Software V&V Support by Parametric Analysis of Large Software Simulation Systems

Johann Schumann[1], Karen Gundy-Burlet[2], Corina Păsăreanu[3], Tim Menzies[4], and Anthony Barrett[5]
[1]RIACS, NASA Ames, Johann.M.Schumann@nasa.gov [2]NASA Ames, Karen.Gundy-Burlet@nasa.gov
[3]CMU, NASA Ames, corina.s.pasareanu@nasa.gov [4]Lane CS & EE, West Virginia University, tim@menzies.us
[5]JPL, California Institute of Technology, barrett@jpl.nasa.gov

Abstract—Modern aerospace software systems simulations usually contain many (dependent and independent) parameters. Due to the large parameter space, and the complex, highly coupled nonlinear nature of the different system components, analysis is complicated and time consuming. Thus, such systems are generally validated only in regions local to anticipated operating points rather than through characterization of the entire feasible operational envelope of the system. We have addressed the factors deterring such a comprehensive analysis with a tool to support parametric analysis and envelope assessment: a combination of advanced Monte Carlo generation with n-factor combinatorial parameter variations and model-based testcase generation is used to limit the number of cases without sacrificing important interactions in the parameter space. For the automatic analysis of the generated data we use unsupervised Bayesian clustering techniques (AutoBayes) and supervised learning of critical parameter ranges using the treatment learner TAR3. This unique combination of advanced machine learning technology enables a fast and powerful multivariate analysis that supports finding of root causes.

1. INTRODUCTION

The design and development of complex systems like Aerospace applications heavily depend on simulation. Only by exploring detailed simulation models, the exact behavior of the system can be analyzed in nominal and off-nominal scenarios. These systems simulations often contain a large number of dependent and independent parameters. The analysis of such simulation systems is very complicated and time-consuming due to a large parameter space and the complex, highly coupled nonlinear nature of the different system components. Still these parameter settings need to be scrutinized during verification and validation (V&V), as those parameters contribute to the performance of the aerospace system and can affect its safety. Typically, the performance of an aerospace vehicle simulation depends on a large number of vehicle hardware parameters including center of gravity, mass, moment of inertia, propulsion system, environment as well as control system parameters such as gains and deadbands. A major task is to establish safe ranges for control system parameters that ensure a safe, on-target landing given the expected variation in vehicle performance. Exhaustive exploration of all parameter combinations is infeasible for such a complex system, so, traditionally, parameters are randomly sampled from a defined distribution for a statistically significant number of runs (Monte Carlo testing). Vast amounts of data can be generated that way, and manual inspection of this data is usually confined to gross features of the solution (such as absolute compliance with requirements). Valuable trend and parameter sensitivities are often ignored and anomalous or unexpected data can easily be overlooked. Thus, such systems are generally validated only in regions local to anticipated operating points rather than through characterization of the entire feasible operational envelope of the system.

Researchers have recognized that "designers must be able to examine various design alternatives quickly and easily among myriad and diverse *configuration possibilities*" [2]. The number of configuration possibilities can be dauntingly large. A model with only 20 binary choices already has $2^{20} > 1,000,000$ possible configurations, far beyond the capability of human comprehension.

Accordingly, since 2000 [1], we have explored sampling those configurations at random, running the resulting model, scoring the output with some oracle, and then using data mining techniques to find the configuration options that most improve model output. In this paper, we describe a new combination of methods and tools to study the configuration parameters and safety envelopes for large software simulations:

- An n-factor combinatorial test vector generation algorithm to target tests toward regions of the parameter space where interactions among parameters are key to performance.
- Model-based generation of testcases allows the analyst to specifically drive simulations toward the execution of interesting states in the mode logic.
- The TAR3 minimal contrast set learner [4] is a supervised learning method that returns the minimal deltas between desired outcomes (hitting the target) and all the other outcomes.
- EM clustering algorithms that are autogenerated by the AUTOBAYES program synthesis tool. Clustering is an unsupervised learning method that obtains the most probable estimates for class frequency and governing parameters.

It was found that the combination of methods yielded more information than any method used in isolation. The operation of each algorithm can be intelligently informed and usefully constrained by using the output of the other. Combi-

1978-1-4244-2622-5/09/$25.00 ©2009 IEEE.
IEEEAC Paper #1316, Version 4 Updated 01/17/2009.

natorial test exposes interactions between parameters, TAR3 focuses the analysis on a small number of variables while AUTOBAYES reveals structures missed by TAR3.

The rest of this paper is structured as follows: In Section 2, we will introduce the MCB project and the HTV (Hover Test Vehicle), which is used in this paper to illustrate our approach. Section 3 gives a high-level description of our tool architecture, which will be discussed in Section 4 (generation of the simulation scenarios) and Section 5 (data analysis). Section 6 presents results of the HTV analysis to determine best parameter ranges for such tasks as safely hovering and landing the hover test vehicle. Section 7 concludes and discusses future work.

2. MODULAR COMMON BUS

The Modular Common Bus (MCB) project is being developed as a family of small, low-cost spacecraft composed of modular components. The intent is that science instruments are solicited that meet the spacecraft's mass and power budget, rather than the current practice of designing a custom spacecraft for each new science instrument. The modular design includes components that enable a range of missions, including orbital and lander type missions. Simulink/Matlab® is used to rapidly prototype the design of the flight software and autogenerate the on-board software for the vehicle. This process also enhances the validation and verification (V&V) of the vehicle requirements and flight software. An initial prototype of the MCB lander configuration named the Hover Test Vehicle (HTV) has been developed and flown in order to assess this spacecraft development methodology.

The HTV is a prototype of a lunar landing configuration. The vehicle is comprised of a main payload module and propulsion module. For earth-based testing, the propulsion module contains a pair of scuba tanks that power one main and 6 attitude control system thrusters. The payload module contains an inertial measurement unit (IMU), avionics and communications equipment. An image of the HTV is shown in Figure 1 (see [3] for a complete description).

The data mining techniques discussed in this paper have been applied to the HTV simulation. It contains models of the vehicle including 6DOF dynamics, propulsion system, effectors, sensors, and avionics. Parametric analysis of the system was performed in order to assess flight readiness and system margins in preparation to the flight tests.

Numerous parameter variations are considered in order to determine the resilience of the algorithms to dispersions in mass properties, orientation, and tank pressures. Other analyses were performed in order to assess optimal controller settings given the expected dispersions in vehicle configuration. Requirements on the behavior of the model are encoded in order to accumulate failure data, and all solutions are rated relative to their adherence to requirements, such as soft landing and minimum excursion from the target point. The data mining techniques discussed here are intended to characterize the operational envelope of the simulation and to find the ranges of parameters that lead to the lengthiest flights that lands close to the target. The margin between the best ranges to the failure points of each parameter can then be determined.

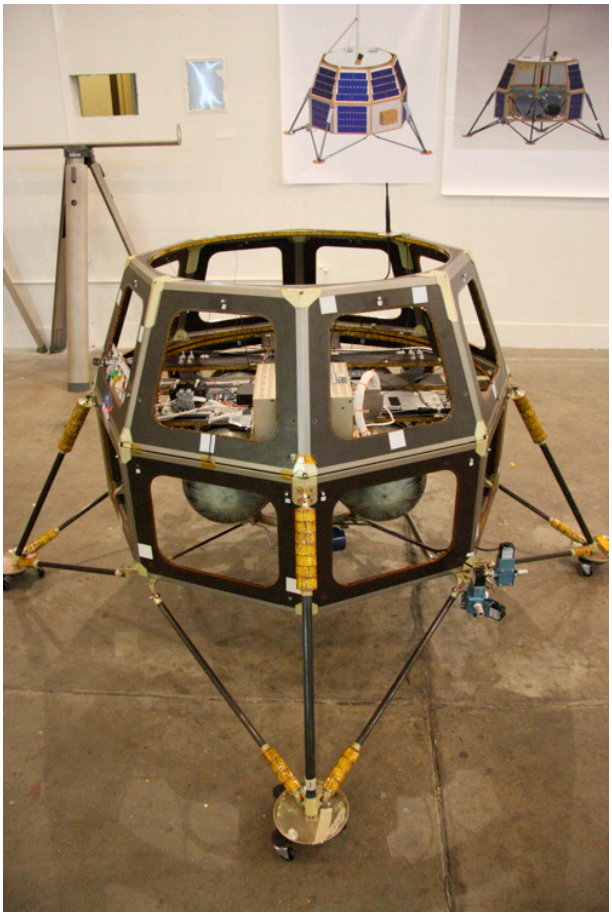


Figure 1. Hover Test Vehicle

3. TOOL ARCHITECTURE

Our tool (Figure 2) is centered around the simulation environment; currently the tool interfaces with the NASA Trick simulation environment and with Mathwork's Simulink/Stateflow. The user sets up the specifications of the desired parameter variations and their statistical properties, and provides high level models (e.g., UML statecharts). Once the test-suite is generated, simulation runs are being executed and the resulting data saved. Data analysis and visualization is controlled via a Matlab graphical user interface.

In the following sections, we will describe the main components of this tool: mechanisms to generate the simulation cases and tool support for the analysis of the simulation data. We will then present some results on experiments with the HTV simulation.

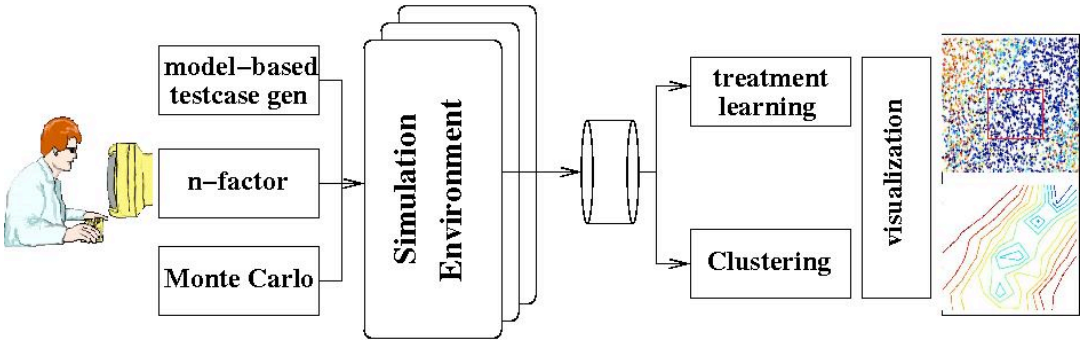


Figure 2. Tool architecture

4. Generation of Simulation Cases

Due to the inherent complexity a combinatorially exhaustive parameter exploration is not feasible for a realistically large system. We therefore use a combination of extended Monte Carlo random generation with n-factor combinatorial exploration and model-based testcase generation.

Covering the Option Space

We have explored two approaches toward covering the option space, a standard Monte Carlo and a 3-factor combinatorial technique. The standard Monte Carlo approach is simplest. It generates parameters for a simulation run by randomly selecting from user defined probability distributions, such as Gaussian or Uniform. The main drawback of this approach is a lack of any coverage guarantee, resulting in the need to run a large number of simulations to attain a given level of user confidence. Unlike the standard Monte Carlo approach, the combinatorial approach makes a coverage guarantee while attempting to perform a minimal number of simulations. In the case of an n-factor combinatorial, the guarantee is that any setting of any n discrete parameters appears in at least one of the simulations. For instance, a 2-factor combinatorial test suite for 20 binary parameters has only 11 tests, which is much less than the million tests needed to exhaustively cover every combination. A 3-factor case over the same parameters only increases the number of tests to 26.

While the number of tests performed using the combinatorial approach is minuscule compared to exhaustive testing, anecdotal evidence suggests that this small number of tests is enough to catch most coding errors [10], [5], [6]. The underlying premise behind the combinatorial approach can be captured in the following statements:

- The simplest failures in a program are triggered by a single input parameter.
- The next simplest failures are triggered by an interaction of two input parameters.
- Progressively more obscure failures involve interactions between more parameters.
- Exhaustive testing involves trying all combinations of all inputs.

So errors can be grouped into families depending on how many parameters need specific settings to exercise the error. The n-factor combinatorial approach guarantees that all errors involving the specific setting of n or fewer parameters will be exercised by at least one test. When real-valued parameters have to be varied, each parameter is represented as a partition of discrete ranges. When turning a computed test into a simulation run, each range is replaced by a real value chosen from a uniform distribution across that range. The result is a multidimensional space of simulation runs that projects down to a uniform distribution on any plane of input parameters.

There is a number of algorithms in the literature (e.g., [7]) to generate 2-factor combinatorial test suites. Our algorithm is a generalization of IPO [8] and has additional features that a real world test suite generator would need [9]. These features include the ability to explicitly include particular combinations, explicitly exclude particular combinations, require n-factor combinatorial coverage of specific subsets of parameters, and tie the existence of particular parameters to the setting of another parameter.

The resulting algorithm is 1041 lines of documented Java code. Even with these extra capabilities the algorithm generates test suites that are comparable to those generated by the more restricted systems in the literature, and are generated very rapidly as shown in Table 1. Times are measured on a 400MHz Windows laptop machine.

Model-based Scenario Generation

The generation of simulation scenarios discussed so far focuses on the exploration of the parameter space. In many cases, however, a systematic exploration with respect to code coverage or modes is important. Complex aerospace software usually contains state machines, which govern the behavior of the system. In order to be able to explore specific modes systematically, we combine our generation approach with the automatic, model-based generation of simulation cases.

A model of the relevant system formalized as a Simulink/Stateflow diagram or a UML statechart is used as the specification. After a translation of the diagram into an in-

Problem Sizes	#Tests	Time
$3^4 \approx 10^2$	9	$\ll 1$ sec
$3^{13} \approx 10^6$	19	$\ll 1$ sec
$4^1 \times 3^{39} \times 2^{35} \approx 10^{29}$	29	< 1 sec
10^{20}	216	1 sec
$3^{1000} \approx 10^{477}$	48	22 sec

Table 1. 2-factor generation. X^Y means that there are Y X-valued parameters.

ternal (procedural) form, SPF (Symbolic Pathfinder, [11]) is used to automatically generate testcases that, when executed, exercise a specific part or state of the state diagram. The synergistic combination of model checking (Java Pathfinder) and symbolic execution of the SPF tool produces ranges of inputs and parameters required to reach the specified state. This information can be used to constrain the Monte Carlo and n-factor combinatorial generation of simulation cases to focus on the exploration of important modes (e.g., hover, landing).

5. DATA ANALYSIS TOOLS & METHODS

In this section, we will discuss two tools to analyze the multivariate data set produced by the simulation runs: TAR3, a minimal contrast set learner to study the influence of input variables and model parameters on the outcome of the simulation, and the AUTOBAYES tool to automatically generate efficient algorithms for unsupervised clustering of multivariate data.

TAR3 Treatment Learning

Multi-Dimensional Optimization—BORE, short for *best* or *rest*, takes instances scored on multiple utilities as input and classifies each of them "best" or "rest". BORE maps the instance outputs into a hypercube, which has one dimension for each utility.

BORE normalizes instances scored on N dimensions into "zero" for "worst", and "one" for "best". The corner of the hypercube at $1, 1, \ldots$ is the *apex* of the cube and represents the desired goal for the system. All examples are scored by their normalized Euclidean distance N_i to the apex. Usually, a domain-specific distance function (e.g., deviation from desired landing spot) is used.

For each run i of the simulator, the n outputs X_i are normalized to the range $0 \ldots 1$ as $N_i = (X_i - \min(X))/(\max(X) - \min(X))$. The Euclidean distance of $\{N_1, N_2, ...\}$ to the ideal position of $\{N_1 = 1, N_2 = 2, ...\}$ is then computed and normalized to the range $0..1$ as

$$W_i = 1 - \sqrt{(N_1^2 + N_2^2 + \ldots)/n}$$

where *higher* W_i ($0 \leq W_i \leq 1$) correspond to *better* runs. This means that the W_i can only be improved by increasing *all* of the utilities. To determine the "best" and "rest" values, all the W_i scores are sorted according to a given threshold BEST. The top BEST% are then classified as "best" and the remainder as "rest".

Treatment Learning with TAR3—Once BORE has classified the data into "best" and "rest", a data miner is used to find input settings that select for the better outputs. This study uses the TAR3 data miner at it returns the *smallest* theories that *most* effect the output. This means that TAR3 tries to determine a minimal set of model parameters, which have the most influence on the behavior of the simulation system.

The inputs to TAR3 is a set of training examples E. Each example maps a set of attribute ranges to some class symbol, i.e., $\{R_i, R_j, \ldots \rightarrow C\}$. The class symbols $C_1, C_2, \ldots$ are stamped with some utility score that ranks the classes $\{U_1 < U_2 < \ldots < U_C\}$. Within E, these classes occur at frequencies $F_1, F_2, \ldots, F_C$ where $\sum F_i = 1$. A "treatment" T of size M is a conjunction of attribute ranges $\{R_1 \wedge R_2 \ldots \wedge R_M\}$. For a subset $e \subseteq E$ that is consistent with the treatment the classes occur at frequencies $f_1, f_2, \ldots, f_C$. TAR3 seeks the smallest treatment T which induces the biggest changes in the weighted sum of the utilities times frequencies of the classes. Formally, this is called the *lift* of a treatment:

$$\textit{lift} = \frac{\sum_C U_C f_C}{\sum_C U_C F_C}$$

Classes in treatment learning get a score U_C and the learner uses this to assess the class frequencies resulting from *applying a treatment* (i.e., applying constraints to the inputs). In normal operation, a treatment learner does *controller learning* that finds a treatment, which selects for better classes and reject worse classes. By reversing the scoring function, treatment learning can also select for the worse classes and reject the better classes. This mode is called *monitor learning* since it finds the thing we should most watch for.

Formally, treatment learning is a weighted-class minimal contrast-set association rule learner. The treatments are associations that occur with preferred classes. These treatments serve to contrast undesirable situations with desirable situation where more of the outcomes are favorable. Treatment learning is different to other contrast set learners like STUCCO [12] since those other learners don't focus on minimal theories. Conceptually, a treatment learner explores all possible subsets of the attribute ranges looking for good treatments. Such a search is infeasible in practice so the art of treatment learning is quickly pruning unpromising attribute ranges. This study uses a version of the TAR3 treatment learner [20] with stochastic search.

AutoBayes

A well-known method to find structure in large sets of data is to perform clustering. Clustering is an unsupervised learning method that tries to estimate class membership matrix and

```
model mog as 'Multivar. Mix of Gaussians';

const int D as 'number of variables'
const int N as 'number of data points'
const int C  as 'number of classes'
      with 1 < C; with C << N;

double phi(1..C) as 'class probabilities'
      with 1 = sum(_i := 1..C, phi(_i));
double mu(1..D, 1..C), sigma(1..D, 1..C);

output int c(1..N) as 'latent variable';
c(_) ~ discrete(phi);

data double x(1..D, 1..N);
x(_i,_j) ~ gauss(mu(_i,c(_j)),sigma(_i,c(_j)));

max pr(x|{phi, mu, sigma}) wrt {phi, mu, sigma};
```

Figure 3. AutoBayes specification for mixture model. Keywords are underlined.

class parameters, only given the data. A variety of algorithms can be used, for example, the EM-algorithm [16]. A number of EM-implementations is available (e.g., Autoclass [17], EMMIX [18], or MCLUST [19]) and could be used for this problem. However, in order to refine the statistical model (e.g., by incorporating other probability distributions for certain variables or to introduce domain knowledge), the EM-algorithm needs to be modified substantially for each problem variant, making experimentation a time-consuming and error-prone undertaking. Thus, we are using AUTOBAYES tool to produce customized variants of the EM-algorithm.

AUTOBAYES [15], [14] is a fully automatic program synthesis system that generates efficient and documented C/C++ code from abstract statistical model specifications. From the outside, AUTOBAYES looks similar to a compiler for a very high-level programming language: it takes an abstract problem specification in the form of a (Bayesian) statistical model and translates it into executable C/C++ code that can be called from Matlab.

On the inside, however, AUTOBAYES works quite different: AUTOBAYES first derives a customized algorithm skeleton implementing the model and then transforms it into optimized C/C++ code. Hereby, the input specification is translated into a Bayesian Network [22]. The program synthesis system uses a schema based approach to break down large problems into statistically independent subproblems and tries to solve them symbolically. If no solution can be found, a customized numerical algorithm is instantiated. This task is heavily supported by a domain-specific schema library, an elaborate symbolic subsystem, and an efficient rewriting engine. After optimization C or C++ code is generated for various platforms (e.g., embedded systems, Matlab, Simulink, or Octave). For our experiment, we used AutoBayes to generate code that can be called from Matlab as a MEX function.

The basic statistical model used for this study describes the properties of the data in a fully declarative fashion: for each


```
data double x(1..D, 1..N);
data double flg(1..D, 1..N);
data double arr(1..D, 1..N);
x(_i,_j) ~ gauss(mu(_i,c(_j)),sigma(_i,c(_j)));
flg(_i,_j) ~ discrete(rho_flg(_i,c(_j)));
arr(_i,_j) ~ poisson(rate(_i,c(_j)));

max pr({x, flg, arr}|{phi, mu, sigma, p_flg, rate})
  wrt {phi, mu, sigma, p_flg, rate};
```

Figure 4. Specification for mixture model with different probability density functions.

problem variable of interest (i.e., observation or parameter), properties and dependencies are specified via probability distributions and constraints. Figure 3 shows how Gaussian mixture model with diagonal covariance matrices can be represented. The model assumes that the data consists of `N` points in `D` dimensions such that each point belongs to one of `C` classes; the first few lines of the specification just declare these symbolic constants and specify the constraints on them. Each point `x(1..C,_j)` (where `..` corresponds to Matlab's subrange operator `:` and `_i`, `_j` are index variables) is drawn independently from a univariate Gaussian with mean `mu(_i,c(_j))` and standard deviation `sigma(_i,c(_j))`. The unknown distribution parameters can be different for each class and each dimension; hence, we declare them as matrices. The unknown assignment of the points to the distributions (i.e., classes) is represented by the latent variable `c`; since we are interested in the classification results as well (and not only the distribution parameters), `c` is declared as `output`. `c` is distributed as a discrete distribution with the relative class frequencies given by the also unknown vector `phi`. Since each point must belong to a class, the sum of the probabilities must be equal to one. Finally, we specify the goal inference task, maximizing the conditional probability `pr(x|{phi, mu, sigma})` with respect to the parameters of interest, `phi`, `mu`, and `sigma`. This means that we are interested in a maximum likelihood estimate (MLE) of the model parameters.

However, not all simulation variables are Gaussian distributed, e.g., boolean values or angles ($0 \ldots 2\pi$). Although in most engineering applications, these values are converted to Gaussians (e.g., by adding noise), the AUTOBAYES system automatically handles mixtures with different probability density functions. All that needs to be done to the specification in Figure 3 is to add declarations for the distribution parameters, the additional data variables, and the distribution and goal statement—Figure 4 shows a specification snippet. For details see [14].

6. RESULTS

For the experiment shown below, we used standard Monte Carlo and three-factor combinatorial techniques. For the Monte Carlo cases, one thousand cases were run, with random values chosen for each of the input parameters from their respective probability distributions. Here, the parame-

ter ranges are determined such that they are likely to include failure cases.

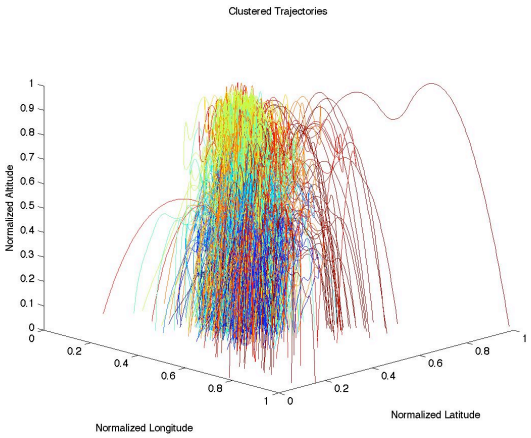


Figure 5. HTV trajectories for 1000 simulation runs with different simulation parameters.

The output data encompass a wide range of variables that are saved at each iteration point. Thus, they are actually time series data over a large number of variables. Figure 5 shows a 3D representation of 1000 simulated trajectories generated through a traditional Monte Carlo style test generation technique.[1] Data representing key parameters such as maximum altitude and excursion, time of flight and landing velocities were extracted from the simulation for the analysis that follows. After preprocessing, we obtained a data set with 10 dimensions.

These normalized data then were clustered using the Matlab/C code as was generated by AUTOBAYES (790 lines of C code). The generated data analysis algorithm determined that with 10 classes, a good separation can be achieved.

The results of clustering, relative to time of flight and initial wet mass is shown in Figure 6. Different colors indicate into which class a specific simulation run falls. The classes are ranked using a penalty function based on landing velocity and position error. Blue indicates the lowest penalty function, while red indicates the highest penalty function. The trend seen in the figure is that lower times of flight generally have lower penalty functions. Wet mass is key to time of flight, but other factors contribute to the failure clusters exhibited in the plot. Here, the yellow through red classes show repeated violations of the landing requirements. Overall statistics for the compliance with performance requirements was (over the 1000 runs):

- Maximum Position Excursion no greater than 3m: 258 cases failed
- Vertical Velocity on Landing no greater than 4.0m/s: 167 failed

[1] In this paper, colors encode the class membership as determined by the AUTOBAYES clustering algorithm.

- Horizontal velocity on Landing no greater than 1.0m/s: 49 failed

Some of the cases failed more than one of the requirements, so the total number of cases without failure was 656. This simulation exhibited a sufficient range of acceptable/failure cases such that the key parameter values defining the failure front could be determined.

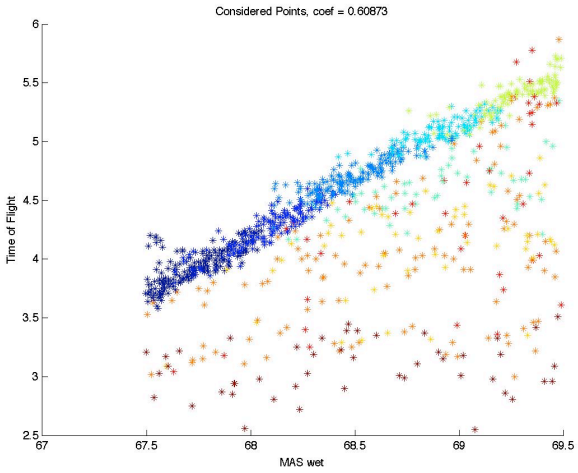


Figure 6. Relationship between Time of Flight and Initial Wet Mass Colors indicate class membership.

To determine the root cause of the poor landing clusters, TAR3 was utilized to determine the key parameters driving the simulation behavior. It identified several key parameters, that in conjunction with each other drove the failure behavior. In particular, TAR3 identified the following combined set of parameter ranges that induced failure:

- MAS cgy location wet=0.005951...0.007885
- MAS Iyz wet=-0.961177...-0.835669
- MAS wet=68.826103...69.489502
- INI rotx=0.928976...2.996930

This indicates that the initial mass, y center of gravity in combination with one the Iyz moment of inertia and the initial x rotation of the vehicle would cause unacceptable behavior. This finding introduced a set of day-of-flight flight rule restrictions on initial orientation and a more careful evaluation of the center of gravity of the vehicle. It was determined that reduced mass in the tanks would lead to a lower y-cg offset, so a reduced tank pressure was also utilized for the first free-hover test flight.

Figure 7 shows contours of "likelihood" for two parameters identified as critical by TAR3. To generate this plot, the input domain was discretized and the likelihood of success was computed for each cell. Here, likelihood combines the overall probability of success in the whole population, ratio of success in the local cell, and local cell population. Likelihood will approach one in well-populated cells with a high ratio

of success, and will approach zero if either there is poor statistical support or a low ratio of success. If a variable is not correlated, then the local likelihood will approach the overall ratio of success in the complete population. Using this metric, it is seen that choosing a lower mass and carefully orienting the vehicle improves the resilience of the simulation to dispersions in other key parameters.

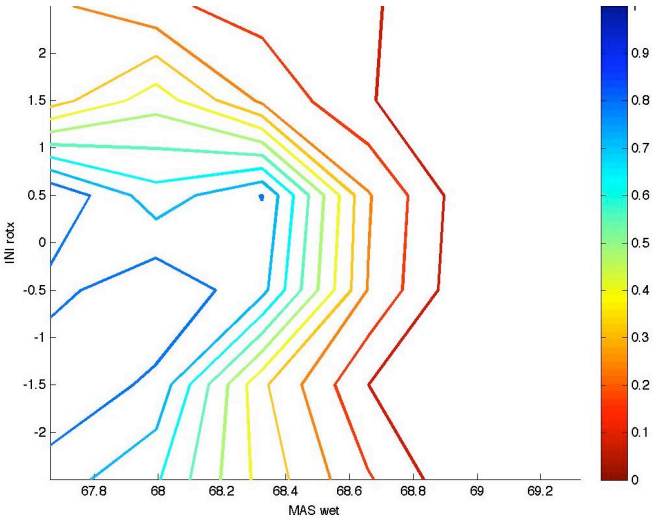


Figure 7. Likelihood of successful flight relative to initial x rotation and mass

7. Conclusions

In this paper, we have explored a combination of two learners (AutoBayes and TAR3) to explore the internal state space of some flight guidance software with combinatorial test techniques. The combination of these technologies revealed features that would have been invisible for state of the practice. Further, the experiment suggests some novel ways that these technologies could usefully augment each other. In the case of the first test flight of the hover test vehicle, the combined technologies provided guidance as to strategies that would increase the chances of a successful test flight. In fact, the hover test vehicle performed in the manner suggested by the simulation, and a successful set of free-flying test flights were conducted.

More generally, given the growing importance of model-based reasoning in software engineering, the ability to use data miners to find and constrain the most important parts of our software models, should prove to be a technique of growing importance in the years to come.

Acknowledgments

This research was sponsored in part by NASA under the ISRDS Contract NNA07BB97C (RIACS/USRA). Part of this work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA.

References

[1] Menzies, T. and Sinsel, E. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from http://menzies.us/pdf/00ase.pdf.

[2] Gray, J., Lin, Y., and Zhang, J. Automating change evolution in model-driven engineering. *IEEE Computer*, 39(2):51–58, February 2006.

[3] Bell, J., E. A. Hover testing of a prototype small planetary spacecraft. In *Submitted to IAC'08, the 59th International Astronautical Congress, Glasgow*, 2008.

[4] Menzies, T. and Hu, Y. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from http://menzies.us/pdf/03tar2.pdf.

[5] Dunietz, I. S., Ehrlich, W. K., Szablak, B. D., Mallows, C. L., and Iannino, A. Applying design of experiments to software testing: experience report. In *Proc. ICSE '97*, pages 205–215, 1997.

[6] Wallace, D. R. and Kuhn, D. R. Failure modes in medical device software: an analysis of 15 years of recall data. *International Journal of Reliability, Quality and Safety Engineering*, 8(4), 2001.

[7] Mats Grindal, Jeff Offutt, S. F. A. Combination testing strategies: a survey. *Software Testing, Verification and Reliability*, 15(3):167–199, 2005.

[8] Tai, K. and Lie, Y. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, 2002.

[9] Czerwonka, J. Pairwise testing in real world, practical extensions to test case generators. In *Proceedings of 24th Pacific Northwest Software Quality Conference*, 2006.

[10] D. Cohen, S. Dalal, J. Parelius, and G. Patton, "The combinatorial design approach to automatic test generation," *Software, IEEE*, 13(5):83–88, Sep 1996.

[11] C. S. Păsăreanu, J. Schumann, P. Mehlitz, M. Lowry, G. Karsai, H. Nine, S. Neema. Model Based Analysis and Test Generation for Flight Software In preparation, 2009.

[12] Bay, S. and Pazzani, M. Detecting change in categorical data: Mining contrast sets. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999.

[13] C. S. Păsăreanu, P. C. Mehlitz, D. H. Bushnell, K. Gundy-Burlet, M. Lowry, S. Person, and M. Pape, "Combining unit-level symbolic execution and system-level concrete execution for testing NASA software," in *Proc. of ISSTA*, 2008.

[14] J. Schumann, H. Jafari, T. Pressburger, E. Denney, W. Buntine, and B. Fischer. AutoBayes Program Synthesis System Users Manual. NASA/TM-2008-215366, 2008.

[15] B. Fischer and J. Schumann, "AutoBayes: A system for

generating data analysis programs from statistical models," *J. Functional Programming*, 13(3):483–508, 2003.

[16] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J. of the Royal Statistical Society series B*, 39:1–38, 1977.

[17] Cheeseman, P. and Stutz, J. Bayesian classification (AutoClass): Theory and results. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Proc. 2nd Intl. Conf. Knowledge Discovery and Data Mining*, pages 153–180. AAAI Press, 1996.

[18] McLachlan, G., Peel, D., Basford, K. E., and Adams, P. The EMMIX software for the fitting of mixtures of normal and t-components. *J. Statistical Software*, 4(2), 1999.

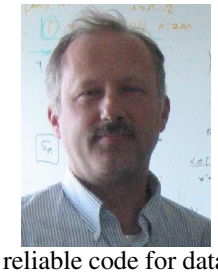
[19] Fraley, C. and Raftery, A. E. MCLUST: Software for model-based clustering, density estimation, and discriminant analysis. Technical Report 415, Department of Statistics, University of Washington, October 2002.

[20] Y. Hu, "Treatment learning," masters thesis, University of British Columbia, 2002.

[21] K. Gundy-Burlet, J. Schumann, T. Menzies, and T. Barrett, "Parametric analysis of Antares re-entry guidance algorithms using advanced test generation and data analysis," in *Proc. iSAIRAS*, 2008.

[22] Buntine, W. L. Operations for learning with graphical models. *J. AI Research*, 2:159–225, 1994.

BIOGRAPHY



Dr. Johann Schumann (PhD 1991, Dr. habil 2000, Munich, Germany) is a Senior Scientist with RIACS and working in the Robust Software Engineering Group at NASA Ames. He is engaged in research on verification and validation of autonomy software and adaptive controllers, and on automatic generation of reliable code for data analysis and state estimation. Dr. Schumann is author of a book on theorem proving in software engineering and has published more than 80 articles on automated deduction and its applications, automatic program generation, and neural network oriented topics.

Karen Gundy-Burlet Karen Gundy-Burlet is a research scientist in the Robust Software Engineering group at NASA-Ames Research Center. Her current interests are in the application of advanced information analysis techniques to validation and verification of simulation models and Guidance, Navigation and Control (GNC) software. She was the group lead for the NASA-Ames Intelligent Flight Control program that developed and implemented neural-adaptive algorithms for control of damaged aircraft, and demonstrated and evaluated them in full-motion piloted simulations. Other research efforts include optimal control of low-thrust spacecraft trajectories, and computational modeling of unsteady flows in turbomachinery with an emphasis on rotor/stator interaction in compressors and hot streak migration in turbines. She has interests in parallel systems and high-performance computing. Dr. Gundy-Burlet graduated with a B.S. in Mechanical Engineering from U.C. Berkeley and obtained M.S. and Ph.D. degrees in Aerospace Engineering from Stanford.

Corina Păsăreanu Dr. Corina Păsăreanu is a Research Scientist at the NASA Ames Research Center, Robust Software Engineering Group. She is working on using abstraction and symbolic execution in the context of software model checking and testing, and on automating assume-guarantee compositional verification. She has served on program committees for many meetings in the formal analysis area, such as CAV, ISSTA, FSE, and ICSE. She is also associate editor for the ACM TOSEM journal. More information can be found at: `http://ti.arc.nasa.gov/people/pcorina/`.

Tim Menzies Dr. Tim Menzies has been working on advanced modeling and AI since 1986. He received his PhD from the University of New South Wales, Sydney, Australia and is the author of over 164 refereed papers. A former research chair for NASA, Dr. Menzies is now a associate professor at the West Virginia University's Lane Department of Computer Science and Electrical Engineering. For more information, see `http://menzies.us`.

Anthony Barrett Dr. Anthony Barrett is a member of the Artificial Intelligence Group at the Jet Propulsion Laboratory, California Institute of Technology where his R&D activities involve planning & scheduling, plan execution, diagnosis, and test suite generation applied to spacecraft. He holds a B.S. in Physics, Computer Science, and Applied Mathematics from James Madison University and both an M.S. and PhD in Computer Science from the University of Washington. His research interests are in the areas of planning, scheduling, and execution/diagnostics in the context of single and multi-agent systems.