A NEW MONTE CARLO FILTERING METHOD FOR THE DIAGNOSIS OF MISSION-CRITICAL FAILURES

GREGORY GAY, TIM MENZIES * AND MISTY DAVIES, KAREN GUNDY-BURLET†

Abstract. Testing large-scale systems is expensive in terms of both time and money. Running simulations early in the process is a proven method of finding the design faults likely to lead to critical system failures, but determining the exact cause of those errors is still time-consuming and requires access to a limited number of domain experts. It is desirable to find an automated method that explores the large number of combinations and is able to isolate likely fault points.

Treatment learning is a subset of minimal contrast-set learning that, rather than classifying data into distinct categories, focuses on finding the unique factors that lead to a particular classification. That is, they find the smallest change to the data that causes the largest change in the class distribution. These treatments, when imposed, are able to identify the settings most likely to cause a mission-critical failure. This research benchmarks two treatment learning methods against standard optimization techniques across three complex systems, including two projects from the Robust Software Engineering (RSE) group within the National Aeronautics and Space Administration (NASA) Ames Research Center. It is shown that these treatment learners are both faster than traditional methods and show demonstrably better results.

Key words. treatment learning, stochastic search, minimal contrast-set learning, simulation, testing

AMS subject classifications. 62P30, 65C50, 65K10, 68T20, 68W20

1. Introduction. Large-scale industrial systems, often containing both hardware and software components, are expensive to design and expensive to build. The cost of failures decreases exponentially the earlier that they are discovered in the design process [8]. Modeling and simulation of these large-scale systems has become increasingly popular over the past decade as a proven method of finding and eliminating potentially costly fault points before a complete prototype is constructed.

However, simply simulating the behavior of a system is not enough. Knowing that something fails in simulation is not the same as knowing *why* it failed. No matter what domains a system primarily falls into, there are a limited number of experts and their time is even further limited. It is cost-prohibitive to ask those experts to pore over gigabytes of simulation settings and outcomes. Therefore, it is desirable to find methods of eliminating that bottleneck: methods that either reduce the amount of data that experts need to examine or methods that can identify the most obvious faults automatically.

The field of data mining uses techniques from statistics and artificial intelligence to find small, yet relevant, patterns in large sets of data. The standard practice in this field is to *classify*, to look at an object and make a guess at what category it belongs to. As new evidence is examined, these guesses are refined and improved. When testing a complex hardware system, you might try to classify by guessing whether that particular simulation *succeeded* or *failed*.

Treatment learning [27] focuses on a different goal. It does not try to determine *what is*, it tries to determine *what could be*. Classifiers take past evidence and use that to categorize new data. Treatment learners work in reverse. They take the

---
*Lane Department of CS and EE, West Virginia University, Morgantown WV (greg@greggay.com, tim@menzies.us).

†NASA Ames Research Center, Moffett Field CA (misty.d.davies@nasa.gov, karen.gundy-burlet@nasa.gov).

classification of a piece of evidence and try to identify the factors that led to that classification. Rather than deciding whether a simulation succeeded or failed, you want to determine why it failed. These learners take those factors and produce a minimal treatment—a set of rules that, if imposed, will change the expected class distribution. By only accepting examples that follow the rules set in the treatment, you improve the odds that these examples will fall within your target class.

Ultimately, classifiers will strive to increase the representational accuracy. They will stack any evidence and impose any rules in order to accurately guess what category a new example belongs to. If the target is complex, the resulting tree will also be complex. Treatment learning focuses on minimality: what is the *smallest* rule that can be imposed to cause the *largest* change.

Stated formally, treatment learning is a form of minimal contrast-set association rule learning. The treatments contrast undesirable situations with the desirable ones (represented by weighted classes). Treatment learning, however, is different from other contrast-set methods like STUCCO [6] because of its 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 lies in quickly pruning unpromising attribute ranges.

This study utilizes the TAR3 [18–20] and TAR4.1 treatment learners, which operate similarly but score treatments in radically different manners. In order to assess the effectiveness of these learners, they must be benchmarked against standard methods. Fundamentally, both of the treatment learning algorithms are attempting to maximize a set mathematical objective function. Therefore, they can be compared to standard optimization algorithms. Specifically, this research tests the dedicated treatment learners against Simulated Annealing [22, 28] and Quasi-Newton [17] algorithms using similar objective functions. The results of this study clearly demonstrate that:

- Treatment learners are orders of magnitude faster than standard methods.
- TAR3's treatments are more precise than those from standard techniques.
- TAR4.1's treatments demonstrate a higher recall, while maintaining a lower false positive rate, than the standard techniques.

2. Practical Implications. NASA often uses high-fidelity physics simulations early in the design process to verify that flight software will meet the mission requirements. The possible inputs to the simulation can be design parameters like lift coefficients and center of gravity positions; they may also be environmental parameters like the average magnitudes and the standard deviations of wind gusts; they may be flags that indicate the failure of a hardware component at some critical time; or they may be any of a plethora of other parameters that specify the bounds on the acceptable flight envelope. Errors in software design are much less expensive to fix early in the design process, but the design space early in the process is very large. To explore all of the parameter combinations exhaustively is infeasible. However, a very large sampling of the configurations increases the chance that a design error will be caught early, and it allows for the possibility of identifying trends or anomalies within the data.

Manual inspection of these large datasets requires domain expertise, and is likely to focus only on absolute compliance with requirements. For systems this complex many different kinds of failures are possible. For aero-braking scenarios, for instance, representative failures include skipping out of the Earth's atmosphere instead of re-entering and parachutes failing to open. One common heuristic for these analyses is

to push the bounds of the flight envelope until between 10 and 30 percent of all of the attempted cases have failed in one way or another—this kind of analysis allows you to find the margins to failure within the system. In the probable event that failures are identified this early in the process, it is a time-consuming task to determine the cause of those failures; the failures may be associated with environmental factors (e.g. strong sustained wind gusts overwhelm the control system), they may be associated with software errors in the unit under test (e.g. the gains on the control system are set incorrectly for the nominal case), or they may be associated with software errors in the simulation used to perform the test (e.g. a legacy environmental model in the simulation uses different units than expected). A likely first step towards determining the cause of any failures is to find the input parameters that the failures are associated with. Even this first step is non-trivial—there are usually hundreds of input parameters associated with each dataset, and those parameters were chosen from thousands of input parameters to the actual simulation.

Two of the datasets used for this paper were generated using the Advanced NASA Technology ARchitecture for Exploration Studies (ANTARES) [1] simulation tool. The two datasets represent the Monte Carlo runs done for a re-entry study and for a launch abort study. An example set of trajectories for one of the studies is shown in Figure 2.1. The collected variables include environmental parameters, internal simulation values (like random seeds), and spacecraft state specifications—including both continuous and modal information. Both of these datasets had many different possible failure types. The third dataset referenced here was collected during a bicycle ride. The data comes from a bicycle power meter that calculates the power output and collects the following data at 60 Hz: distance traveled, heart rate, speed, wind speed, cadence, elevation, hill slope, and temperature. The power calculation for this particular meter could be very noisy and there was an open question about what measured parameters were most associated with this noise.

To decrease the amount of time necessary to isolate suspicious inputs, the Robust Software Engineering (RSE) group at Ames utilizes a multi-step process [18, 19, 35] that includes targeting tests with n-factor combinatorial test vectors and sorting the data into clusters with an unsupervised EM algorithm [15] in order to find anomalies and to aid visualization. This tool is known as the Margins Analysis. A key component in the Margins Analysis is the use of a treatment learner to tie behaviors in the dataset to their associated variables and ranges. The treatment learner is used many times throughout the analysis process. It first is used to find variables associated with the overall performance; a penalty is built for each dataset that accounts for all failures and often includes some continuous metric like a target miss distance. The treatment learner is then used to find the parameters associated with each individual type of failure; in practice there are 10's to 100's of different possible failure types identified for each dataset. Finally, the treatment learner is used within a loop to discover the variables associated with each unsupervised cluster. This analysis aids the researcher in understanding the underlying structure of the dataset and can uncover structure in the dataset that leads to new requirements.

3. Related Work. The work being performed here—choosing the inputs and ranges most likely to lead to some output—can be thought of as a type of sensitivity analysis known as Monte Carlo Filtering [32]. The heavy lifting in most sensitivity analyses of this type is currently being done either with straight linear correlation between the output class and the inputs, a method known as regional sensitivity analysis (RSA) which relies heavily on standard statistical tests such as the Smirnov
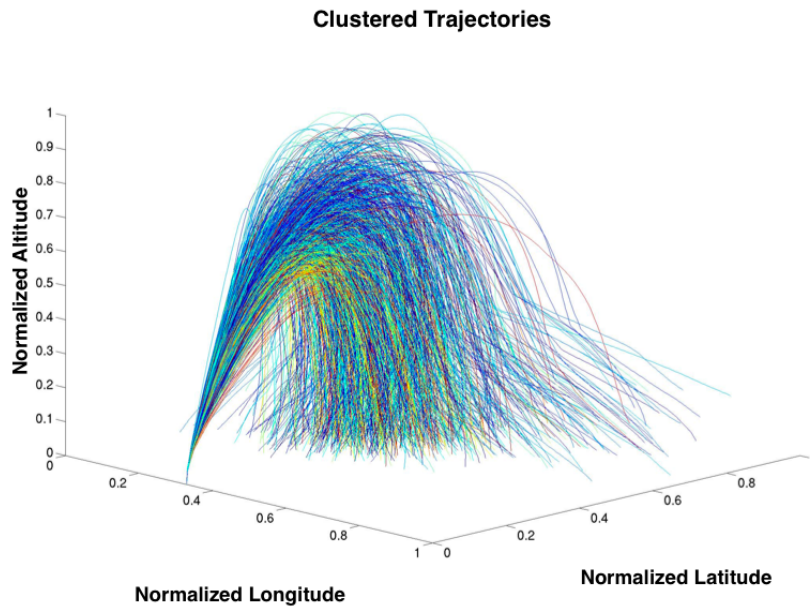
Clustered Trajectories

FIG. 2.1. *Visualization of trajectory data.*

two-sample test, or it is being done using some sort of regression analysis in which the relationships between the inputs and the outputs are derived from the data [4, 29, 34]. Linear correlations fail in models which are not smooth. For large models, RSA tends to have a very low success rate [39]. Regression analysis builds some sort of polynomial relationship by finding the correlation coefficients between the inputs and outputs. These correlation coefficients are then used to solve the original question—which inputs and their ranges most affect the output—by looking at the magnitudes of the correlation coefficients across the entire range. Regression analysis is computationally expensive, and tends to be limited to relatively small numbers of theoretically independent inputs, and also tends to assume that relationships between the inputs and outputs are smooth [29, 34].

The types of problems being solved in this paper are non-smooth and of high dimensionality. To overcome the complications involved in finding the correlation coefficients for this sort of problem, we choose in practice to ignore the correlation coefficients altogether and use machine learning techniques that sample the space and solve the original question directly. One example of another existing sensitivity analysis that uses machine learning is the identification of tool faults in the semiconductor industry. Intel uses a technique similar in spirit to the RSE analysis to find spatial fault patterns on silicon wafers [21]. While the overall goal and appearance of Intel's method is comparable to RSE's, the details for every step of the analysis are very different and they do not use treatment learning for their analysis [13, 40, 42]. The fact that parametric testing is being used across widespread applications demonstrates its promise; the massive divergence in the individual components of the technique is evidence that there is still significant research to be done to streamline its use for real-world data.

```
LSTAT <= 14.98
|   RM <= 6.54
|   |   DIS <= 1.6102
|   |   |   DIS <= 1.358: high (4.0/1.0)
|   |   |   DIS > 1.358
|   |   |   |   LSTAT <= 12.67: low (2.0)
|   |   |   |   LSTAT > 12.67: medlow (2.0)
|   |   DIS > 1.6102
|   |   |   TAX <= 222
|   |   |   |   CRIM <= 0.06888: medhigh (3.0)
|   |   |   |   CRIM > 0.06888: medlow (4.0)
|   |   |   TAX > 222: medlow (199.0/9.0)
|   RM > 6.54
|   |   RM <= 7.42
|   |   |   DIS <= 1.8773: high (4.0/1.0)
|   |   |   DIS > 1.8773
|   |   |   |   PTRATIO <= 19.2
|   |   |   |   |   RM <= 7.007
|   |   |   |   |   |   LSTAT <= 5.39
|   |   |   |   |   |   |   INDUS <= 6.41: medhigh (25.0/1.0)
|   |   |   |   |   |   |   INDUS > 6.41: medlow (2.0)
|   |   |   |   |   |   LSTAT > 5.39
|   |   |   |   |   |   |   DIS <= 3.9454
|   |   |   |   |   |   |   |   RM <= 6.861
|   |   |   |   |   |   |   |   |   INDUS <= 7.87: medhigh (9.0)
|   |   |   |   |   |   |   |   |   INDUS > 7.87: medlow (3.0/1.0)
|   |   |   |   |   |   |   |   RM > 6.861: medlow (3.0)
|   |   |   |   |   |   |   DIS > 3.9454: medlow (14.0/1.0)
|   |   |   |   |   RM > 7.007: medhigh (29.0)
|   |   |   |   PTRATIO > 19.2: medlow (11.0/1.0)
|   |   RM > 7.42
|   |   |   PTRATIO <= 17.9: high (25.0/1.0)
|   |   |   PTRATIO > 17.9
|   |   |   |   AGE <= 43.7: high (2.0)
|   |   |   |   AGE > 43.7: medhigh (3.0/1.0)
LSTAT > 14.98
|   CRIM <= 0.63796
|   |   INDUS <= 25.65
|   |   |   DIS <= 1.7984: low (5.0/1.0)
|   |   |   DIS > 1.7984: medlow (37.0/2.0)
|   |   INDUS > 25.65: low (4.0)
|   CRIM > 0.63796
|   |   RAD <= 4: low (13.0)
|   |   RAD > 4
|   |   |   NOX <= 0.655
|   |   |   |   AGE <= 97.5
|   |   |   |   |   DIS <= 2.2222: low (8.0)
|   |   |   |   |   DIS > 2.2222: medlow (6.0/1.0)
|   |   |   |   AGE > 97.5: medlow (5.0)
|   |   |   NOX > 0.655
|   |   |   |   CHAS = 0: low (80.0/8.0)
|   |   |   |   CHAS = 1
|   |   |   |   |   DIS <= 1.7455: low (2.0)
|   |   |   |   |   DIS > 1.7455: medlow (2.0)
```

FIG. 4.1. *A decision tree generated by the WEKA's j48 classifier*

4. Treatment Learning Examples and Techniques.

4.1. Example Output. One reason to recommend treatment learning is that its theories are succinct and easy to understand. Figure 4.1 shows the output of a classifier (the j48 tree learner from the WEKA[1] toolkit) as contrasted with the output of a treatment learner (TAR3, as defined in a following section) in Figure 4.2 on housing data from the city of Boston (a nontrivial dataset with over five hundred examples). The tree output by the j48 classifier is detailed, but difficult to understand. A user

---

[1] http://www.cs.waikato.ac.nz/ml/weka/

```
Baseline class distribution:
low:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~      [    133 - 29%]
medlow:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~    [    131 - 29%]
medhigh:~~~~~~~~~~~~~~~~~~~~~           [     97 - 21%]
high:~~~~~~~~~~~~~~~~~~~~~              [     94 - 21%]


 Treatment:[PTRATIO=[12.6..16)
           RM=[6.7..9.78)]


New class distribution:
low:                                   [      0 -  0%]
medlow:                                [      0 -  0%]
medhigh:                               [      1 -  3%]
high:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~     [     38 - 97%]
```

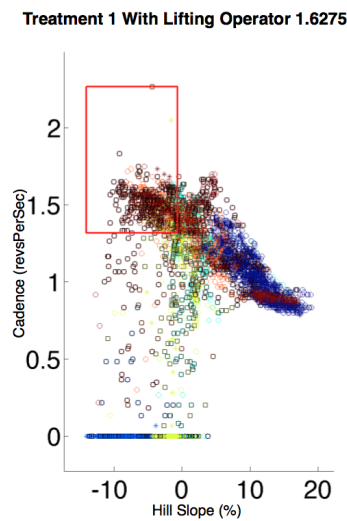FIG. 4.2. *A textual treatment generated by the TAR3 learner.*

FIG. 4.3. *A graphical representation of a treatment generated by the TAR3 learner.*

must follow the branches of the tree in order to derive conclusions. The treatment given by TAR3 is much easier to understand. It only presents three important pieces of information. First, it tells us the baseline class distribution. It then shows the best treatment—the smallest constraint that most changes the class distribution. In this case, the best houses had 6.7 to 9.78 rooms, and the optimal parent-teacher ratio was 12.6 to 16. Finally, it shows the class distribution if that treatment is applied.

These same treatments can be mapped graphically in a way that is even easier to understand. Figure 4.3 shows a two-variable treatment for a dataset that represents the timeseries operation of a bicycle. Each variable or combination of variables within the treatment is given a one or two-dimensional plot. All of the possible values for the noted attributes are plotted, with different shapes to indicate the class. The area inside of the rectangle is the data that the treatment learner is trying to isolate. Lines are plotted around the minimum and maximum values of the ranges given by the treatment. A single glance informs the user that the area contained within these bold lines is the area of interest. For example, the plot in Figure 4.3 shows the values for the attributes "Hill Slope" and "Cadence" The treatment suggests limiting the

values for this variable to the area surrounded by these lines (about -14% to -1% for Hill Slope and 1.3 to 2.2 for Cadence). This output is easy to read and understand, especially when compared to the complex tree output by a classfier (see Figure 4.1).

Cognitive scientists have demonstrated that humans are far more likely to use simple models over more complex ones when making decisions [16]. The simpler the output, the easier it is to implement and the more likely that designers will make use of it. This is a key point. For treatment learning to make an effective difference during the design phase of a project, it must produce output that can be understood in a single look.

4.2. BORE Classification. The raw datasets that we are using have not been classified. Therefore, before they can be used by the treatment learner, they must be sorted into distinct categories. This is done by a process called *BORE*, short for *best* or *rest*. BORE takes instances scored by a penalty function (which is uniquely defined for each dataset) as inputs and categorizes each of them "best" or "rest".

BORE maps the individual factors into a hypercube, which has one dimension for each scored utility. It then normalizes instances scored on the N dimensions into 0 for "worst," and 1 for "best." The corner of the hypercube at 1,1,... is the *apex* of the cube and represents the desired goal for the system. All of 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. For each run i of the simulator, the n outputs X_i are normalized to the range 0...1 as follows:

$$(4.1) \qquad N_i = \frac{X_i - \min(X)}{\max(X) - \min(X)}.$$

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

$$(4.2) \qquad W_i = 1 - \frac{\sqrt{N_1^2 + N_2^2 + ...}}{\sqrt{n}},$$

where higher W_i ($0 \le W_i \le 1$) 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 of the W_i scores were sorted according to a given threshold. Those that fall above this threshold are classified as "best" and the remainder as "rest."

4.3. TAR3. TAR3 (and its predecessor TAR2 [27]) are based on two fundamental concepts—lift and support. The *lift* of a treatment is the change that some decision makes to a set of examples after imposing that decision. TAR3 is given a set of training examples E. Each example $e \in E$ maps a set of attribute ranges $R_i, R_j, ... \rightarrow C$ to some class score. The class symbols C_1, C_2,... are stamped with a rank generated from a utility score $U_1 < U_2 < ... < U_C$. Within E, these classes occur at frequencies F_1, F_2,..., F_C where $\sum F_i = 1$. A treatment T of size M is a conjunction of attribute ranges $R_1 \wedge R_2... \wedge R_M$. Some subset of $e \subseteq E$ is contained within the treatment. In that subset, the classes occur at frequencies f_1, f_2,..., 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, the lift is defined as

$$(4.3) \qquad lift = \frac{\sum_c U_c f_c}{\sum_c U_c F_c}.$$

The classes used for treatment learning get a score U_C and the learner uses this to assess the class frequencies resulting from applying a treatment by finding the subset of the inputs that falls within the reduced treatment space. In normal operation, a treatment learner conducts *controller learning*; that is, it finds a treatment which selects for better classes and rejects worse classes. By reversing the scoring function, treatment learning can also select for the worst classes and reject the better classes. This mode is called *monitor learning* because it locates the one thing we should most watch for.

Real-world datasets, especially those from hardware systems, contain some *noise* —incorrect or misleading data caused by accidents and miscalculations. If these noisy examples are perfectly correlated with failing examples, the treatment may become overfitted. An overfitted model may come with a massive lift score, but it does not accurately reflect the details of the entire dataset. To avoid overfitting, learners need to adopt a threshold and reject all treatments that fall on the wrong side of this threshold. We define this threshold as the *minimum best support*.

Given the desired class, the best support is the ratio of the frequency of that class within the treatment subset to the frequency of that class in the overall dataset. To avoid overfitting, TAR3 rejects all treatments with best support lower than a user-defined minimum (usually 0.2). As a result, the only treatments returned by TAR3 will have both a high *lift* and a high *best support*. This is also the reason that TAR3 prefers smaller treatments. The fewer rules adopted, the more evidence that will exist supporting those rules.

FIG. 4.4. *Probability distribution of individual attribute scores.*

TAR3's lift and support calculations can assess the effectiveness of a treatment, but they are not what generates the treatments themselves. A naive treatment learner might attempt to test all subsets of all ranges of all of the attributes. Because a dataset of size N had 2^N possible subsets, this type of brute force attempt is inefficient. The art of a good treatment learner is in finding good heuristics for generating candidate treatments.

The algorithm begins by discretizing every continuous attribute into smaller ranges by sorting their values and dividing them into a set of equally-sized bins. It then assumes the small-treatment effect; that is, it only builds treatments up to a user-defined size. Past research [18, 19] has shown that a treatment's size should be less than four attributes. TAR3 will then only build treatments from the discretized ranges with a high heuristic value.

TAR3 determines which ranges to use by first determining the lift of each individual attribute. These individual scores are then sorted and converted into a cumulative probability distribution, as seen in Figure 4.4. TAR3 randomly selects values from this distribution, meaning that low-scoring ranges are unlikely to be selected. To build a treatment, n (random from 1...max treatment size) ranges are selected and combined. These treatments are then scored and sorted. If no improvement is seen after a certain number of rounds, TAR3 terminates and returns the top treatments.
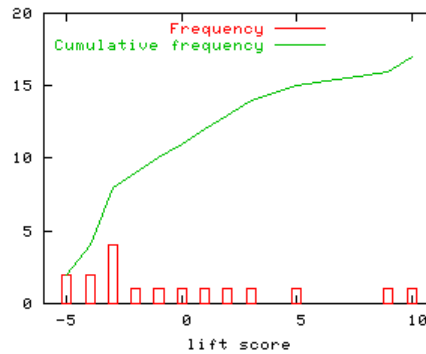
4.4. TAR4.1. TAR3 is not a data miner. Not only does it have a different goal (treatment versus classification), its very design runs counter to that of traditional classifiers such as Naive Bayes [43]. Data miners, according to Bradley et al [9]:

- Require one scan, or less, of the data.
- Are on-line, anytime algorithms.
- Are suspendable, stoppable, and resumable.
- Efficiently and incrementally add new data to existing models.
- Work within the available RAM.

TAR3 stores all examples from the dataset in RAM. It also requires three scans of the data in order to discretize, build theories, and rank the generated treatments. TAR4 was designed and modeled after the SAWTOOTH [30] incremental Naive Bayes classifier in order to improve the runtime, lessen the memory usage, and better scale to large datasets.

Naive Bayes classifiers offer a relationship between fragments of evidence E_i, a prior probability for a class $P(H)$, and a posteriori probability $P(H|E)$ defined by

$$(4.4) \qquad P(H|E) = \prod_i P(E_i|H)\frac{P(H)}{P(E)}.$$

For numeric features, a features mean μ and standard deviation σ are used in a Gaussian probability function [43]:

$$(4.5) \qquad f(x) = 1/(\sqrt{2\pi}\sigma)e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Simple naive Bayes classifiers are called "naive" since they assume independence of each feature. Potentially, this is a significant problem for data sets where the static code measures are highly correlated (e.g. the number of symbols in a module increases linearly with the module's lines of code). However, Domingos and Pazzini have shown theoretically that the independence assumption is a problem in a vanishingly small percent of cases [12].

TAR4.1 still requires two passes through the data, for discretization and for building treatments. These two steps function in exactly the same manner as the corresponding steps in the TAR3 learner. TAR4.1, however, eliminates the final pass by building a scoring cache during the BORE classification stage. As explained previously, examples are placed in a U-dimensional hypercube during classification, with one dimension for each utility. Each example $e \in E$ has a normalized distance $0 \le D_i \le 1$ from the *apex*, the area where the best examples reside. When BORE classifies examples into *best* and *rest*, that normalized distance is added as D_i to the *down* table and $1 - D_i$ to the *up* table.

When treatments are scored by TAR4.1, the algorithm does a linear-time table lookup instead of scanning the entire dataset. Each range $R_j \in example_i$ adds $down_i$ and up_i to frequency counts $F(R_j|base)$ and $F(R_j|apex)$. These frequency counts are then used to compute the following probability and likelihood equations:

$$(4.6) \qquad P(apex) = \frac{\sum_i up_i}{\sum_i up_i + \sum_i down_i},$$

$$(4.7) \qquad P(base) = \frac{\sum_i down_i}{\sum_i up_i + \sum_i down_i},$$

$$(4.8) \qquad P(R_j|apex) = \frac{F(R_j|apex)}{\sum_i up_i},$$

$$(4.9) \qquad P(R_j|base) = \frac{F(R_j|base)}{\sum_i down_i},$$

$$(4.10) \qquad L(apex|R_k \wedge R_l \wedge ...) = \prod_x P(R_x|apex) * P(apex),$$

$$(4.11) \qquad L(base|R_k \wedge R_l \wedge ...) = \prod_x P(R_x|base) * P(base).$$

TAR4.1 finds the *smallest* treatment T that *maximizes*

$$(4.12) \qquad P(apex|T) = \frac{L(apex|T)^2}{L(apex|T) + L(base|T)}.$$

Note the squared term in the top of the equation, $L(apex|T)^2$. This term was not squared in the first design of TAR4. The first version of TAR4 was a failure. Its results were muddled by the dependent attributes. The standard Naive Bayes assumes independence between all attributes and keeps singleton counts. TAR4 added redundant information, which altered the probabilities. In effect, it produced treatments with high scores, but without the *minimum best support* required in TAR3. TAR4.1 was redesigned to prune treatments with low support in the original dataset. By squaring the likelihood that a range appears in the apex, those treatments that lack support are dropped in favor of those that appear often in the apex.

5. Optimization Techniques. Treatment learning is a relatively unexplored field, limiting the number of algorithms that TAR3 and TAR4.1 can be benchmarked against. One way of casting the treatment problem is as an *optimization* [11] problem. The scoring methods used are simply mathematical objective functions. Therefore, it becomes possible to compare the treatment learning tools against standard techniques used to address optimization problems. When presented as an optimization problem, the objective function F for TAR3 looks like

$$(5.1) \qquad \text{maximize } F(x) = \frac{\sum_c U_c f_c(x)}{\sum_c U_c F_c},$$

where U_c, f_c and F_c are defined as for Equation 4.3, and x is the attributes and ranges for a suggested treatment. Similarly, the objective function for TAR4.1 is defined as

$$(5.2) \qquad \text{maximize } F(x) = \frac{L(apex|x)^2}{L(apex|x) + L(base|x)},$$

where the likelihood functions L are the same as those given in Equation 4.12. For both Equation 5.1 and Equation 5.2, x is the treatment—the attributes (discrete values) and their ranges (both continuous and discrete values)—and the size of x will vary based on the number of attributes that the algorithms choose for that particular treatment. Note that since the algorithms used by TAR3 and TAR4.1 do not use gradients there is no requirement for either of the above $F(x)$ to be smooth, and, in practice, both objective functions are highly non-smooth.

5.1. Simulated Annealing. Simulated Annealing is a classic stochastic search algorithm. It was first described in 1953 [28] and refined in 1983 [22].The algorithm's namesake, annealing, is a technique from metallurgy, where a material is heated, then cooled. The heat causes the atoms in the material to wander randomly through different energy states and the cooling process increases the chances of finding a state with a lower energy than the starting position.

The Simulated Annealing algorithm used in this experiment begins by making an initial guess. This guess is a twelve number vector that approximates a four variable treatment. The exact structure of this vector is {ATTRIBUTE,MIN,MAX} repeated four times. The algorithm then tries to solve objective functions corresponding to Equation 5.1 and Equation 5.2, except that, in this case x is limited by the algorithm to the structure of the initial guess. Note that Simulated Annealing only requests that the objective functions be smooth in a limited region near the final solution. We have no such guarantees for our problem, but in practice this is a workable approximation.

The algorithm will continue to operate until a.) the number of tries is exhausted, b.) improvement has not been seen for several rounds, or c.) a certain temperature threshold (a function of the time) is met. The current worth will then be compared to a minimum worth threshold and, if that value is not met, the algorithm will reset. The number of possible resets can be limited, but was not for this experiment.

5.2. Gradient-Based Optimization. The data mining problem posed here requests some subset of the variables and then requests a range for those variables. The best solution to this problem necessarily consists of a combination of integer and (likely nonlinear) continuous values. What's more, the search space is large: there are on the order of hundreds of attributes and on the order of thousands of individual runs. To complicate the problem further, the possible values for each attribute may themselves be continuous or discrete. As a final barrier to traditional optimization techniques, the input variables may not be directly correlated with the output class that is being chosen, either because the appropriate input variables were not selected out of the thousands or (sometimes) tens of thousands of possible input variables or because of some non-determinism in the solution. These factors cause the objective function to be highly non-smooth—there are likely to be many discontinuities and there are no guarantees that the neighborhood of the global solution will be discontinuity-free. Classically, this is the sort of problem that must be solved by direct search optimization methods. However, on a practical basis, optimization techniques derived by assuming that the optimization function is smooth tend to be much more efficient, and often work better-than-expected by utilizing a wide array of numerical "tricks" that work to make the objective function act as if it were more smooth.

The particular optimization technique implemented for this work is a Quasi-Newton method with a BFGS update [38]. This $O(n^2)$ method builds up Hessian information as the iterations proceed, and the approximate Hessian is updated with a rank-one matrix. This constant building of the curvature information tends to avoid the subspace-searching problem that Conjugate Gradient methods can fall into for large problems like those solved within the RSE group. It also avoids the uncertainty that comes with parameter tuning for trust-region methods like Levenberg-Marquardt. However, like all descent methods, the algorithm assumes that the function it is optimizing is essentially continuous.

Mixed integer-nonlinear problems can be solved in several ways [17]. The first such way is a combinatorial approach–solving all possible combinations and choosing the combination that gives the best answer to the objective function. For the types of problems solved in practice at RSE, the combinatorial approach is computationally intractable. One recent example looked for the best 4 attribute treatment out of 128 attributes; the solution of this problem would have required almost 11 million separate optimizations. Even limiting the problem to choose the one best treatment would still require 128 different optimizations. Instead, as suggested by Gill et al [17],

we introduced a new set of variables n_i into the TAR3 objective function where each variable was the percentage likelihood that the attribute i should be included in the treatment. We then introduced the term $\sum_i n_i{}^{2^{-1}}$ into the objective function to further increase the likelihood of a single discrete choice being made by the optimizer. The final form of this objective function looks like

$$(5.3) \qquad \text{minimize } F(x) = \sum_i \frac{1}{n_i{}^2} \frac{-\sum_c U_c f_c(x, \delta_x(n_i))}{\sum_c U_c F_c},$$

where U_c, f_c and F_c are defined as for Equation 4.3, i is the index corresponding to the attributes and δ_x is a threshold function that determines whether a particular attribute has enough of a percentage likelihood of being included in the treatment. The vector x now takes the form $\{n_i, \text{MIN}, \text{MAX}, ...\}$ where all of the components are continuous and is 3 times the number of attributes in size. While it is possible that we could have improved the performance of any gradient descent-based method by recasting the objective function to one that was more smooth or (perhaps) by recasting the problem as a constrained minimization problem, no such solution presented itself after a good-faith effort to discover it. This is a common limitation of gradient descent-based methods.

Another serious limitation of gradient-based optimization methods for these sorts of data mining problems is that the objective function can have cliffs even with respect to the continuous ranges because the input to the objective function, the data, is also inherently discrete. As a result, it is very likely that the optimizer will become stuck in local equilibria and that the final solution will be highly dependent on the initial conditions. The particular optimizer we are using here attempts to resolve this problem by attempting to determine that it has reached a cliff and choosing random search directions to make progress.

6. Experiment. Twelve Monte Carlo Filtering analyses were run for three different datasets, using five different methods: TAR3, TAR4.1, Simulated Annealing with the TAR3 objective function, Simulated Annealing with the TAR4.1 objective function, and a Quasi-Newton BFGS method with a modified version of TAR3's scoring function. Two of the datasets are from actual analyses performed within the RSE group at NASA Ames. The data in these two sets were gathered during Monte Carlo runs using a high-fidelity physics simulation. One of these datasets represents reentry simulations while the other dataset represents launch abort simulations. The first dataset contains 191 runs worth of 52 different attributes. The second dataset contains 1000 runs worth of 249 attributes. The data from these two different examples had complicated penalty functions based on all of the flight requirements—these requirements include metrics like bounds on miss distances, fuel consumption, and the stress on the parachutes. Data with the highest penalty functions are said to have 'failed' and data with the lowest penalty functions are considered to be the 'best' data. Note that, because of the complicated penalty function, the 'failed' data are likely to have exceeded the allowed values on more than one requirement. An analysis like this gives an overall view of the safest flight conditions given all of the different possible individual failures. While the analysis within the RSE group goes on to do the same sort of analysis for each individual failure, that was not the focus of our main experiment.

To demonstrate the broad applicability of the technique, we also ran a Monte Carlo Filtering analysis on some data obtained during a bicycle ride. The software

that generated the data gave a particularly noisy power measurement; the penalty function in this case evaluated each time point as an individual run and penalized each run by the noise in the power measurement. The goal was to see which of the other measured parameters was most likely to correspond with the noisy power measurement. This dataset contained 4435 runs over 11 attributes.

7. Results. During each trial, several statistics were collected in order to assess the treatments output by that algorithm. Let $\{A, B, C, \text{and } D\}$ denote the true negatives, false negatives, false positives, and true positives. From these measures, we can compute certain standard formulas.

$$(7.1) \qquad recall = probability\ of\ detection = \frac{D}{B + D}$$

$$(7.2) \qquad probability\ of\ false\ alarm = \frac{C}{A + C}$$

$$(7.3) \qquad precision = \frac{D}{D + C}$$

For recall and precision, higher values are better. For the probability of false alarm, lower values are desired. Those performance measures, along with the runtime (which should be minimized) were collected for each individual run of each algorithm, then the averages, medians, and standard deviations were saved for each trial. After all ten trials, those statistics were combined and used to create quartile charts. The results for each of our algorithms were combined and ranked using the Mann-Whitney rank-sum test [23].

These quartile charts, sorted by the Mann-Whitney ranks, can be seen in Figure 7.1, Figure 7.2, and Figure 7.3. Note that SA-T3 and SA-T4 refer to the two variants of Simulated Annealing, using the TAR3 and TAR4.1 objective functions respectively. Also note that only the median values from each individual run were used in the combined results. In each quartile chart, the horizontal lines show the 25 to 75 percentile range, and the black dot represents the median point. The ranks come from the Mann-Whitney rank-sum test at a 95% confidence level. Each rank, from one to five, is statistically different and better than the following rank. If two algorithms have the same rank, their results are not statistically different.

After looking at the results from all three datasets, a clear ranking emerges for each of the collected performance statistics. These are (from best to worst, with parentheses denoting a tie):
- Runtimes: TAR4.1, TAR3, QN, (SA-T4, SA-T3)
- Recall: (TAR4.1, QN), SA-T4, TAR3, SA-T3
- Probability of False Alarm: TAR3, SA-T3, TAR4.1, SA-T4, QN
- Precision: TAR3, SA-T3, TAR4.1, (SA-T4, QN)

While these rankings do not show a single "winner," they do present a clear victory for the two treatment learning techniques. Either TAR3 or TAR4.1 is ranked at the top in every category, and neither of them are ranked worst in any category.

The Simulated Annealer acted in accordance with its objective function. When using the TAR3 objective function, it tends towards high precision and low recall. Likewise, when using the TAR4.1 objective function, Simulated Annealing returns treatments with higher recall and low precision. In both cases, it performed more weakly than its treatment learner counterpart. Both the weaker results and slower runtime can be explained by the very design of Simulated Annealing: because it makes

Metric	Project 1			
		Rank	Program	50%
		1	TAR4.1	0.13
Runtime		2	TAR3	0.31
		3	QN	6
		4	SA-T4	15
		4	SA-T3	16

	Rank	Program	50%	Quartiles
	1	TAR4.1	59	
	1	QN	36	
Recall	2	SA-T4	25	
	3	TAR3	22	
	4	SA-T3	20	
				0 50 100

	Rank	Program	50%	Quartiles
	1	TAR3	1	
	2	SA-T3	9	
P(False Alarm)	3	TAR4.1	25	
	4	QN	34	
	4	SA-T4	71	
				0 50 100

	Rank	Program	50%	Quartiles
	1	TAR3	90	
	2	TAR4.1	44	
Precision	2	SA-T3	42	
	3	QN	24	
	3	SA-T4	11	
				0 50 100

FIG. 7.1. *Results on several criterion, sorted by Mann-Whitney rank, for the RSE Project 1. In each quartile chart, the horizontal lines (if any) show the 25 to 75 percentile range, and the black dot represents the median point. Row i is* ranked *higher than row i − 1 iff their their are statistically different (Mann-Whitney 95% confidence level) and the median of row i is better than row i + 1. For recall and precision, higher values are better. For the probability of false alarm, lower values are desired.*

a single initial guess and mutates it, it is unable to try as many combinations as TAR3 or TAR4.1. It must keep trying to make its guess better, and only resets after certain timers expire. It keeps resetting until a certain score threshold is met, which is why it is slower than the other algorithms. If its initial guess is particularly poor, it will never be able to mutate it into something that scores highly. Thus, it will reset until it is able to find a good mutation.

Quasi-Newton performs very well on recall, even tying with TAR4.1 in the rank-sum test. However, it also has the worst probability of selecting false positives. In fact, its false positive rate exceeds its true positive rate on the bicycle dataset. Quasi-Newton is extremely imprecise, it tries to suggest treatments that contain most of the data rather than making any attempt to fit the treatment to the data. As with

Metric	Project 2			
		Rank	Program	50%
Runtime		1	TAR4.1	0.1
		2	TAR3	0.2
		3	QN	3.3
		4	SA-T3	2.5
		5	SA-T4	3.1

Metric	Rank	Program	50%	Quartiles
Recall	1	TAR4.1	48	
	1	QN	44	
	2	SA-T4	27	
	3	TAR3	23	
	4	SA-T3	19	
				0 50 100

Metric	Rank	Program	50%	Quartiles
P(False Alarm)	1	SA-T3	3	
	2	TAR3	6	
	3	SA-T4	23	
	3	TAR4.1	28	
	4	QN	40	
				0 50 100

Metric	Rank	Program	50%	Quartiles
Precision	1	TAR3	58	
	1	SA-T3	53	
	2	TAR4	37	
	3	SA-T4	25	
	3	QN	25	
				0 50 100

FIG. 7.2. *Results on several criterion, sorted by Mann-Whitney rank, for RSE Project 2. In each quartile chart, the horizontal lines (if any) show the 25 to 75 percentile range, and the black dot represents the median point. Row i is* ranked *higher than row i − 1 iff their their are statistically different (Mann-Whitney 95% confidence level) and the median of row i is better than row i + 1. For recall and precision, higher values are better. For the probability of false alarm, lower values are desired.*

Simulated Annealing, Quasi-Newton returns results that are highly dependent on the initial conditions. This is because the data has a tendency to be discrete, while Quasi-Newton assumes continuous conditions. In these situations, the algorithm is likely to become stuck in local minima.

Both Simulated Annealing and Quasi-Newton require favorable initial conditions. This weakness is not shared by the treatment learners because of their highly randomized nature. They do not make any single initial guess, and they do not try to manipulate their findings. The use of stochastic search algorithms has been criticized because their results may not be optimal; they may miss potentially powerful treatments because they randomly skip around the space of possible solutions. However, the problem that we are trying to solve is inherently not smooth (much less not

Metric	Bicycle					
		Rank	Program	50%		
		1	TAR4.1	0.18		
Runtime		2	TAR3	0.23		
		3	QN	8.3		
		4	SA-T4	158.1		
		4	SA-T3	398.5		
	Rank	Program	50%	Quartiles		
Recall	1	TAR4.1	40			
	1	QN	30			
	2	SA-T4	24			
	3	TAR3	21			
	4	SA-T3	19			
				0 50 100		
	Rank	Program	50%	Quartiles		
P(False Alarm)	1	TAR3	6			
	2	SA-T3	11			
	3	SA-T4	22			
	4	TAR4.1	31			
	5	QN	33			
				0 50 100		
	Rank	Program	50%	Quartiles		
Precision	1	TAR3	54			
	2	SA-T3	32			
	3	TAR4	30			
	4	SA-T4	23			
	4	QN	22			
				0 50 100		

FIG. 7.3. *Results on several criterion, sorted by Mann-Whitney rank, for the bicycle dataset. In each quartile chart, the horizontal lines (if any) show the 25 to 75 percentile range, and the black dot represents the median point. Row i is* ranked *higher than row i − 1 iff their their are statistically different (Mann-Whitney 95% confidence level) and the median of row i is better than row i + 1. For recall and precision, higher values are better. For the probability of false alarm, lower values are desired.*

convex), which means that gradient-based optimization techniques are also likely to miss the optimal solution. This effect is somewhat mitigated by TAR3 and TAR4.1 because they form treatments from a cumulative probability distribution that favors high-scoring ranges.

8. Discussion. While the results show the advantage for using treatment learning algorithms for these kinds of problems, they do not answer *which* one to use. TAR3 and TAR4.1 excel in different areas, and there is a notable tradeoff between the two. This makes it difficult to clearly recommend one over the other.

TAR3 is extremely specific in its recommendations. It tends to produce treatments that maximize the *lift* calculation while just meeting the support requirement.

The result of this are treatments with a low recall value and a very high level of precision. While the recall values are weak, reflecting the lower support, the false alarm rate is nonexistent. This alone may be a reason to favor TAR3's treatments. TAR3 will not give you all of the sources of failure, but it will suggest very few false positives.

TAR4.1, on the other hand, is prone to suggesting treatments with a very high level of support, leading to a higher probability of detection. The problem with TAR4.1's results is that its treatments are not well-fitted to the data, they do a poor job of filtering out noisy factors or unnecessary information. This results in a much higher false alarm rate then that seen in TAR3's predictions.

Both the data mining and information retrieval fields have weighed in on the tradeoff between precision and recall on numerous occasions [2,3,10,24,25]. The final recommendation on which treatment learner to use comes down to the priorities of a particular project. It may even be advisable to use both and compare the treatments that emerge, and the average runtime of < 10 seconds for each easily allows for this scenario.

The results for both treatment learners when asked to solve the problems as designed by the RSE group, regardless of their respective strengths, tend to be low when compared to the results of standard data mining problems in the literature. Note that, in most cases, every single algorithm used in this experiment returned performance values below 50%. This effect is largely due to the type of problem being solved within the RSE group. In standard use at RSE, these treatment learners are being asked to look for *any* failure type in addition to *specific* failure types. The experiment run for this paper was trying to solve the significantly harder problem of looking for the causes of *any* failure type. This has a tendency to blur the results because the learners must correlate back to a wide range of inputs, perhaps with disjoint ranges, and there is no guarantee that key inputs for an some individual failure are in the dataset.

To gain a clearer look at their potential performance, both the treatment learning and optimization algorithms were used to look at a specific failure type in isolation for the second RSE project. Those results can be seen in Figure 8.1. Both TAR3 and TAR4.1 are better able to fit their treatments to the specific problem, resulting in a much lower false alarm rate and almost 100% precision. Interestingly, TAR3 and TAR4.1 performed almost identically, with TAR4.1 maintaining a slightly higher recall and TAR3 a slightly higher precision. TAR3's recall rose significantly, from a 23% median to 36%, while TAR4.1's dropped roughly the same amount, from 48% to 39%.

This experiment in looking at a specific failure type is an even clearer example of why treatment learning techniques should be used. While there was an increase in precision across the board due to the more precise nature of the problem, the performance of the optimization methods was far below the treatment learners. When the TAR3 objective function was used by the simulated annealer, it performed similarly to the actual TAR3. However, its results were poorer and its runtime was slower. Simulated annealing with the TAR4.1 objective function and the Quasi-Newton method showed particularly poor results. For both of these algorithms, the false alarm rate was higher than the median detection rate.

9. External Validity. These experiments were conducted at NASA Ames Research Center with assistance from NASA contractors and civil servants. Additionally, two of the primary sources of data were from large-scale NASA project simulations.

Metric	Project 2 (error: angle of attack)			
		Rank	Program	50%
		1	TAR4.1	0.01
Runtime		2	TAR3	0.02
		3	SA-T3	2.6
		3	QN	3.1
		4	SA-T4	3.0

	Rank	Program	50%	Quartiles
	1	TAR4.1	39	
	2	QN	34	
Recall	3	TAR3	36	
	3	SA-T4	24	
	4	SA-T3	21	
				0 50 100

	Rank	Program	50%	Quartiles
	1	TAR3	0	
	1	SA-T3	0	
P(False Alarm)	2	TAR4.1	7	
	3	SA-T4	30	
	4	QN	43	
				0 50 100

	Rank	Program	50%	Quartiles
	1	TAR3	100	
	2	TAR4.1	97	
Precision	2	SA-T3	97	
	3	SA-T4	81	
	3	QN	82	
				0 50 100

FIG. 8.1. *Results on several criterion, sorted by Mann-Whitney rank, for a specific error type in the second RSE project. In each quartile chart, the horizontal lines (if any) show the 25 to 75 percentile range, and the black dot represents the median point. Row i is ranked higher than row $i-1$ iff their their are statistically different (Mann-Whitney 95% confidence level) and the median of row i is better than row $i+1$. For recall and precision, higher values are better. For the probability of false alarm, lower values are desired.*

Therefore, a possible threat to validity exists from the data and environment used for this experiment. The external validity of NASA-based research has been debated by other authors [26, 41]. Basili et al. [5] have argued that conclusions derived from NASA data are relevant to the overall software industry because NASA contractors are obliged to demonstrate an understanding and adherence to modern industrial best practices. These same contractors service numerous industries. For example, Rockwell-Collins builds systems for both defense contractors and civilian aerospace corporations.

However, the work of other authors is not enough to completely dispel the issue

of external validity. It was for this reason that the bicycle dataset was included in this experiment. The data, recorded during the operation of a bicycle, was not collected using NASA hardware or at a NASA facility. Despite this separation, the same trends occurred and each of the algorithms performed at a similar efficiency. This replication of trends shows that treatment learning is not a task that has been tuned to NASA data; it, in fact, has applications for both large-scale and small-scale industrial testing.

10. Conclusions and Future Work. Building a large-scale industrial system is a difficult task. It can cost millions of dollars and require months to years of testing. Early simulation makes a large difference, cutting both the cost and time to market [36]. However, this early simulation does nothing if there is not a way to tell which specific factors led to system failures. Experts are expensive and their time is limited, they cannot waste hours sorting through gigabytes of simulation logs. They need a way to limit the number of possible combinations that they are looking at.

Treatment learning, formally a subset of minimal contrast-set learning, is one method of accomplishing this task. A treatment learner gathers evidence from labeled simulation instances and determines the smallest rule that, when imposed, makes it most likely that a specific outcome will occur.

This research benchmarked two different treatment learning techniques against two standard optimization algorithms—a Quasi-Newton BFGS method and Simulated Annealing. Three sets of data were used, two from large NASA projects and one from the operation of a bicycle. Each algorithm was used multiple times on each dataset and performance statistics were collected. The results show that treatment learning shows better performance when compared to standard optimization algorithms for these sorts of problems.

The immediate research direction for the treatment learners will center around improvements to the internal heuristics. One idea proposed has been to change the discretization technique. Currently, both TAR3 and TAR4.1 use a simple equal-bin scheme. This naive approach is likely to miss important curves in the data space. Experiments are being conducted with various alternative schemes, including recursive cliff-based methods [14]. Other planned improvements center around optimization of the source code. There are still numerous memory issues that should be addressed and the code should be re-engineered to follow the highest industrial programming standards.

The current research directions for the overall Margins Analysis project with the RSE group are aimed at increasing the number of tested states without losing the wide applicability of the technique. The next step for the tool is to include unit-level symbolic execution tools to aid in test case generation. The proof-of-concept for this hybrid technique found a serious bug in a flight software component [31]. If the performance of this technique with the Margins Analysis is comparable to the performance in the proof-of-concept, we should be able to guarantee much greater percentage of execution of the code while still completing in a reasonable time. Specific avenues of approach for this analysis include using the symbolic execution to weed out duplicate test vectors or to create the initial seeds for the statistical analysis; using the symbolic analysis as extra information to improve the machine learning steps; instrumenting parts of the code to develop the symbolic analysis during statistical simulation; and using the symbolic analysis of some component of the code to inform code coverage of the entire system during statistical analysis.

One abstraction that the Margins Analysis is currently making is that it ignores the time-dependency of data in the post-processing step. The analysis will consider

extreme or mean values over some period of time, it can determine if a particular mode was ever entered into, or it will look at all the data at some key event. A goal for this analysis is to find a way to use sequential data within the machine learning techniques in order to automatically identify more kinds of interesting behavior. This is likely to involve adding a Markov Model or Linear Dynamical System capability into the clustering step [7].

Another potential set of improvements to the Margins Analysis come from incorporating techniques to simulate rare events. There is an obvious relationship between uncovering potential errors at long run times and finding the behavior in the system in rare events. Much of the current rare-event simulation is being done to maintain reliability in petascale computing. Current simulation-based testing research includes optimal splitting [37] and a hybrid method developed by the University of Illinois and IBM that simulates discrete events on different timescales by switching numerical methods [33]. A variation on either of these techniques may be integrable with our current techniques and create a greater guarantee of reliability of safety-critical software.

REFERENCES

[1] A. Acevedo, J. Arnold, W. Othon, and J. Berndt, *Antares: Spacecraft simulation for multiple user communities and facilities*, in AIAA Modeling and Simulation Technologies Conference and Exhibit, 2007, pp. AIAA 2007–6888.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, *Recovering traceability links between code and documentation*, IEEE Transactions on Software Engineering, 28 (2002), pp. 970–983.

[3] G. Antoniol and Y. Gueheneuc, *Feature identification: A novel approach and a case study*, in ICSM 2005, 2005, pp. 357–366.

[4] P. Austin, P. Grootendorst, and G. Anderson, *A comparison of the ability of different propensity score models to balance measured variables between treated and untreated subjects: a monte carlo study*, Statistics in Medicine, 26 (2007), pp. 734–753.

[5] V. Basili, F. McGarry, R. Pajerski, and M. Zelkowitz, *Lessons learned from 25 years of process improvement: The rise and fall of the NASA software engineering laboratory*, in Proceedings of the 24th International Conference on Software Engineering (ICSE) 2002, Orlando, Florida, 2002. Available from `http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/83.88.pdf`.

[6] S.B. Bay and M.J. Pazzani, *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`.

[7] C. Bishop, *Pattern Recognition and Machine Learning*, Springer New York, 2007.

[8] B. Boehm and P. Papaccio, *Understanding and controlling software costs*, IEEE Trans. on Software Engineering, 14 (1988), pp. 1462–1477.

[9] P. Bradley, U. Fayyad, and C. Reina, *Scaling clustering algorithms to large databases*, in Knowledge Discovery and Data Mining, 1998, pp. 9–15. Available from `http://citeseer.ist.psu.edu/bradley98scaling.html`.

[10] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, *The detection and classification of non-functional requirements with application to early aspects*, in RE 2006, 2006, pp. 36–45.

[11] R. Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.

[12] P. Domingos and M. Pazzani, *On the optimality of the simple bayesian classifier under zero-one loss*, Machine Learning, 29 (1997), pp. 103–130.

[13] V. Eruhimov, V. Martyanov, and E. Tuv, *Constructing High Dimensional Feature Space for Time Series Classification*, Springer Berlin / Heidelberg, 2007, ch. Knowledge Discovery in Databases: PKDD 2007, pp. 414–421.

[14] U M Fayyad and I H Irani, *Multi-interval discretization of continuous-valued attributes for classification learning*, in Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1993, pp. 1022–1027.

[15] B. Fischer and J. Schumann, *Autobayes: a system for generating data analysis programs from statistical models*, Journal of Functional Programming, 13 (2003), pp. 483–508.

[16] G. Gigerenzer and D.G. Goldstein, *Reasoning the fast and frugal way: Models of bounded rationality*, Psychological Review, (1996), pp. 650–669.

[17] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, 1981.

[18] K. Gundy-Burlet, J. Schumann, T. Barrett, and T. Menzies, *Parametric analysis of antares re-entry guidance algorithms using advanced test generation and data analysis*, in Submitted to the 9th International Symposium on Artifical Intelligence, Robotics and Automation in Space, 2007.

[19] ———, *Parametric analysis of a hover test vehicle using advanced test generation and data analysis*, in AIAA Aerospace, 2009.

[20] Y. Hu, *Treatment learning: Implementation and application*, master's thesis, Department of Electrical Engineering, University of British Columbia, 2003.

[21] H. Jing, R. George, and E. Tuv, *Informatics in Control, Automation and Robotics II*, Springer Berlin / Heidelberg, 2007, ch. Contributors to a Signal from an Artificial Contrast, pp. 71–78.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science, Number 4598, 13 May 1983, 220, 4598 (1983), pp. 671–680.

[23] H. B. Mann and D. R. Whitney, *On a test of whether one of two random variables is stochastically larger than the other*, Ann. Math. Statist., 18 (1947), pp. 50–60. Available online at `http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&hand%le=euclid.aoms/1177730491`.

[24] A. Marcus and J. Maletic, *Recovering documentation-to-source code traceability links using latent semantic indexing*, in Proceedings of the Twenty-Fifth International Conference on Software Engineering, 2003.

[25] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, *Problems with precision*, IEEE Transactions on Software Engineering, (2007). `http://menzies.us/pdf/07precision.pdf`.

[26] T. Menzies, J. Greenwald, and A. Frank, *Data mining static code attributes to learn defect predictors*, IEEE Transactions on Software Engineering, (2007). Available from `http://menzies.us/pdf/06learnPredict.pdf`.

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

[28] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, *Equation of state calculations by fast computing machines*, J. Chem. Phys, 21 (1953), pp. 1087–1092.

[29] J. Oakley and A. O'Hagan, *Probabalistic sensitivity analysis of complex models: a bayesian approach*, Journal of the Royal Statistical Society B, 66 (2004), pp. 751–769.

[30] A.S. Orrego, *Sawtooth: Learning from huge amounts of data*, master's thesis, Computer Science, West Virginia University, 2004.

[31] C. Pasareanu, P. Mehlitz, D. 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 In ISSTA '08, 2008.

[32] K. Rose, E. Smith, R. Gardner, A. Brenkert, and S. Bartell, *Parameter sensitivities, monte carlo filtering, and model forecasting under uncertainty*, Journal of Forecasting, 10 (1991), pp. 117–133.

[33] E. Rozier, W. Belluomini, V. Deenadhayalan, J. Hafner, K. Rao, and P. Zhou, *Evaluating the impact of undetected disk errors in raid systems*, in Proceedings of the International Conference on Dependable Systems and Networks (DSN), 2009.

[34] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global Sensitivity Analysis: The Primer*, Wiley, 2008.

[35] J. Schumann, K. Gundy-Burlet, C. Pasareanu, T. Menzies, and T. Barrett, *Tool support for parametric analysis of large software systems*, in Proc. Automated Software Engineering, 23rd IEEE/ACM International Conference, 2008.

[36] S. Sendall and W. Kozacaynski, *Model transformation: The heart and soul of model-driven software development*, IEEE Software, 20 (2003), pp. 42–45.

[37] J. Shortle and C. Shen, *A preliminary study of optimal splitting for rare-event simulation*,

in Proceedings of the 2008 Winter Simulation Conference, 2008.

[38] C. Sims, *Matlab optimization software.* QM&RBC Codes, Quantitative Macroeconomics & Real Business Cycles, Mar. 1999.

[39] R. Spear, T. Grieb, and N. Shang, *Parameter uncertainty and interaction in complex environmental models*, Water Resources Research, 30(11) (1994), pp. 3159–3169.

[40] K. Torkkola and E. Tuv, *Feature Extraction*, Springer Berlin / Heidelberg, 2006, ch. Ensembles of Regularized Least Squares Classifiers for High-Dimensional Problems, pp. 297–313.

[41] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, *On the relative value of cross-company and within-company data for defect prediction*, Empirical Software Engineering, (2009). Available from `http://menzies.us/pdf/08ccwc.pdf`.

[42] E. Tuv, A. Borisov, and K. Torkkola, *Intelligent Data Engineering and Automated Learning – IDEAL 2006*, Springer Berlin / Heidelberg, 2006, ch. Best Subset Feature Selection for Massive Mixed-Type Problems, pp. 1048–1056.

[43] I.H. Witten and E. Frank, *Data mining: Practical Machine Learning Tools and Techniques 2nd edition*, Morgan Kaufmann, 2005.