can be augmented by seeding the initial population with individuals specified by experts. This can significantly speed up the search in finding solutions in particular areas; although, it may also reduce the likelihood of finding unexpected solutions as the search space may be too focused from the start and unable to effectively reach other areas of the search space. A good balance is to seed the population with some random individuals as well as individuals hand selected by experts. Another approach, which we took, is to incorporate expert knowledge by constraining the search space to specific areas of interest. This approach proved extremely effective in the application of GAs to test selection for pre-existing Dawn test campaigns.

The last thing to consider for initialization is what a reasonable population size is. We chose a population size of 15 individuals, which is typical of evolution strategies (an evolutionary computation method similar to GAs) and perhaps a bit small for GAs. This number was chosen for a variety of reasons. First, experiments with pre-existing Dawn data showed that changes in population size (between 10-30) did not make a big difference in performance. Second, and perhaps more importantly this choice would allow us to balance exploitation and exploration of the search for our specific problem of Dawn fault protection testing. The limiting factor for us was the time required to run tests (aka individuals) on the actual Dawn testbed, which was roughly two hours per test or equivalently 360 tests per month (assuming everything went smoothly). A population of 15 individuals then equates to 24 generations, which is a sufficient number of generations for good solutions to have evolved. Later, we will describe how we improved the number of individuals explored through pre-processing of tests via ASPEN (Automated Scheduling and Planning Environment) to remove non-achievable cases.

C. Selection

Selection is the process by which individuals are selected for survival to the next generation. By biasing selection towards better performing individuals (e.g. higher fitness), the expectation is that traits that increase fitness will be transmitted and exploited in subsequent generations. This bias, also termed selection pressure, can be tuned through different selection approaches.

In its infancy GAs used what is called fitness proportional selection, which requires the use of a scalar fitness function (fitness functions will be described more in detail later under Evaluation). In proportional selection, the probability of selecting a particular individual is equal to the individual's fitness contribution to the aggregate fitness of the entire population. A deficiency of this approach is that the selection pressure depends greatly on the actual fitness values. For example, selection pressure becomes very high if a single individual significantly outperforms (e.g. order of magnitude) all others as it will be over sampled in the next generation. Similarly if the fitness scores of each individual are very closely bunched, then selection becomes little more than random choice and there is no selection pressure.

In order to maintain a constant selection pressure that is independent of the absolute fitness differences, we chose to use tournament selection. This approach avoids the pitfalls of proportional selection by running tournaments between subsets of the population and selecting the best performing individual from each tournament (e.g. only relative performance matters). Choice of tournament size can then be used to tune selection pressure. Notice that when the tournament size is 1 selection is equivalent to random choice and there is no selection pressure. On the opposite end of the spectrum, when the tournament size is equal to the population size, selection pressure is at a maximum because only the best performing individual in the population will ever be selected. A good rule of thumb for tournaments is one third the population size, which we adopted.

D. Variation (and Representation)

There are two primary variation operators used in GAs whose names, mutation and recombination (aka crossover), are borrowed from genetics and evolutionary biology. Mutation serves as a local search operator that slightly perturbs an individual within its neighborhood. Essentially it can be thought of as a hill climbing operator with a specified step size. Recombination is unique to GAs (or other evolutionary computation algorithms of which GAs are a subset) and is an operator that swaps or recombines portion of two individuals with one another. The idea is that traits of good individuals will retain the characteristics that made them good even when swapped between multiple individuals. This operator allows GAs to take, in contrast to the local nature of mutations, large steps in the search space that exploit high performing individuals by swapping traits between them.

So, how exactly are mutation and recombination carried? Before discussing these details, we need to introduce the concept of representation or how individuals (equivalently points in the search space) are represented in the GA. This representation, often referred to as the genotype space, is an internal representation of the actual search space, often referred to as the phenotype space (we will use these two terms interchangeably in the future). Variation operators must be chosen in such a way that small variations in genotype space result in small changes in phenotype space, e.g. they are highly correlated. Without such consistency, relationships in the actual search space cannot be exploited effectively as the search in genotype space will cause dramatic and uncorrelated changes in phenotype space.

While we describe the Dawn fault protection (FP) testing phenotype space here, we believe it to be generally applicable to FP testing and system testing as a whole. The phenotype space can be roughly delineated into inputs and preconditions. For Dawn FP testing, this manifests itself as: what fault to inject and when in the Dawn mission to inject, respectively. Showing foresight, the Dawn test engineers abstracted the preconditions into mission phases (which we later augmented with other preconditions such as battery state of charge) to reduce the size of the search space. In other words, rather than having every possible precondition parameter as a search parameter, sets of precondition parameter values were lumped into the mission phase definitions. This created an interesting structure to the phenotype space in which certain fault injections were not applicable under all phases. For example, during the launch phase, science instrument fault injections aren't applicable since science instruments aren't powered on during launch. One way to visualize this structure is as an unbalanced tree with the choice of branches further down the tree dependent on choices made higher in the tree. For Dawn FP testing, a representative structure is shown in Figure 5. As we go down the tree, it should be apparent that the phase constrains the choice of fault area, which in turn constrains the choice of fault type, which in turn affects the choice of fault argument. For example, phase p1 can only have fault injections in fault area a1, of fault type t1, and so on. Swap out the fault injection specifics with other inputs and we have a generalized phenotype space for system testing.

Now that the phenotype space has been described, the question becomes how to choose an appropriate genotype space and associated variation operators such that genotype and phenotype spaces are strongly correlated (e.g. small changes to an individual in genotype space result in small changes in phenotype space). Before describing our final approach, it is instructional to describe another approach that did not exhibit the strong correlation, which resulted in efficient traversal of the search space in an efficient manner.

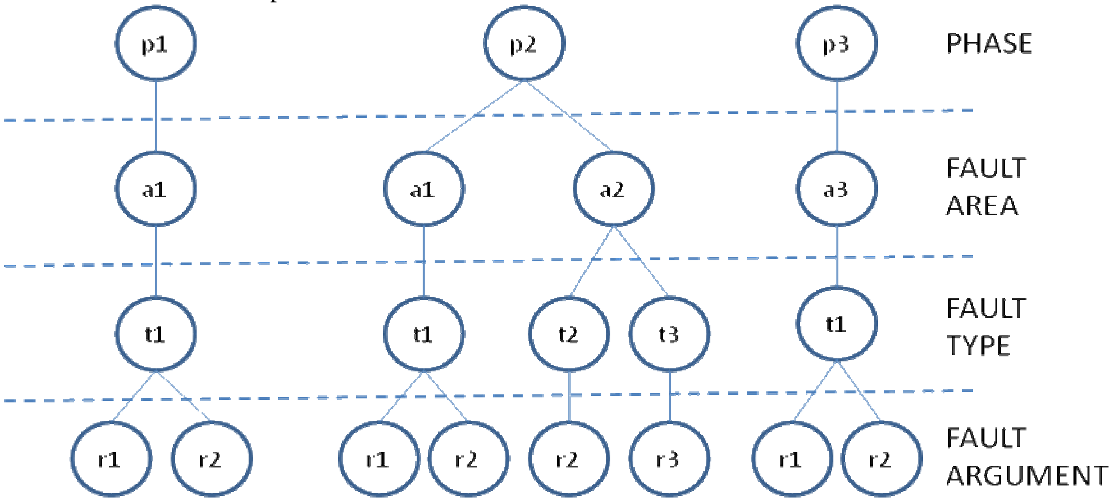


Figure 2. Representative phenotype space for Dawn Fault Protection testing

We decided on an *n*-tuple representation in which each element in the tuple is an integer gene that maps to one of the input or precondition parameters in the phenotype space. Without lack of generality, we describe the approach for a 4-tuple space (which is drawn from the actual Dawn FP testbed). The integer value of each gene indicates the phenotype value whose choices are dependent on any genes upstream in the tuple, in other words an individual needs to be interpreted to phenotype space by walking down the tree. Take for example the tuple. *T = (1, 1, 0, 0)*, where the left most gene is the phase and the right most is the argument. First, let's take a look at *T[0]*, which is the first element in T if we start counting indices from zero. Since there are 3 choices for phase p1, p2, or p3, a value of *T[0] = 1* indicates that the phase is p2, which has two possible area values a1 or a2. The value of *T[1] = 1* then indicates an area of a2, which has two possible types t2 or t3. Continuing down the tree, *T[2] = 0* indicates a fault type of t2, which has one argument r2. Consequently *T[3] = 0* indicates a fault argument of r2. So the genotype (1,1,0,0) corresponds to the phenotype (p2, a2, t2, r2). The full mapping of genotype to phenotype space is shown in Table 1.

For this genotype space, a customized mutation is required. Since the only concept of distance in this space is the Hamming distance, mutation randomly perturbs with equal probability the gene value to one of its other possible values. Depending on which gene is mutated, this may necessitate downstream changes as the previous gene values may no longer be valid. For example, if the individual (1,1,1,0) mutates its phase to p3 from p2, the new individual (2,1,1,0) doesn't make sense. When the individual no longer makes sense, the mutation operator randomly selects all the downstream genes from the possible choices to ensure a valid individual is created leading to a large change in phenotype space. This is done only if there is a need to change. So, if instead the phase of (0,0,0,0) changes from p1 to p2, then the individual (1,0,0,0) does not need to change and can remain as is. In fact, this particular mutation retains all the phenotype structure except of course for the mutated gene value. In these cases, high correlation between genotype and phenotype spaces is maintained. To reduce the occurrence of reinterpretation, which breaks the mold of mutation as an exploitative local search operator, we adjust the mutation rates such that mutation of downstream genes is much more likely than upstream genes.

Table 1. Hierarchical genotype for notional phenotype space.

GENOTYPE	PHENOTYPE			
4-TUPLE	**PHASE**	**AREA**	**TYPE**	**ARGUMENT**
0,0,0,0	P1	A1	T1	R1
0,0,0,1	P1	A1	T1	R2
1,0,0,0	P2	A1	T1	R1
1,0,0,1	P2	A1	T1	R2
1,1,0,0	P2	A2	T2	R2
1,1,1,0	P2	A2	T3	R3
2,0,0,0	P3	A3	T1	R1
2,0,0,1	P3	A3	T1	R2

We adopted a single point scheme for crossover. Since interpretation of the genotype flows in a downstream manner, by adopting single point crossover we reduce the number of reinterpretations that might need to occur by swapping values out of order; hence, the expectation is that single point crossover will be less disruptive than n-point crossover. It should be apparent that this approach to crossover allows repeated structures to be swapped between individuals, allowing whole branches to be maintained across individuals. For example, the subsequence of phenotype a1, t1, r1 can be swapped easily between phases p1 and p2. As with mutation, in the case that there isn't a match of values across the genes, then the recombination requires a reinterpretation of the individual and random selection to make a valid individual. This search operator then allows effective search across like branches while also providing a macro-mutation operation that can significantly alter the phenotype structure.

Recalling that the goal of testing is to find all possible test that expose system flaws rather than a single best test case, we realized that we would need to enact measures to ensure exploration of the search space didn't grind to a halt and converge to a single solution as expected of canonical GAs. One such measure was to co-evolve mutation rate based on feedback on the performance of the search. As the search starts to sputter and not much improvement is made, the mutation rate is ratcheted up to encourage exploration of the broader space. Due to time constraints we were not able to enact this modification.

E. Evaluation

Evaluation is the process of assigning a fitness value, often referred to as the fitness function, to each individual that is indicative of its performance. This value can then be used by selection to bias the search to look in areas of high performance. For system testing, ideally the fitness value would be the number of system flaws that a particular test case exposed. However there isn't an easy way to directly measure the number of flaws exposed so we are dependent on using indicators that are likely associated with flaws. For Dawn FP testing, this resulted in a multi-element vector of indicators that incorporated things like (1) was fault injected, (2) was fault detected, (3) were there resource overruns, etc. Refer to the results for more details on each element. Moreover certain elements in the fitness vector are stronger indicators of a flaw than others and were thus weighted more strongly when considered by the selection algorithm. This was accomplished by using a simple weighted sum over the indicator vector to create a single scalar fitness value. The weights were chosen by hand based on expert perception of indicator strength. It should be noted that in actuality what ends up being tested is the combined Dawn flight system and testbed and as such some indicators were indicators of testbed rather than flight system flaws.

As mentioned previously, the system testing domain seeks to find all solutions, not just a single best solution. Thus, it is in our best interest to preserve diversity (e.g. number of unique individuals within a population) such that exploration continues and the search does not converge. We adopted the use of fitness sharing to aid in this effort. "Fitness sharing modifies the search landscape by reducing the payoff in densely populated regions. It lowers each population element's fitness by an amount nearly equal to the number of similar individuals in the population" (Krahenbuhl, 1998). This prevents the search from converging on particular fitness niches and guides the search away from heavily populated regions leading to more exploration.

GAs are conceptually very simple and in general the majority of computation time is spent in the evaluation step in determining the fitness of each individual. The system testing domain is no exception and for Dawn in particular evaluation requires the execution of a test on the testbed, which requires approximately two hours to complete. In our extended Dawn FP test campaign, it was possible to generate test cases that would never succeed and hence would be a wasted 2 hour effort on the testbed (e.g. the preconditions for the specified inputs would never occur). As part of our extension, we used ASPEN to create the operational scenarios for achieving the preconditions and fault injection points to be input into the testbed driver. ASPEN was a boon to us as it allowed us to avoid expensive testbed runs since it could quickly determine (within tens of seconds) through planning and projection whether or not a given test scenario was achievable.

One aspect of the system testing domain that we did not anticipate was the non-deterministic behavior exhibited by system tests. Stated another way, results of tests using the same preconditions and inputs were not repeatable. We encountered this in both Dawn FP and NGDSN (Next Generation Deep Space Network) testing. Given that one is a mature flight system (already in operations) and the other is immature (an R&TD task), it doesn't seem unreasonable that most systems will encounter a period in their development life-cycle of non-deterministic behavior. The upshot being that noise in a system needs to be addressed. We took a simple approach to handling noise in Dawn FP testing (basically the choices come down to how you resample to remove the effect of noise). Since we weren't preventing previously run individuals from being revisited in later generations, we would re-run individuals suspected of having faulty results (since there were key indicators of the cause of noise).

IV. Results

The Dawn Project is a NASA mission to explore the Ceres and Vesta asteroids. It launched in September of 2007 and will arrive at Vesta in July of 2011. It has a fault protection system that often exhibits complex behavior. As expected of any reasonably complex system, the Dawn test engineers could not exhaustively test all possible combinations of preconditions and inputs to the Dawn FP system. Instead, they restricted the search space to a much smaller and manageable size by abstracting the preconditions into general scenarios or phases and limiting inputs to single fault injections. They also did some preprocessing of the test scenarios to remove the unachievable test cases (e.g. science instrument fault injections during launch are nonsensical). The result of these constraints was that the test space was reduced to a much more reasonable size, which was tested exhaustively in a more or less random order. Given this expert constrained search space, we sought to demonstrate that GAs could find the best test cases (the ones that exposed the most flaws) more quickly than random search, which the actual test campaign implemented. Equivalently, we wanted to show that GAs could find the worst system flaws faster than the standard approach of random selection (sometimes referred to as Monte Carlo approaches) in the context of an expert constrained space.

Figure 3 illustrates the performance characteristics of GAs in relation to random selection in finding the top 4 test cases (remember this is a convention in which the "best" test cases are really the tests that cause the system to exhibit the worst performance). The y-axis expresses the cumulative likelihood of finding a specific test case and the x-axis represents the number of test cases tried to achieve the particular likelihood. In order to obtain statistically significant results for the probabilities, these results were taken from 1000 separate GA runs applied to the pre-existing test data. Random selection has a linearly increasing probability of finding any particular test case. GAs mimic this for approximately one generation (e.g. 15 individuals), which is expected since the initial population is selected at random. However, after the first generation, the GA clearly starts to outperform random selection. These results are very promising and indicate that GAs, when combined with appropriate expert knowledge, can significantly reduce the amount of test cases needed to uncover flaws in the system.

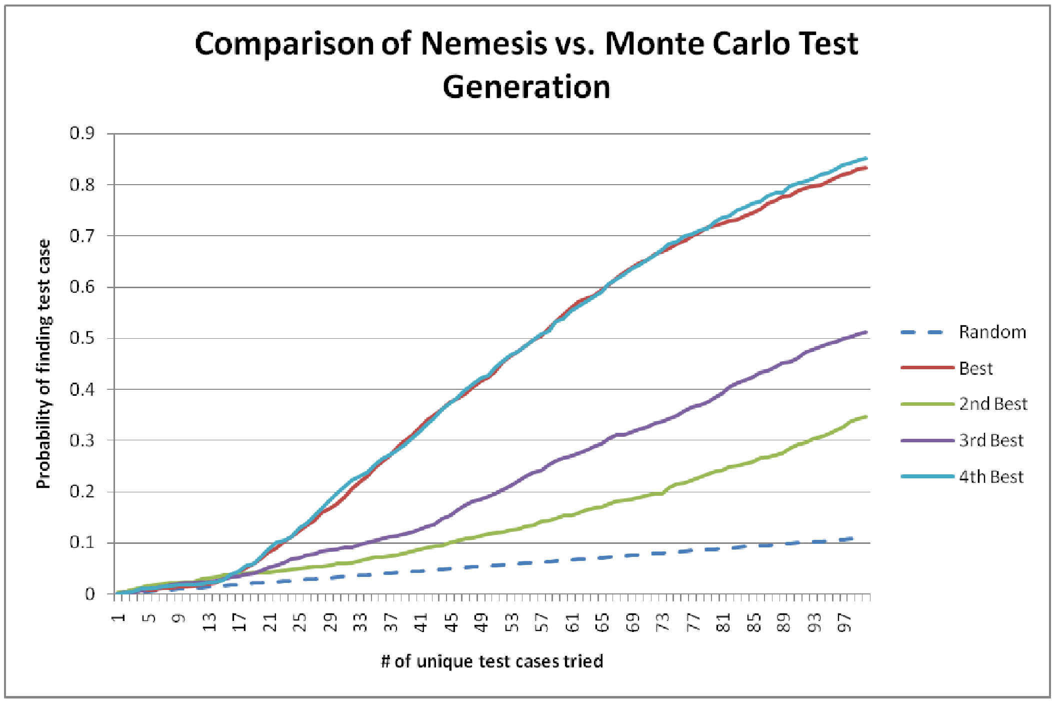


Figure 3. Comparison of GA test case selection to random test case selection (performance in finding the top 4 test cases).

We used the ASPEN software to model the operation of Dawn with a level of detail that took more time to debate than to implement! Figure 4 shows an ASPEN plan for an intentionally unrealistic scenario in which mission activities have been artificially compressed together.

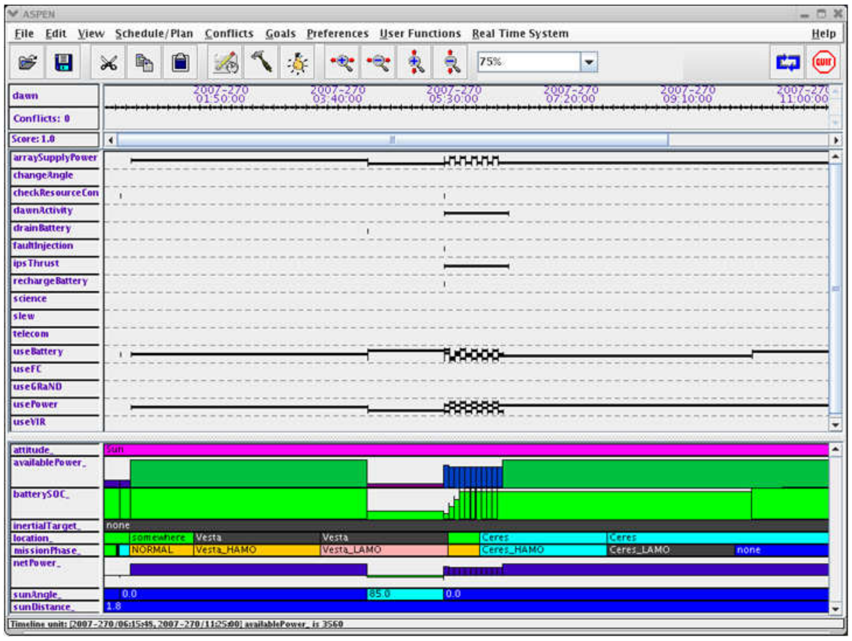


Figure 4. ASPEN plan for an intentionally unrealistic Dawn scenario

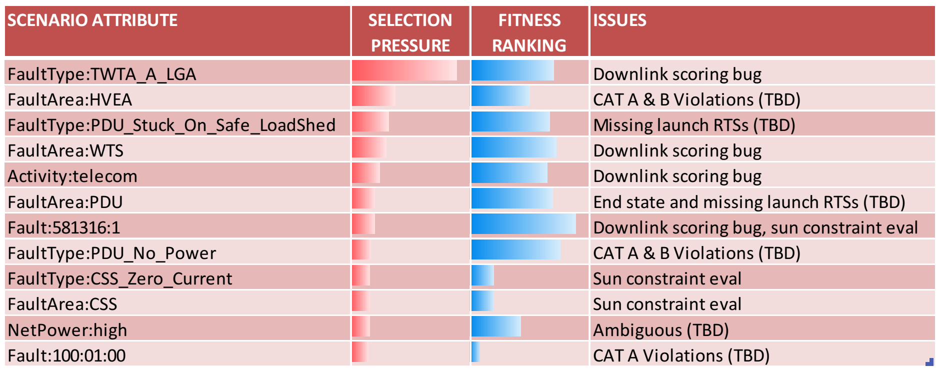
The Dawn proof-of-concept test campaign was limited to a two week period, resulting in the execution of 94 unique scenarios. Unfortunately, the distraction of random test run terminations due to an identified, but not understood, bug in the Dawn test bed thwarted our goal to collect at least 1000 unique cases over a three-month campaign. Our triage of the test results was based upon asking two key questions: (1) What gene variants were associated with high fitness values?; and (2) What genes showed high selection pressure? Gene variants associated with high fitness and selection pressure indicate the scenario features that cause our system under test to behave poorly, and suggest possible flaws in the system under test.

The first question is answered by computing the mean and standard deviation of fitness values for the scenarios that contain a given gene variant. We then take the mean minus two-sigma fitness value as a measure of the lower bound on the fitness for scenarios containing that gene. Implicated gene variants would show a high mean plus two-sigma value, meaning that the fitness is consistently high for all scenarios. This approach helps exonerate variants that happen to occur in an occasional high fitness scenario without actually being the cause of the high fitness.

The second question is answered by computing the expected frequency of occurrence of the gene within the population assuming uniform random selection of valid scenarios, and comparing it to the observed frequency. A high ratio of observed-to-expected frequency indicates selection pressure, implicating the gene variant as an effective inducer of bad behavior in the system under test.

Unfortunately, the small population size dilutes the statistical significance of these assessments. However, within the small number of cases that were run, we had what appeared to be three classes of flaws in the combination of the flight system and test bed. Table 2 shows a relative comparison of the gene variants exhibiting the most selection pressure (where a gene variant occurs at least twice as often as would be expected by chance). The fitness rankings are shown as well to illustrate the degree of correlation between fitness and selection pressure. Four of the top gene variants were connected to a bug in the script to evaluate the downlink signal path for the spacecraft, so the problem was actually in our test infrastructure rather than in the flight system. The fact that three of those genes directly controlled telecom-related features of the scenario provides a strong example of detecting dependencies between genes. A second flaw was due to an overly strict check on sun-pointing violations, again in one of our scripts to evaluate the test runs rather than in the flight system itself. Similar to the first flaw, three implicated gene variants were all directly connected to the same area, in this case, controlling the sun sensor features of the scenario. The last flaw category covers operational and configuration violations. The "CAT A" and "CAT B" violations are indications that the system configured hardware in a way that formal rules of operation define as dangerous or risky. "Missing launch RTS" indicates that the system failed to complete required configuration steps. Investigation of the details for those cases is on-going.

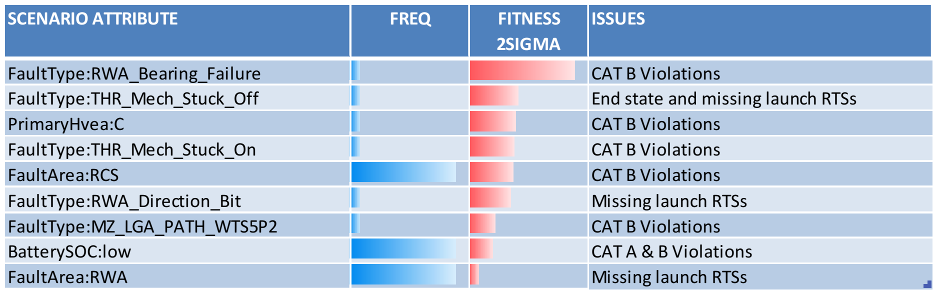
Table 2. Selection Pressure Assessment



SCENARIO ATTRIBUTE	SELECTION PRESSURE	FITNESS RANKING	ISSUES
FaultType:TWTA_A_LGA			Downlink scoring bug
FaultArea:HVEA			CAT A & B Violations (TBD)
FaultType:PDU_Stuck_On_Safe_LoadShed			Missing launch RTSs (TBD)
FaultArea:WTS			Downlink scoring bug
Activity:telecom			Downlink scoring bug
FaultArea:PDU			End state and missing launch RTSs (TBD)
Fault:581316:1			Downlink scoring bug, sun constraint eval
FaultType:PDU_No_Power			CAT A & B Violations (TBD)
FaultType:CSS_Zero_Current			Sun constraint eval
FaultArea:CSS			Sun constraint eval
NetPower:high			Ambiguous (TBD)
Fault:100:01:00			CAT A Violations (TBD)

Table 3 shows a relative comparison of the gene variants associated with the highest fitness values (depicted in the column labeled "FITNESS 2SIGMA"). The frequency is also presented to show where luck, more than selection pressure, may be behind the high fitness. For example, the top scoring gene variant for a reaction wheel assembly (RWA) bearing failure occurred only once during the campaign, so we cannot claim that the fitness played a strong role in uncovering the issue. However, we can say that randomness aspect of the GA approach allowed it to be uncovered since it was not part of the initial generation of cases.

Table 3. Fitness Assessment



SCENARIO ATTRIBUTE	FREQ	FITNESS 2SIGMA	ISSUES
FaultType:RWA_Bearing_Failure			CAT B Violations
FaultType:THR_Mech_Stuck_Off			End state and missing launch RTSs
PrimaryHvea:C			CAT B Violations
FaultType:THR_Mech_Stuck_On			CAT B Violations
FaultArea:RCS			CAT B Violations
FaultType:RWA_Direction_Bit			Missing launch RTSs
FaultType:MZ_LGA_PATH_WTS5P2			CAT B Violations
BatterySOC:low			CAT A & B Violations
FaultArea:RWA			Missing launch RTSs

The Dawn project will soon resume testing using this framework, with the possibility of expanding the scope of the model to allow exploration of even more varied cases.

V. Conclusion

Using genetic algorithms we have successfully demonstrated via four different adaptations the capability to autonomously explore scenarios and to provide a path for finding the worst problems in a system under test. Our errors in the evaluation scripts solidly demonstrated the approach's effectiveness in pointing to flaws, though in this case in the test infrastructure rather than in the system under test. We expect that the Dawn project's resumption of testing with corrected scripts offers the most efficient and cost-effective way for uncovering the worst flaws that remain in the functions needed for in-orbit operations.

Acknowledgments

We thank the Dawn Project for the interesting case study material, Giangi Sacco for his work investigating the sensitivity of the performance to GA parameter variations, and Richard Terrile for his advise on GA application. The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Periodicals

[1]Krahenbuhl, B.S., "Fitness Sharing and Niching Methods Revisited," *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 3, 1998, pp. 97-106.

Books

[4]Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Kluwer Academic Publishers, Boston, MA, 1989.

Proceedings

[7]Thompson, C. M., "Spacecraft Thermal Control, Design, and Operation," *AIAA Guidance, Navigation, and Control Conference*, CP849, Vol. 1, AIAA, Washington, DC, 1989, pp. 103-115