

PARAMETRIC ANALYSIS OF A HOVER TEST VEHICLE USING ADVANCED TEST GENERATION AND DATA ANALYSIS

Karen Gundy-Burlet¹, Johann Schumann², Tim Menzies³, and Tony Barrett⁴

¹NASA-Ames Research Center, Moffett Field, CA, 94035, Karen.Gundy-Burlet@nasa.gov

²RIACS/USRA, NASA-Ames Research Center, Moffett Field, CA 94035, Johann.M.Schumann@nasa.gov

³Lane CS & EE, West Virginia University, tim@menzies.us

⁴Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, barrett@jpl.nasa.gov

ABSTRACT

Large complex aerospace systems are generally validated in regions local to anticipated operating points rather than through characterization of the entire feasible operational envelope of the system. This is due to the large parameter space, and complex, highly coupled nonlinear nature of the different systems that contribute to the performance of the aerospace system. We have addressed the factors deterring such an analysis by applying a combination of technologies to the area of flight envelop assessment. We utilize n-factor (2,3) combinatorial parameter variations to limit the number of cases, but still explore important interactions in the parameter space in a systematic fashion. The data generated is automatically analyzed through a combination of unsupervised learning using a Bayesian multivariate clustering technique (AutoBayes) and supervised learning of critical parameter ranges using the machine-learning tool TAR3, a treatment learner. Covariance analysis with scatter plots and likelihood contours are used to visualize correlations between simulation parameters and simulation results, a task that requires tool support, especially for large and complex models. We present results of simulation experiments for a cold-gas-powered hover test vehicle.

1. INTRODUCTION

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 onboard software for the vehicle. This process also

enhances the validation and verification of the vehicle requirements and flight software. An initial prototype been of the MCB lander configuration entitled the Hover Test Vehicle (HTV) has been developed and flown in order to assess this spacecraft development methodology.

As with any complex aerospace system, the performance of the HTV 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 (traditional 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.

Researchers have recognized that “designers must be able to examine various design alternatives quickly and easily among myriad and diverse *configuration possibilities*” (1). The number of configuration possibilities within a model, such as the HTV described above, 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.

Part of the rapid prototyping process is to take advantage of exercising these analyses early in the design cycle. However the design space in this part of the life is generally very large, greatly expanding the analysis problem. In these early analyses, system prototypes can interact with physics models in order to quickly develop usage policies for the software. Given the ready availability of super-computers, and even inexpensive LINUX clusters,

analysts may have to analyze gigabytes of data generated automatically from simulators. Thus, the usage policies need to be generated in an efficient and accessible manner for the analyst.

Accordingly, since 2000 (2), we have explored sampling those configurations at random, running the resulting model, scoring the output with some oracle, then using data mining techniques to find the configuration options that most improve model output. Here, we try a new combination of methods and tools to study the configuration parameters on a software controller for the HTV:

- 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 (Section 3).
- The TAR3 minimal contrast set learner (3); is a supervised learning method that returns the minimal deltas between desired outcomes (hitting the target) and all the other outcomes (Section 4.1).
- 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 the governing parameters (Section 4.3).

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. Combinatorial 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 discusses the problem domain and how it was analyzed with combinatorial test, TAR3 and AUTOBAYES. We show that data mining in combination with *functional requirements* can be used to determine best parameter ranges for such tasks as safely hovering and landing the hover test vehicle.

2. TEST ARTICLE

The initial test article is a prototype of a lunar landing configuration. The vehicle is comprised of a main payload module and propulsion module. For this 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. For a complete description of the vehicle, please see ref (4).

The data mining techniques discussed in this paper have been applied to the Hover Test Vehicle (HTV) simulation. The HTV simulation has been developed in Mathworks Matlab and Simulink (5). It contains models of



Figure 1. Hover Test Vehicle

the vehicle including 6DOF dynamics, propulsion system, effectors, sensors and avionics. Also included are models of the ground data station, ground station, environment and the flight bus. The flight software is composed of guidance, navigation, control, state estimation and vehicle health management models. The flight software portion of the simulation is autcoded using Mathworks Real-Time Workshop Embedded Coder for control of the physical flight vehicle.

The model-based development environment both speeds the software development process and enhances the workflow for validation and verification of the flight software to NASA standards. An extensive unit test suite was developed in conjunction with the models to verify compliance with requirements. 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 envelop of the simulation and to find the ranges of parameters that lead to the lengthiest flights that landing close to the target. The margin between the best ranges to the failure points of each parameter can then be determined.

Both standard Monte Carlo and three-factor combinatorial techniques were used in this study. 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 parameter ranges are determined such that they are likely to include failure cases. The output data encompass a wide range of variables that are saved at each iteration point. 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. Figure 2 shows a 3D representation of 1000 simulated trajectories generated through a traditional Monte Carlo style test generation technique.¹

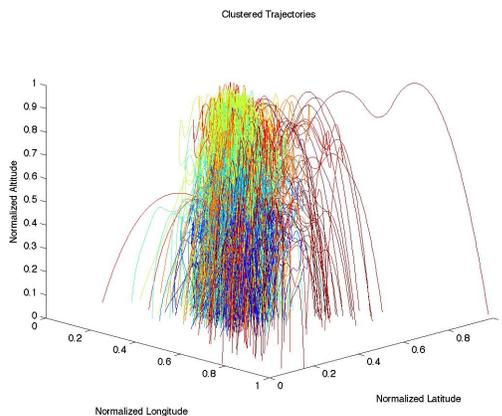


Figure 2. HTV trajectories for 1000 simulation runs with different simulation parameters.

3. COVERING THE OPTION SPACE

As mentioned previously, a model with 20 binary choices has more than a million possible configurations. For the ANTARES system it is anticipated that, in normal practice, the number of parameters to vary will greatly exceed 100, which results in an exponentially larger number of possible configurations. Worse yet, when dealing with simulations of physical systems, the input parameters are often real values, making choices non-discrete and the possible configurations infinite. So guaranteeing coverage of the option space is a non-trivial problem.

¹In this paper, colors encode the class membership as determined by the clustering algorithm. (Section 4.3, AUTOBAYES)

```

1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 1 1 1 0
0 1 1 0 1 1 1 0 1 1 1 0 0 0 1 0 0 0 0 1
1 0 1 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 0
0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 1
0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1
1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 1 1 0 0 0
1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1
0 0 1 1 1 1 1 1 0 1 0 1 1 1 0 0 0 0 0 0
1 1 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 1 1 0
1 0 1 1 1 1 0 1 0 1 0 0 0 0 0 0 1 1 1 0
0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1

```

Figure 3. A 2-factor combinatorial test suite for 20 binary parameters

3.1. Approach

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 from this approach is a lack of any coverage guarantee, resulting in a 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 is shown in Figure 3. Note that there are only 11 tests, much less than the million tests needed to exhaustively cover every combination. The 3-factor case only increases the number of tests to 26, still minuscule when compared to a million.

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 (6; 7; 8). The underlying premise behind the combinatorial approach can be captured in the following four statements.

- The simplest bugs in a program are triggered by a single input parameter.
- The next simplest bugs are triggered by an interaction of two input parameters.
- Progressively more obscure bugs 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. Applying

Problem Sizes	Number of Tests	Time
3^4	9	\ll 1 sec
3^{13}	19	\ll 1 sec
$4^{15} \times 3^{17} \times 2^{29}$	35	< 1 sec
$4^1 \times 3^{39} \times 2^{35}$	29	< 1 sec
10^{20}	216	1 sec
3^{1000}	48	22 sec

Table 1. Performance of test suite generator on 2-factor combinatorial problems

an n -factor combinatorial approach to testing ANTARES involved characterizing each real-valued parameter 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.

3.2. Implementation

To generate 2-factor combinatorial test suites there are a number of algorithms in the literature (9). Our algorithm is a generalization of IPO (10) to facilitate generating n -factor combinatorial test suites in addition to a number of features that a real world test suite generator would need (11). 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 with an example output that appears in Figure 3. Even with these extra capabilities the algorithm generates test suites that are comparable to those generated by the more restricted systems in the literature. As shown in Table 1 the code generates quality solutions very rapidly on a 400MHz Windows laptop. In the problem sizes, the X^Y syntax means that there are Y X -valued parameters.

4. DATA ANALYSIS TOOLS & METHODS

4.1. TAR3

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 $1, 1, \dots$ is the *apex* of the cube and represents the desired goal for the system. All the examples are scored by their normalized Euclidean distance to the apex.

For this study, outputs were scored on one dimension-distance to the desired target. Future studies will exploit more of BORE’s facilities and will score outputs on other dimensions such as average standard deviation on attitude control, minimum use of propellant, minimal maximum G-force, etc.

For each run i of the simulator, the n outputs X_i are normalized to the range $0 \dots 1$ as follows:

$$N_i = \frac{X_i - \min(X)}{\max(X) - \min(X)}$$

The Euclidean distance of $\{N_1, N_2, \dots\}$ to the ideal position of $\{N_1 = 1, N_2 = 2, \dots\}$ is then computed and normalized to the range $0..1$ as follows:

$$W_i = 1 - \frac{\sqrt{N_1^2 + N_2^2 + \dots}}{\sqrt{n}}$$

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 were 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 since this learning method 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.

TAR3 inputs a set of training examples E . Each example maps a set of attribute ranges to some class symbol; i.e., $\{R_i, R_j, \dots \rightarrow C\}$. The class symbols C_1, C_2, \dots are stamped with some utility score that ranks the classes; i.e., $\{U_1 < U_2 < \dots < U_C\}$. With E , these classes occur at frequencies F_1, F_2, \dots, F_C where $\sum F_i = 1$. A “treatment” T of size M is a conjunction of attribute ranges $\{R_1 \wedge R_2 \dots \wedge R_M\}$. Some subset of $e \subseteq E$ is consistent with the treatment. In that subset, the classes occur at frequencies f_1, f_2, \dots, 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:

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

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 the TAR3 treatment learner (13) that uses stochastic search to find its treatments.

4.2. TAR3: Results

When applied to our spacecraft re-entry simulator, BORE/TAR3 works as follows. Firstly, BORE generates a *baseline* distribution where the simulation outputs are divided into two categories: 25% are classified as *best* (closest to the target region) while the others were *rest* (the remaining 75% of the examples). The first row in Table 2 shows the initial setting; the *worth* of this scenario is defined to be one.

<i>Tr</i>	<i>worth</i>	Constraint	<i>best</i>	<i>rest</i>
–	1.0	–	25%	75%
1	1.46	$0 \leq c \leq 0.34 \wedge 0.68 \leq f < 1.0$	82%	18%
2	1.43	$0.67 \leq c < 1.0 \wedge 0 \leq f < 0.34$	79%	21%
3	1.15	$0.85 \leq c < 0.98 \wedge 0.67 \leq f < 1.0$	44%	56%

Table 2. TAR3 treatments

TAR3 then seeks the minimal delta between *best* and *rest*. Several candidate deltas are generated and scored according to their normalized *worth* with respect to the baseline. The top three deltas as produced by TAR3 are shown in Table 2. Each treatment consists of a conjunction of linear constraints on the input variables. For inputs adhering to the constraints of the treatment, their effect is calculated. In this case, a restriction of the input variables *c* and *f* (two of 24 simulation parameters) to the shown ranges causes substantial improvement over the initial setting, as the “best” and “rest” values, which

BORE produces, show. The relative improvement is indicated by the *worth* of the treatment.

Note that:

- The deltas with the highest worth have the greatest percentage of *best* examples.
- The top two treatments have very similar worths, and the third much less so.
- The top two treatments comment on different extremes of the variables *c*, *f*.
- Only very few of the 24 variables in the simulation appear in the treatments.

Of these effects, the last is most important. TAR3 is a focus tool that tells an analysts “of all the things you *might* think about, these few variables are all you *should* be considering”.

4.3. 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 class parameters, only given the data. A variety of algorithms can be used, for example, the EM-algorithm (14). A number of EM-implementations is available (e.g., Autoclass (15), EM-MIX (16), or MCLUST (17)) 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 (18) is a fully automatic program synthesis system that generates efficient and documented C/C++ code from abstract statistical model specifications. Developed at NASA Ames, AUTOBAYES is implemented in approximately 90,000 lines of documented SWI Prolog code. From the outside, it 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 processes the data or, in our case, can be called from Matlab.

On the inside, however, it works quite different: AUTOBAYES first derives a customized algorithm skeleton implementing the model and then transforms it into optimized C/C++ code. The input specification is translated into a Bayesian Network (19), which is a compact internal representation of the statistical model. Then, the program synthesis system uses a schema based approach to translate the statistical problem into smaller

```

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 4. AutoBayes specification for mixture model

problems and then, after symbolically solving subproblems, to transform the instantiated customized algorithm into efficient code. 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 problem variable of interest (i.e., observation or parameter), properties and dependencies are specified via probability distributions and constraints. Figure 4 shows how our clustering, a Gaussian mixture model with diagonal covariance matrices can be represented in AUTOBAYES’s specification language. 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 ϕ . 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 $\text{pr}(x|\{\phi, \mu, \sigma\})$ with respect to the parameters of interest, ϕ, μ , and σ . This means that we are interested in a maximum likelihood estimate (MLE) of the model parameters.

If additional domain knowledge, e.g., priors on the mean values of the features for each class are known, more

```

const double mu_0(1..D, 1..C) as 'expected means';
const double kappa_0(1..D, 1..C) as 'confidence';
with 1 < kappa_0(_i, _j);
const double sigma_0(1..D, 1..C);
double mu(1..D, 1..C);
mu(_i, _j) ~ gauss(mu_0(_i, _j),
sigma(_i, _j)*kappa_0(_i, _j));
...
max pr({mu, sigma, x}|phi) wrt {phi, mu, sigma};

```

Figure 5. Additions to AutoBayes specification for Gaussian mixture model with priors (additional domain knowledge)

complicated models (e.g., maximum a posteriori estimates, MAP) can be easily specified. Only a few lines (Figure 5) with specification of the prior and an updated maximization goal is necessary to produce a substantially different data analysis algorithm. Note that all these models are completely declarative and do not require the user to prescribe any algorithmic aspects of the estimation program.

5. RESULTS

The data produced by the simulation are actually time series data over a large number of variables. Characteristic parameters were extracted from the data stream and utilized in this analysis. Data dimensions obviously include the landing position, the sum of consumed fuel, maximal structural loads, as well as a measure of the duration of extended time intervals where the gravitational forces exceed a safe limit. With this preprocessing, we obtained a data set with 10 dimensions. All data were normalized.

These data then were clustered using the Matlab/C code as was generated by AUTOBAYES (790 lines of documented 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:

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

- Horizontal velocity on Landing no greater than 1.0 M/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 should be determined.

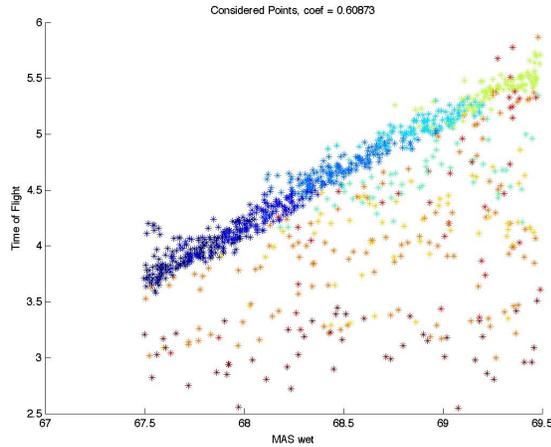


Figure 6. Relationship between Time of Flight and Initial Wet Mass Colors indicate class membership.

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.

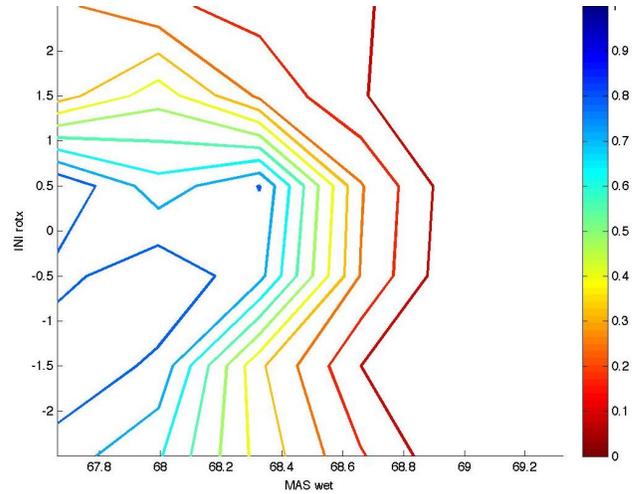


Figure 7. Likelihood of successful flight relative to initial x rotation and mass

6. CONCLUSIONS

In this paper, we have explored a combination 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.

Acknowledgements Part of this work was carried out at JPL, California Institute of Technology, under a contract with NASA.

REFERENCES

- [1] Gray, J., Lin, Y., and Zhang, J. Automating change evolution in model-driven engineering. *IEEE Computer*, 39(2):51–58, February 2006.
- [2] 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>.
- [3] Menzies, T. and Hu, Y. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from <http://menzies.us/pdf/03tar2.pdf>.
- [4] Bell, J., E. A. Hover testing of a prototype small planetary spacecraft. In *Submitted to IAC'08, the 59th International Astronautical Congress, Glasgow*, 2008.
- [5] URL: <http://www.mathworks.com/>.
- [6] Cohen, D., Dalal, S., Parelius, J., and Patton, G. The combinatorial design approach to automatic test generation. *Software, IEEE*, 13(5):83–88, Sep 1996.
- [7] Dunietz, I. S., Ehrlich, W. K., Szablak, B. D., Malloys, C. L., and Iannino, A. Applying design of experiments to software testing: experience report. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 205–215, 1997.
- [8] 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.
- [9] Mats Grindal, Jeff Offutt, S. F. A. Combination testing strategies: a survey. *Software Testing, Verification and Reliability*, 15(3):167–199, 2005.
- [10] Tai, K. and Lie, Y. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, 2002.
- [11] Czerwonka, J. Pairwise testing in real world, practical extensions to test case generators. In *Proceedings of 24th Pacific Northwest Software Quality Conference*, 2006.
- [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. Available from <http://www.ics.uci.edu/~pazzani/Publications/stucco.pdf>.
- [13] Hu, Y. Treatment learning, 2002. Masters thesis, University of British Columbia, Department of Electrical and Computer Engineering. In preparation.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] Fischer, B. and Schumann, J. AutoBayes: A system for generating data analysis programs from statistical models. *J. Functional Programming*, 13(3):483–508, May 2003.
- [19] Buntine, W. L. Operations for learning with graphical models. *J. AI Research*, 2:159–225, 1994.