



National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

A Parametric Testing Environment for Finding the Operational Envelopes of Simulated Guidance Algorithms

Anthony Barrett
Jet Propulsion Laboratory
California Institute of Technology





Motivation

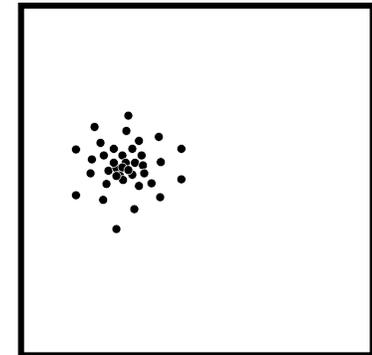


- **The Problem**

- As NASA missions become ever more complex and subsystems become ever more complicated, testing for correctness becomes progressively more difficult. Exhaustive testing is usually impractical, so how does one select a smaller set of test cases that is effective at finding/analyzing bugs?

- **State of the Art**

- Currently missions address this problem by performing Monte Carlo tests and analyzing the results. Unfortunately, this approach does not provide any coverage guarantees, does not provide any help in actually analyzing the results, and limits testing to small regions of the option space.



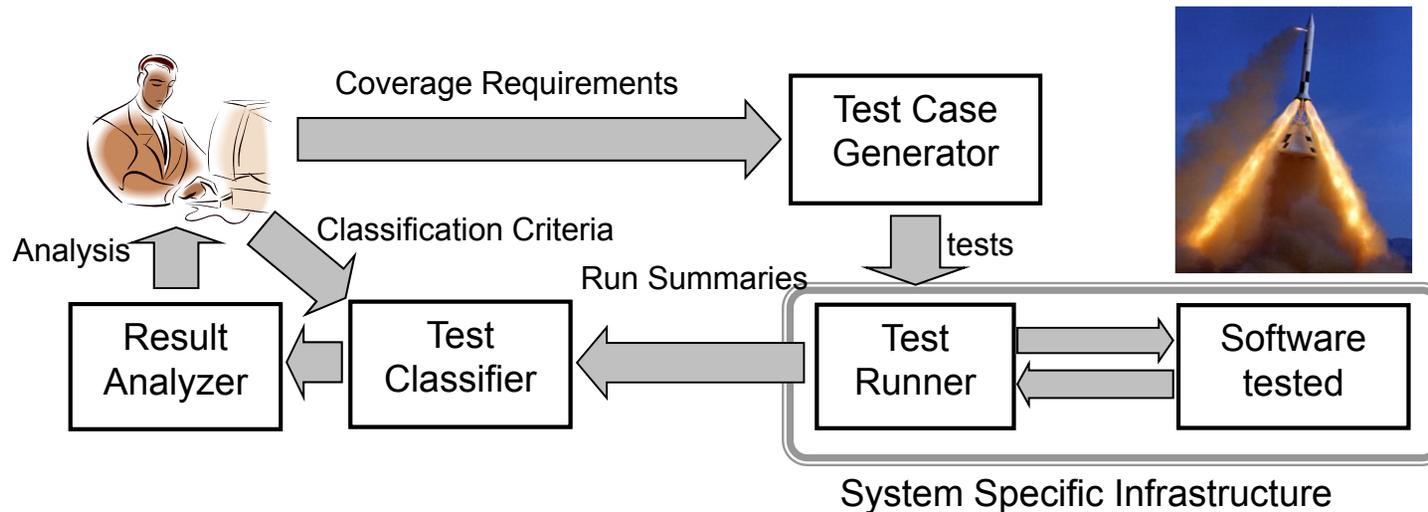


Motivation



- **Solution Presented Here**

- Let an analyst pose test-space coverage requirements and then refine these requirements to focus on regions of interest in response to visualized test results.
- Instead of validating correctness around set points (with Monte Carlo analysis) find and characterize the margins of the performance envelop where the system starts to fail.





Outline



- **Example Scenario**
- **Test Case Generation**
- **Simulation Classification**
- **Test Space Analysis**
- **Margin Analysis**
- **Visualization**
- **Changing The Test Space**



Example Scenario



- Orion Launch Abort System

Orion

Attitude Control
Motor (ACM)

Jettison Motor (JM)

Abort Motor (AM)

Crew Module (CM)

Service Module (SM)

Launch
Abort
System
(LAS)

Orion

Launch
Abort
Vehicle
(LAV)

QuickTime™ and a decompressor are needed to see this picture.

Ares

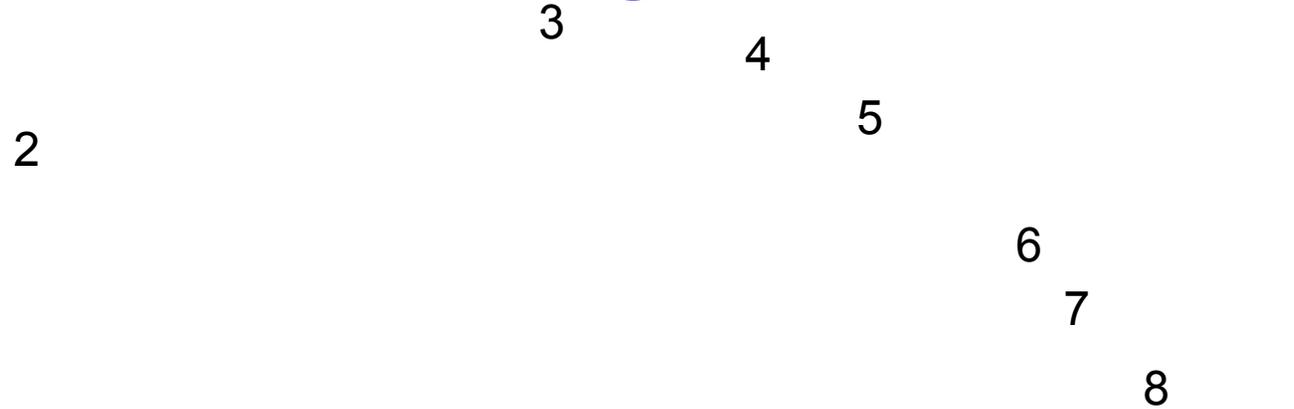
Idicula, J., Williams-Hayes, P., Stillwater, R., and Yates, Lt. M., "A Flight Dynamics Perspective of the Orion Pad Abort One Flight Test," AIAA-2009-5730



Example Scenario



• Launch Abort PA-1 Flight Profile



QuickTime™ and a decompressor are needed to see this picture.

- | | | |
|--|--|---|
| <ol style="list-style-type: none"> 1. Abort sequence initiated
AM ignition
ACM ignition
Liftoff 2. Reorient LAV 3. LAS jettison
LAS JM ignition 4. FBC jettison
FBC chute mortars fire | <ol style="list-style-type: none"> 5. Drogue mortars fire 6. Drogues at full stretch 7. Drogue attach cutters fire 8. Mains deploy
Main canopy pilot mortars fire
Mains full line stretch 9. Touchdown 10. Mains attach cutters fire | <ol style="list-style-type: none"> 9 10 |
|--|--|---|

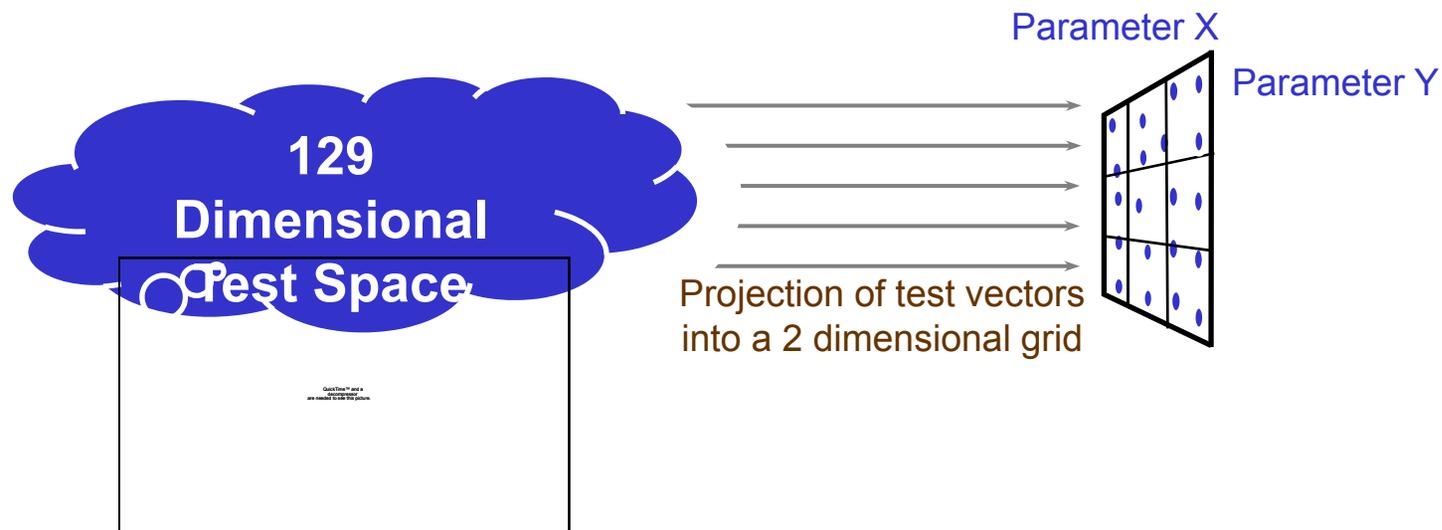
Idicula, J., Williams-Hayes, P., Stillwater, R., and Yates, Lt. M., "A Flight Dynamics Perspective of the Orion Pad Abort One Flight Test," AIAA-2009-5730



Test Case Generation



- The launch abort PA-1 scenario had 129 independent parameters, resulting in a 129D test space.
 - We want to sample the test space such that any projection is evenly covered.





Test Case Generation



- Instead of random sampling, we use combinatorial methods
 - For pairwise testing of 13 ternary variables requires only 19 tests, and 33 for 129 ternary variables.
 - For continuous parameters we randomly sample from bins as specified by the ternary variables

This test suite contains only 19 test cases, yet it exercises every pair-wise combination of 13 parameters!

	Parameters												
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
1	0	0	0	0	1	2	2	1	0	2	0	1	2
2	1	0	2	2	2	0	1	0	1	0	1	2	0
3	2	0	1	1	0	1	0	2	2	1	2	0	1
4	1	2	0	1	1	0	0	1	1	1	0	0	0
5	0	2	2	0	0	1	1	2	0	0	1	1	1
6	2	2	0	2	2	2	2	2	2	2	2	2	2
7	0	1	1	2	1	0	2	0	0	1	2	1	0
8	1	1	0	0	0	2	1	0	2	2	0	0	1
9	2	1	2	1	1	2	1	1	1	0	1	2	2
10	1	2	1	0	2	1	2	1	1	2	2	1	1
11	2	1	2	0	2	0	0	2	0	2	0	2	0
12	0	2	1	1	2	2	0	0	0	0	0	0	2
13	0	1	2	2	0	0	2	1	2	0	1	0	2
14	1	1	0	2	1	1	0	2	0	0	0	1	2
15	2	2	0	1	1	1	2	0	2	2	1	1	0
16	0	0	1	0	0	2	1	2	1	1	1	2	0
17	2	2	2	2	2	1	1	0	2	1	2	2	2
18	0	1	2	2	1	0	0	2	0	0	2	2	1
19	1	0	0	0	1	2	0	0	0	1	1	0	0



Issues With Random Approach



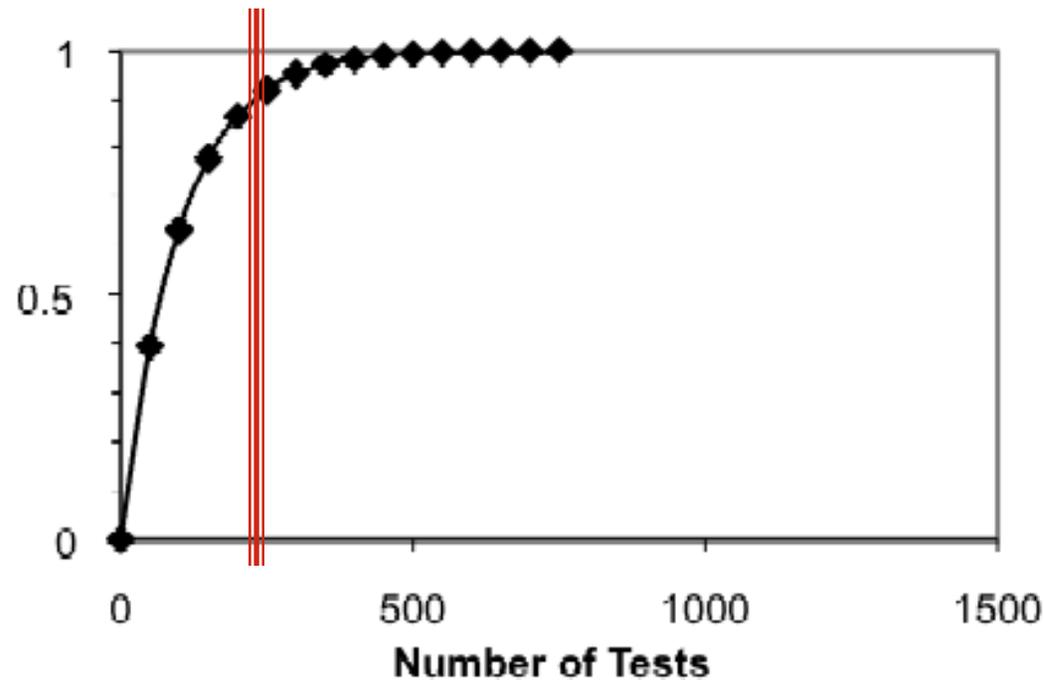
- **How many tests are enough?**
 - Typically just perform tests until some time limit is reached, resulting in large numbers of tests
- **What kind of guarantee does this provide?**
 - At best a probabilistic guarantee...



Finding Some Pairwise Problem



- For 20 ten-valued parameters (10^{20})
 - When is random testing good enough?

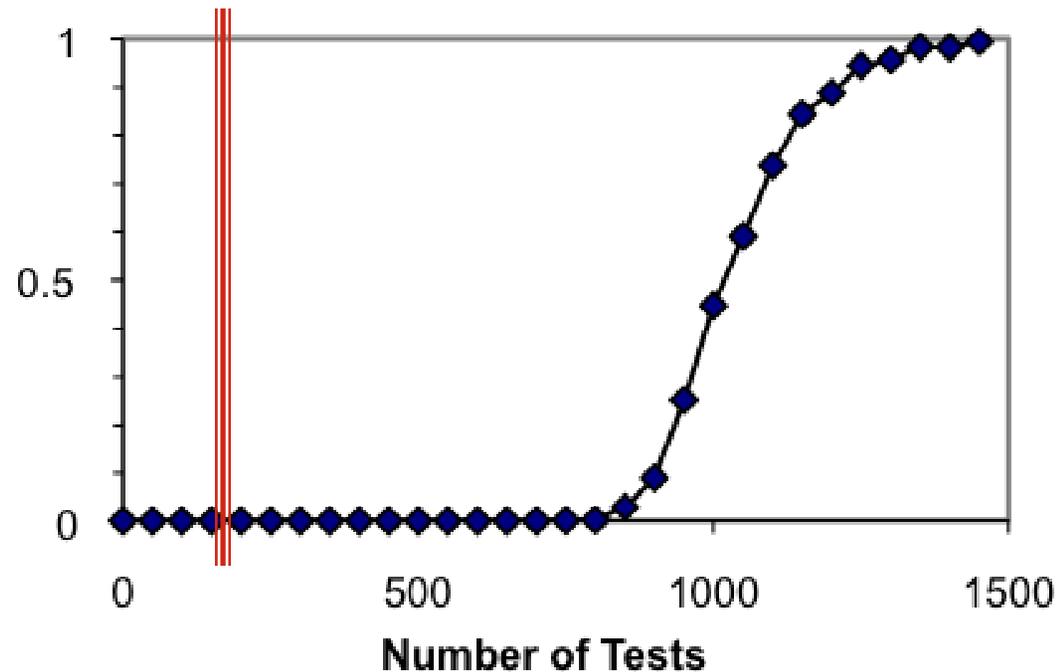




Finding Any Pairwise Problem



- For 20 ten-valued parameters (10^{20})
 - Not quite if you want a guarantee





Pair-wise Performance



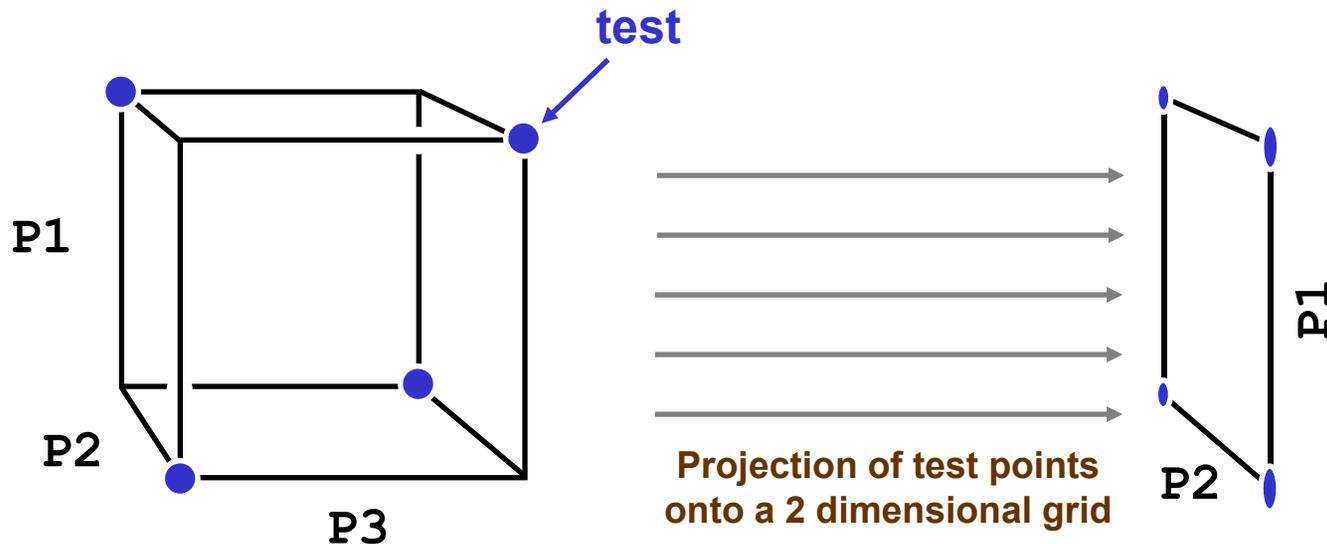
Factor Sizes	# of Tests	Time
3^4	9	<<1 sec
3^{13}	19	<<1 sec
$4^{15}3^{17}2^{29}$	35	<1 sec
$4^{13}3^{39}2^{35}$	29	<1 sec
10^{20}	216	1 sec
3^{1000}	48	22 sec



From a Graphical Perspective



- For pair-wise coverage, every projection of the tests in the 3 dimensional test space onto a 2 factor plane fully covers the grid.





Underlying Premise



- The simplest bugs in a program are generally triggered by a single input parameter
- The next simplest bugs are triggered by an interaction between two input parameters
- Progressively more obscure bugs involve interactions between more parameters
 - These are progressively rarer and harder to test for
- Exhaustive testing involves testing all possible combinations of all inputs.
 - This blows up exponentially with the number of inputs



Test Coverage Guarantees



- **Command to add tests to the test space**

```
addtests granularity nary { parameter }
```

- `granularity` -- Number of grid lines in projections.
- `nary` -- Number of dimensions over which to guarantee coverage.
- `{ parameter }` -- Limit guarantee to just over specified parameters.

- **Example:**

```
addtests 3 2
```

- This command generates tests that are guaranteed to have at least one test in every box of any projection onto a 3x3 grid.
- In the case of the PA-1 domain, this only required 33 tests.

- **Important property:**

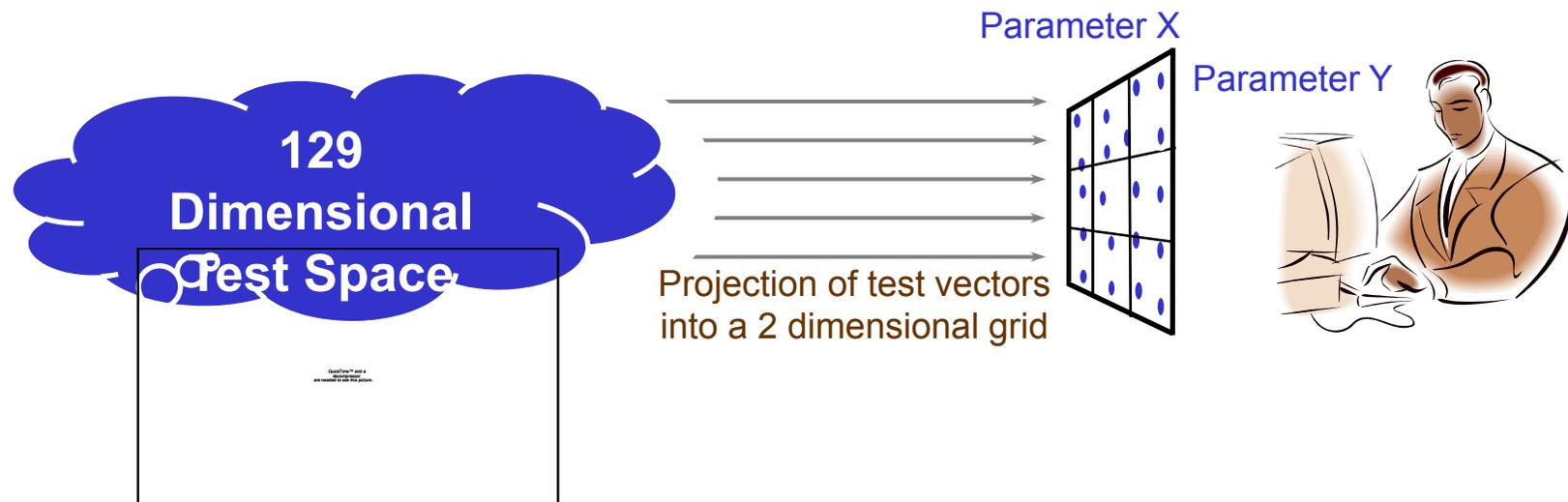
- This command generates completely different tests each time it is invoked. Invoking it twice will double the coverage guarantee with completely different tests.



Test Space Analysis



- Visualizing a 129 dimensional test space is out of the question.
 - There are $129 \times 128 = 16,512$ projections onto two dimensions.
 - At a second per projection it takes 4.5 hours to inspect them all, and 72 work days for all 3D projections
 - To speed this process, we apply learning techniques.





Analysis Via Treatment Learning



- **Command to learn treatment rules**

```
learn granularity nary percent
```

- **granularity** -- Number of grid lines
- **nary** -- Maximum number of parameters
- **percent** -- Rules must match at least this percentage of target class

- **Example:**

```
learn 5 3 50
```

- This command heuristically looks at the set of 1 to 3 parameter rules for those rules that raise the percentage of the target class the most while also containing at least 50% of the target class.

- **This command is based on a treatment learner**

- The learned rules are not perfect in that they can miss some of the target class, and they can match some of the non-target class.
- G. Gay, T. Menzies, M. Davies, and K. Gundy-Burlet. “Automatically finding the control variables for complex system behavior.” *Automated Software Engineering* 17(4), December 2010, pp 439-468.

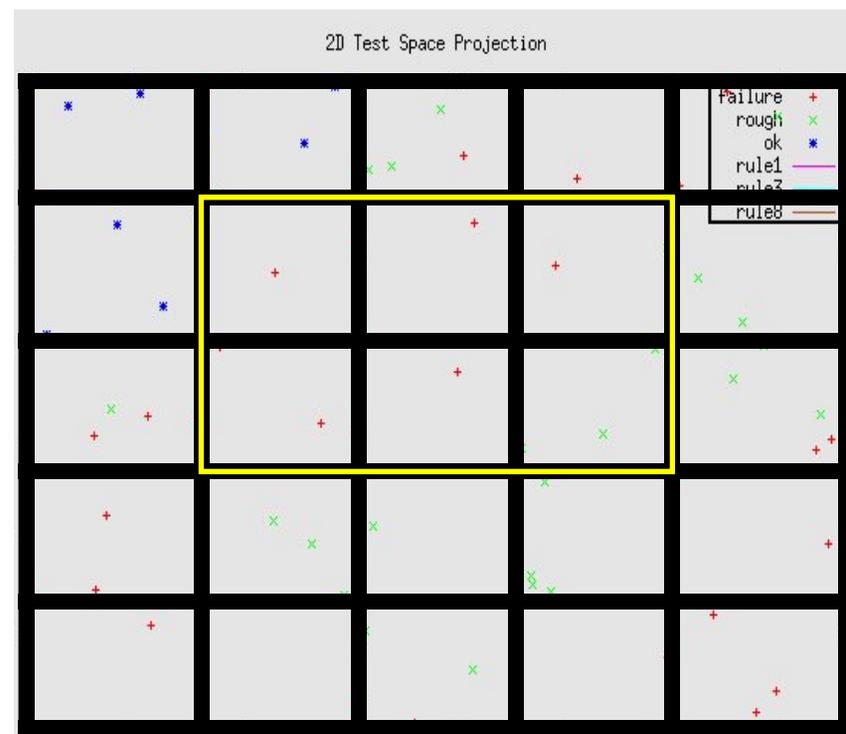
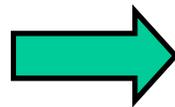


Treatment Learning Problem



- learn 5 3 50 -- Among all projections onto 1D lines, 2D boxes or 3D volumes (over grid lines) return the boxes that raise the ratio of successes the most while containing at least 50% of the successes.

One of 16,512 projections

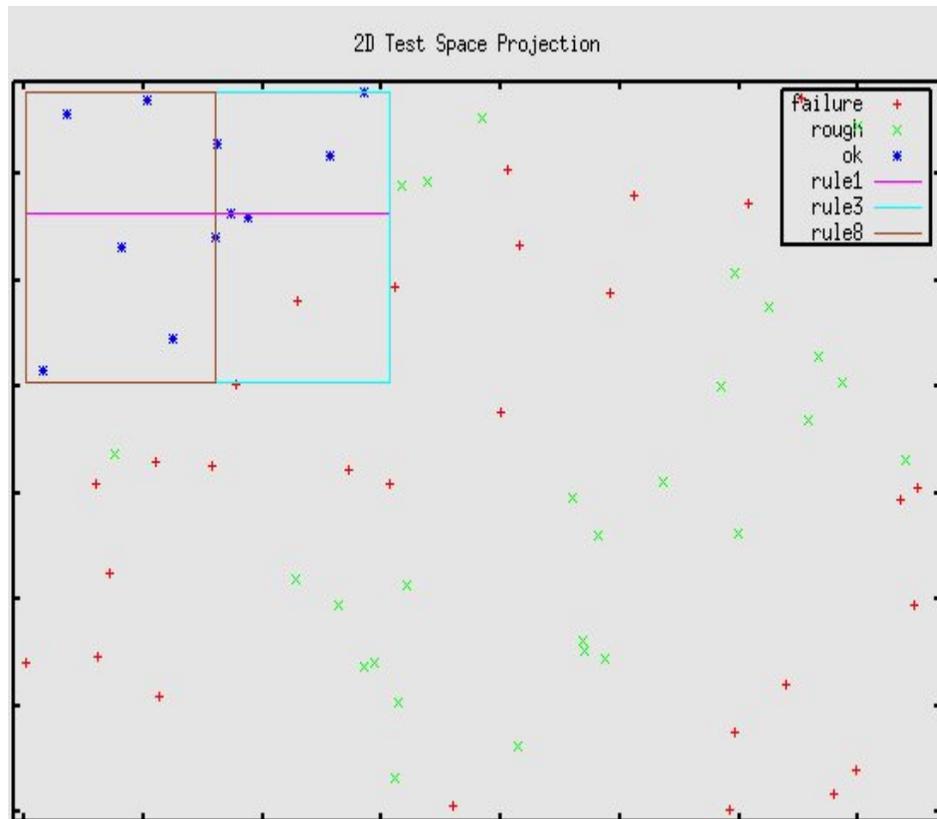




A Treatment Learning Example



- A two element treatment rule can be projected as a box like the following. These rules were generated with: `learn 5 3 50`



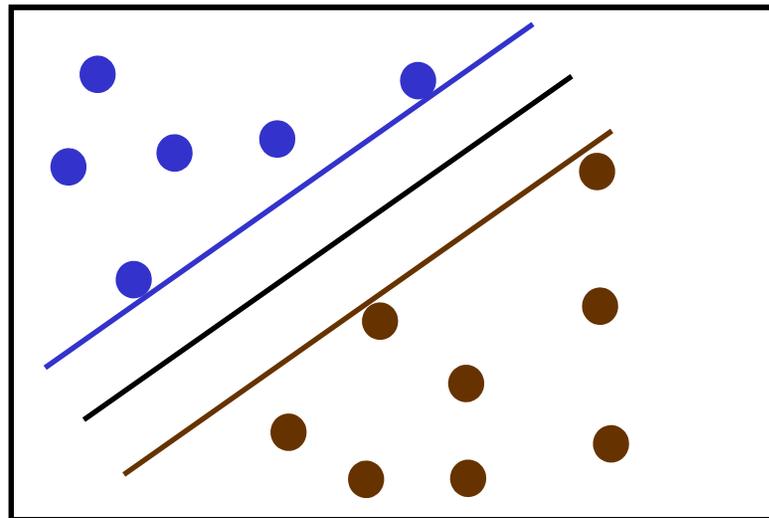
- It took <5 minutes to learn the treatment rules.
- The learned rules direct our attention to this projection.
- This is a lot faster than 4.5 hours!

This projection illustrates a linear margin involving two of the Launch Abort System propellant tank's dry inertias

- **Command to learn a linear margin**

SVMlearn ratio

- **ratio** -- relative importance for being 100% correct vs having larger separations. This is normally 1 for enforcing correctness, but can be relaxed if the tests are not linearly separable.
- **This command is based on svm_light.**
 - T. Joachims, “Making Large-Scale SVM Learning Practical” In Advances in Kernel Methods -- Support Vector Learning. MIT-Press 1999





Simplifying a Margin



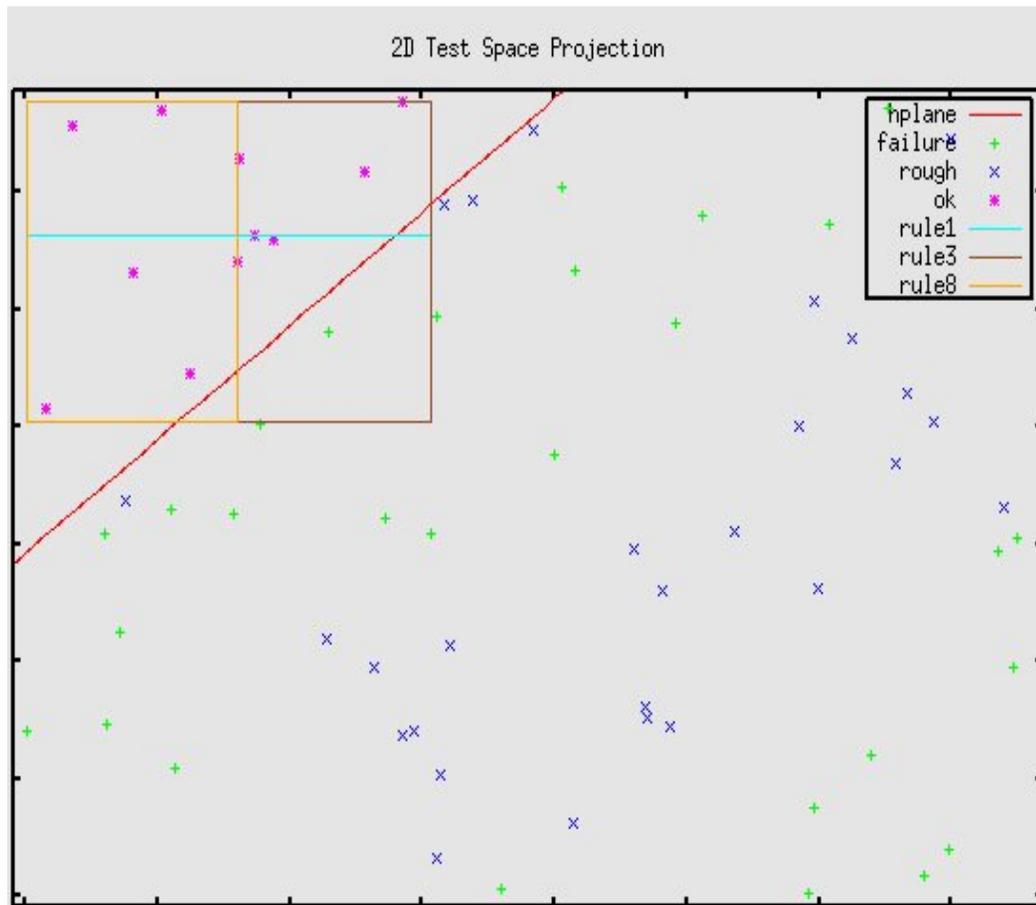
- **Problem: The learned margin is $K-1$ dimensional, where K is the dimension of the test space.**
 - **Not very easy to visualize.**
- **Command to simplify a margin**
`SVMsimplify ratio`
 - **ratio -- remove those parameters that do not affect the affect the hyperplane as ratio of the most important parameter's effect.**
- **As the ratio increases to 1 more parameters are excluded, but the hyperplane can also become incorrect.**



Example of a learned margin



- A two component linear margin is projected as a line in a 2D projection. This one was generated with SVMsimplify 0.1



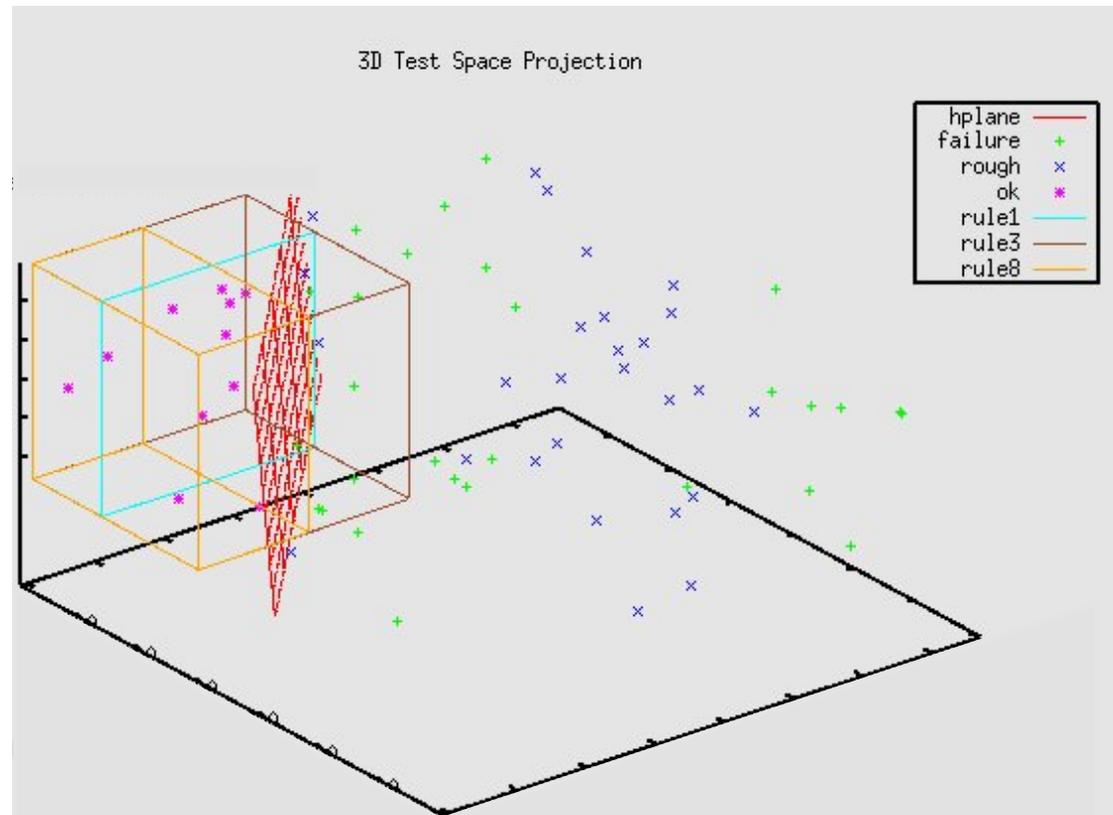
- While the simplified margin was 100% correct, it did have a lower separation than the original hyperplane.
- This suggests that extra parameters contribute.
- SVMsimplify 0.01 gets the extra parameter.



Example of a Learned Margin



- Final analysis of this margin shows that differences in the Launch Abort System propellant tank's dry rotational inertias can cause issues, and one of the crew module's rotational inertias also participates in the effect.

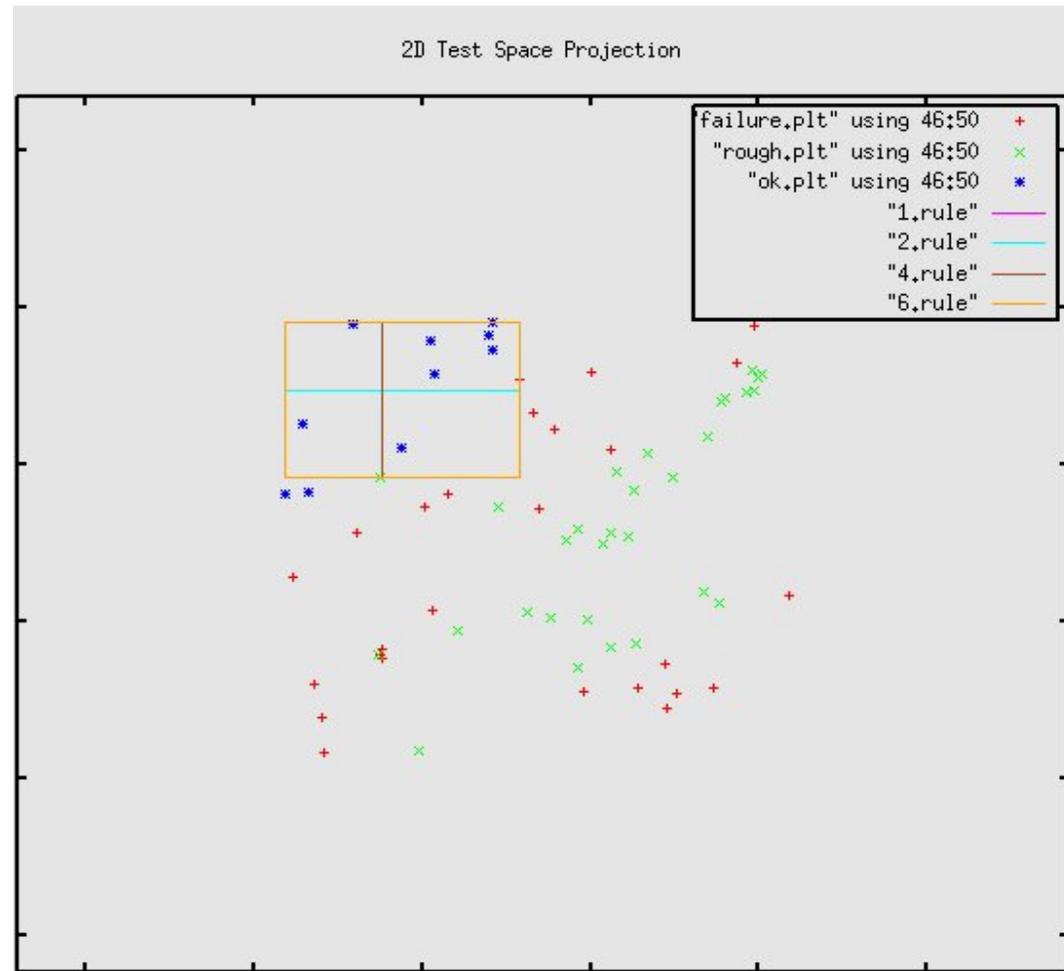




Looking for Another Margin



- Expand the focus and then add more tests

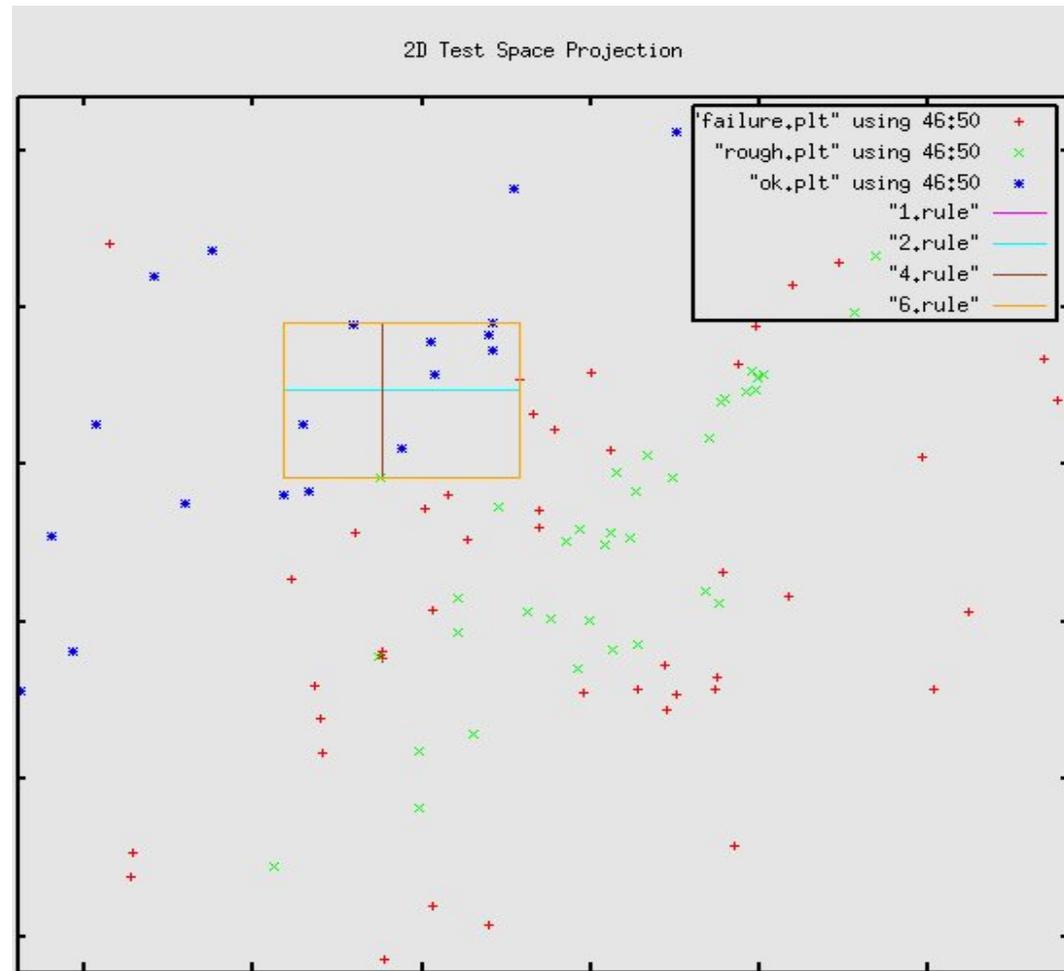




Looking For Another Margin



- Found another margin to analyze.





Final Result



- **It took under three hours to find/document two distinct margins.**
 - **Two hours to run the 132 simulations with 5 linux machines**
 - **15 minutes to analyze/visualize simulation results**
- **Initial physical interpretation of margins**
 - **Differences between dry rotational inertias of the Launch Abort System propellant tank strongly affect performance.**
 - **The Crew Exploration Vehicle's rotational inertias had a lesser effect**
 - **Other components had affects to an even lesser degrees.**



Conclusions



- **Combinatorial methods drastically reduced the number of tests needed to evenly cover the test space with respect to projections.**
 - **The first margin was detected and analyzed with only 66 tests.**
- **129 dimensions requires mechanical assistance for analysis.**
 - **Finding simple regions that improve the ratio of target class to everything else also identifies the best projections to inspect for margins.**
 - **Finding separating hyperplanes identifies those parameters that participate in an observed margin.**
 - **At all times, projection/visualization facilitates validating a margin's correctness.**



Copyright



- **Copyright 2011 California Institute of Technology.
Government sponsorship acknowledged.**