Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

APPLYING ENHANCED FAULT LOCALIZATION TECHNOLOGY TO MONTE CARLO SIMULATIONS

David Kamensky
Ross Gore
Paul F. Reynolds Jr.

University of Virginia
151 Engineer's Way
P.O. Box 400740
Charlottesville, VA, 22901 USA

ABSTRACT

This paper describes and explores applications of several new methods for explaining unexpected behavior in Monte Carlo simulations: (1) the use of fuzzy logic to represent the extent to which a program behaves as expected, (2) the analysis of variable value density distributions, and (3) the geometric treatment of predicate lists as vectors when comparing simulation runs with normal and unexpected outputs. These methods build on previous attempts to localize faults in computer programs. They address weaknesses of existing techniques in cases where programs contain real-valued random variables. The new methods were able to locate a source of error in a Monte Carlo simulation and find faults in benchmarks used by the fault localization community.

1 INTRODUCTION

Simulations and computational models have become a common tool for subject matter experts (SMEs) in a variety of disciplines. Predictions based on simulation outcomes have entered the mainstream of critical public policy and research decision-making practices, often affecting millions of people and billions of dollars. Unfortunately SMEs can struggle for decades with the resolution of unexpected simulation outcomes. Their methods are generally manual and do not scale. Ideally, SMEs would have a tool that accepts the simulation's source code, a series of inputs, and a specification of the expected behavior for each input. The tool would return a list of the program statements most relevant to the unexpected behavior.

Prior work in automatically localizing sources of unexpected outcomes in software, not necessarily simulations, has focused on fault localization. Fault localization is the process of narrowing or guiding the search through source code to help an SME find statements containing faults that cause software failures. Because we are building on work from the fault localization community, we will adopt their vocabulary and use the term *fault* to stand for any source of unexpected simulation behavior, including conceptual errors, logical errors or correct statements that simply lead to unexpected but correct outcomes.

Existing fault localization techniques identify failures by noting outputs that deviate from correct behavior. They identify suspicious source code by associating program statements (Jones and Harrold 2005), variable states (Huang et al. 2007), or predicates (Liblit 2004) with these failures. They often depend on repeatable behavior (Cleve and Zeller 2005, Jeffrey et al. 2008, Jeffrey et al. 2009), assuming that programs are deterministic. Further, they tend to focus on integral rather than floating-point data types (Nainar et al. 2010, Liblit 2004, Arumuga Nainar et al. 2007, Liblit et al. 2003, Liblit et al. 2005, Zhang et al. 2006b). Programs with real-valued random variables are common in simulation, but may cause existing techniques to fail.

Simulationists rarely know exactly what behavior is correct; if they did, simulations would be unnecessary. Even if a modeler can contrive cases with analytical solutions, the outputs of Monte Carlo methods include some random amount of noise. Even given an oracle for a problem, a fault localization method that runs programs repeatedly looking for failures might conclude that a correct Monte Carlo program fails every time, due to small differences between its output and the oracle's. In Section 3 we introduce a fuzzy pass/fail distinction that addresses this issue.

In addition to perturbing the output of a simulation, random variables add noise to the intermediate variable states that fault localization techniques may record. If samples of a random variable rarely repeat, no one variable state will appear suspicious as each will be associated with only one or a handful of failures. In Section 4, we discuss how constructing density distributions of variable states may avoid this issue and help identify suspicious trends.

Section 5 demonstrates the effectiveness of these techniques by applying them to a faulty Monte Carlo simulation of the Ising model. Section 6 generalizes state density distributions to vectors of predicates and evaluates their performance on a traditional fault localization benchmark. Section 7 suggests how this performance might be improved, outlining some possible future studies.

2 RELATED WORK

In this section, we present a summary of existing fault localization techniques, especially those that we are extending.

2.1 Statistical Debugging

Our approach to identifying the causes of unexpected behaviors in simulations is a statistical debugging technique. Statistical debugging techniques use data collected during executions of passing and failing tests to estimate the likelihood that statements contain faults. The TARANTULA technique (Jones and Harrold 2005) counts the passing and failing tests that execute each statement and identifies suspicious statements from this data. Other techniques, such as Cooperative Bug Isolation (CBI) (Liblit 2004), take a predicate-based approach to statistical debugging. CBI uses a set of conditional propositions, or predicates, tested at particular program points. The predicates are given an importance score based on how frequently they are true for specified inputs. A single predicate partitions the space of all inputs into two subspaces: those satisfying the predicate and those not. Selection and treatment of predicates distinguishes a number of other statistical debugging methods, such as HOLMES (Chilimbi et al. 2009), Adaptive Bug Isolation (Nainar et al. 2010), and SOBER (Liu et al. 2005).

2.2 Other Methods

Some fault localization techniques examine limited *slices* of the source code. A slice is a subset of the program statements. Various heuristics can identify slices of statements that are likely to contain faults. Static slicing gathers all statements that could possibly contribute to failing outputs (Tip 1994, Ayewah et al. 2008). Dynamic slicing picks out the statements that are observed to influence failing outputs in test executions (Agrawal and Horgan 1990, Korel and Laski 1988, Zhang et al. 2006b).

Other approaches attempt to induce correct behavior in faulty programs by either automatically repairing the source code (He and Gupta 2004, Weimer et al. 2010) or changing variable states during program execution (Zhang, Gupta, and Gupta 2006a). These techniques are difficult to apply, however, to simulations with random variables and no clear specification of correct output. Automatic repair requires either formal specifications or a training set of tests encoding correct behavior. Deciding what variable states to change during execution is difficult when variable states are randomized and specific tests are not guaranteed to pass or fail.

3 FUZZY PASSING AND FAILING

When the expected behavior of a simulation includes some random variance or is not precisely known, there may be no clear way to partition outputs into passing and failing sets. We suggest using fuzzy passing and failing sets that allow outputs to have an extent of membership in each. This has a theoretical advantage over binary passing and failing, discussed in 3.2. We will apply this method to a faulty simulation in Section 5.

3.1 Formal Definition

We assume that the output of a program is (or can easily be transformed to) some ordered set of real numbers. This is typical of physical simulations. We calculate the passing extent, u, of some output list, X, as follows:

$$u = \left(\frac{\sum_{i=1}^{|X|} W_i f_i(X_i)}{\sum_{w \in W} w} \right) \qquad \text{where} \quad f_i \colon \mathbb{R} \to [0,1] \quad \text{and} \quad |W| = |X| \quad .$$

W is an ordered set of weights to focus attention on particular parts of the output. The functions f_i encode information about the expected behavior and the tolerance for deviation from that behavior. The failing extent is $1-u$. A simple example to consider is the case where the program outputs a single real number, x, that is expected to be $\bar{x}$. An appropriate f might be a bell curve centered on $\bar{x}$: $\exp(-(x-\bar{x})^2/a^2)$, where a is some parameter providing tolerance for deviation.

This method is a generalization of ordinary pass/fail detection (Jones and Harrold 2005, Liblit 2004). If program outputs are mapped to single real numbers, correct outputs form the set X_c, and $f(x) = \delta_x(X_c)$, then the method will reproduce the behavior of existing techniques, partitioning program runs into a passing set with $u = 1$ and a failing set with $u = 0$.

3.2 Theoretical Advantage

We have shown how the fuzzy pass/fail scheme allows replication of ordinary pass/fail detection, performing at least as well as existing methods for any application. Here we demonstrate how fuzzy passing can improve on existing methods.

Consider a program in which execution may or may not encounter a fault. Given an output, x, there is probability $P[B|x]$ that x resulted from a faulty execution and probability $1-P[B|x]$ that only correct behavior contributed to x. Now consider some predicate, p, indicating something about a program state during the execution that produced x. A quantity we might be interested in is the probability that p was true of a faulty execution: $P[B|p]$. This figures prominently in the predicate importance score used by Liblit (2004). With ordinary passing and failing, our best guess at this probability is

$$P[B|p] \approx \frac{F(p)}{F(p)+S(p)} \tag{1}$$

where $F(p)$ is the number of failed cases where p was true and $S(p)$ the number of passed cases. In the fuzzy formalism, we substitute the sum of fail extents for $F(p)$ and the sum of pass extents for $S(p)$. The denominator of (1) is still the number of runs where p was true and $P[B|p]$ becomes the average fail extent of the test cases.

If the fail extent of every output x equals $P[B|x]$ and the representation of output x in the results of test cases is considered to be its probability, the estimate of $P[B|p]$ becomes exact. Choosing a good function for the fail extent of x is a nontrivial problem, but the fuzzy formalism *allows* a choice that can outperform ordinary passing and failing when $P[B|x] \notin \{0,1\}$. While fuzzy passing doesn't guarantee improvements, it makes them possible with no significant extra computation.

4 VARIABLE STATE DENSITIES

Here we assume that the states taken on by a real-valued variable at a particular point in a program are randomly distributed according to some probability density function. A finite test suite will produce many sample values of this variable. Each such sample will have occurred in a passing or failing test case (or, using the method of Section 3, the sample will be associated with a pass extent). We call these variable state samples *varticles* (variable particles). We call the associated appearance of the variable in the program code a *varsite* (variable site). A varticle has a location (the value of the variable when sampled), a passing mass (the pass extent of the associated test), and a failing mass (one minus the passing mass).

Given many varticles, we're interested in determining the density of passing or failing variable states in different locations. Physicists have solved a similar problem when representing fluids with discrete samples in a simulation technique known as Smoothed Particle Hydrodynamics (SPH) (Gingold and Monaghan 1977). In Section 4.1, we apply concepts from SPH to determine local densities of variable states.

Given density distributions of passing and failing states for each varsite, we would like to determine which varsites are suspicious. In Section 4.2 we suggest a method of comparing passing and failing distributions of states to produce suspiciousness scores.

4.1 Construction of Densities

To determine how densely varticles cluster around different locations, we mimic the density calculation used in SPH (Gingold and Monaghan 1977). SPH uses a *smoothing kernel*, W, with compact support, to spread out the mass of a single particle. An operation similar to convolution estimates the density, ρ, at a point, $\mathbf{r}$, by summing over the masses, m_i, of surrounding particles, weighted by the smoothing kernel evaluated at the distance of each particle from $\mathbf{r}$:

$$\rho(\mathbf{r}) = \sum_i W(|\mathbf{r} - \mathbf{r}_i|) m_i$$

Our method replaces the particles with varticles. Recall that a varticle's mass is equal to the pass or fail extent of its associated program execution. For each varsite, our method creates two density distributions: a passed distribution, ρ_p, and a failed distribution ρ_f. These distributions are functions of the varsite's variable's value.

Typical functions used for W are Gaussian bell curves or similarly shaped polynomial splines. In practice, Ws typically have some range beyond which they are zero. A *smoothing length*, h, determines this range. This length may vary by location, becoming very short in regions where particles are close together and very long in regions where particles are far apart. One heuristic for determining h is to ensure that some constant number of particles are always within a sphere of radius $2h$ centered at $\mathbf{r}$ (Faber, Lombardi, and Rasio 2003). In a one-dimensional universe, such as the one that varticles of one varsite inhabit, this heuristic is straightforward to enforce if the list of varticles is sorted by location.

To have predictable running times, we propose evaluating density at only the locations of varticles and assuming linear interpolation between them. Enforcing the heuristic that n varticles are always within range of the smoothing kernel, the total time required to compute the density distribution of N varticles is $O(N \log N + nN)$.

4.2 Comparison of Density Distributions

The pass and fail density distributions, ρ_p and ρ_f, for a varsite will be positive, square-integrable functions. There exists, then, a natural way to rate their similarity on a scale from zero to one: the inner product.

$$\langle \psi(x), \phi(x) \rangle \equiv \int_{-\infty}^{\infty} \psi(x)\phi(x)\,dx$$

The inner product as stated above is not guaranteed to be in the unit interval, but, if the two functions, ψ and ϕ, are normalized such that $\langle \psi, \psi \rangle = \langle \phi, \phi \rangle = 1$, the Cauchy-Schwartz inequality enforces $|\langle \psi, \phi \rangle| \leq 1$. We can therefore rate the suspiciousness, S, of a particular varsite on a scale from zero to one as follows:

$$S = 1 - \left(\frac{\langle \rho_p, \rho_f \rangle}{\sqrt{\langle \rho_p, \rho_p \rangle \langle \rho_f, \rho_f \rangle}} \right)$$

The intuition behind this equation is that varsites with similar distributions of states in passing and failing tests are probably not related to faults, while varsites with different distributions are. A similar intuition motivates the SOBER method of ranking predicates based on differences in their evaluation patterns in passing and failing tests (Liu et al. 2005).

In Section 4.1, we suggested that the density functions should be evaluated at the locations of the varticles only, with linear interpolation between. For such functions, the integrals of products are straightforward to compute. They are simple if the discontinuities in the piecewise linear functions' derivatives are in the same places. This is true for ρ_p and ρ_f because the varticle locations are the same (only the masses are different). S is therefore a sum over definite integrals of products of linear functions. Such integrals have analytical solutions from elementary calculus.

5 A CASE STUDY

Here we present the results of analyzing state density distributions of varsites in a Monte Carlo simulation of the thoroughly studied (Brush 1967) 2-dimensional Ising model. Our implementation represents the spin lattice as a 2-dimensional array of integers, each valued at positive or negative one. The program makes randomly proposed changes to these spins, using a Metropolis algorithm to enforce the Boltzmann distribution (Metropolis et al. 1953). We assumed no external magnetic field. Pseudo-code for the simulation is given below:

```
ISING($w, h, t, N, \beta$)
 1  $L \leftarrow$ (2-D array of $w \times h$ random spins)
 2  $i \leftarrow 0$
 3  $energy \leftarrow 0$
 4  $magnetization \leftarrow 0$
 5  while $i < N$
 6        do
 7            for (each element in $L$)    (a sweep)
 8                do
 9                    $x \leftarrow$ (random integer $\in \{0, \dots w-1\}$)
10                    $y \leftarrow$ (random integer $\in \{0, \dots h-1\}$)
11                    $\Delta E \leftarrow$ (change in energy from flipping $L_{x,y}$)
12                    $r \leftarrow$ (random real $\in [0,1]$)
13                    if $(r < e^{-\beta \Delta E})$, flip $L_{x,y}$
14                    if $i \geq t$    (the first t sweeps are for thermalization)
15                        do
16                            for $(s \in L)$, $energy \leftarrow energy +$ (energy of $s$)
17                            $currentMagnetization \leftarrow \sum_{j,k} L_{j,k}$
18                            $magnetization \leftarrow magnetization + |currentMagnetization|$
19            $i \leftarrow i + 1$
20  $energy \leftarrow energy / ((N-t) w^2 h^2)$
21  $magnetization \leftarrow magnetization / ((N-t) w^2 h^2)$
```

5.1 Experiment

We took the output of the simulation to be the final value of *magnetization* – the absolute value of the lattice's total spin per unit volume – averaged over configurations generated by the Metropolis algorithm. This should approximate the analytical solution, given by Onsager (1944). For some temperature ranges, the lattice will magnetize spontaneously in a randomly-chosen direction. We injected the following fault, based on a mistake in our initial implementation: the absolute value of each sweep's magnetization was not taken while summing magnetizations to compute the average (line 18 of the pseudo-code). Over many sweeps, one would expect the faulty behavior to always result in zero magnetization (the algorithm would thoroughly sample states magnetized in positive and negative directions). Over fewer sweeps, though, the magnetization will often be close to plus or minus the analytical solution. The reason for this is that both the positive and negative magnetizations are surrounded by relatively probable configurations. If the random walk of the Metropolis algorithm wanders toward one direction of magnetization, it has a low probability of moving to the other within a small number of moves. The result is an Ising model that produces correct behavior at times and faulty behavior at others. Next, we apply variable state densities and fuzzy pass extents to this case study to isolate the unexpected behavior.

We first instrumented the Ising model to write variable states to a log file during execution. We then ran the instrumented simulation several times and processed the log file into varsites and varticles. For each varsite, we generated ρ_p and ρ_f and computed S. We used a tent function for the smoothing kernel, W. This violates the condition from SPH that W be differentiable, but we are not computing the gradients of any quantities and thus never need the derivative of W. To determine the masses of varticles, we used a Gaussian fuzzy pass extent function, centered on the analytical solution for magnetization.

5.2 Results

The top few most suspicious varsites were related to the final calculation and output of the magnetization. They computed and wrote faulty outputs in failed cases and correct outputs in passing cases. They were, predictably, flagged as suspicious for this behavior. A programmer would likely realize this and ignore those statements. The next most suspicious varsite was the faulty varsite. The variable *currentMagnetization* (in line 18 of the pseudo-code) should be enclosed in an absolute value function, as shown. It's normalized passing and failing densities are shown in Figure 1. These two density functions are clearly different, an observation quantitatively captured by their low inner product (and thus high suspiciousness). Compare with the similar distributions in Figure 2 of a varsite (*energy* in line 16 of the pseudo-code) that was unrelated to the faulty behavior and not flagged as suspicious.

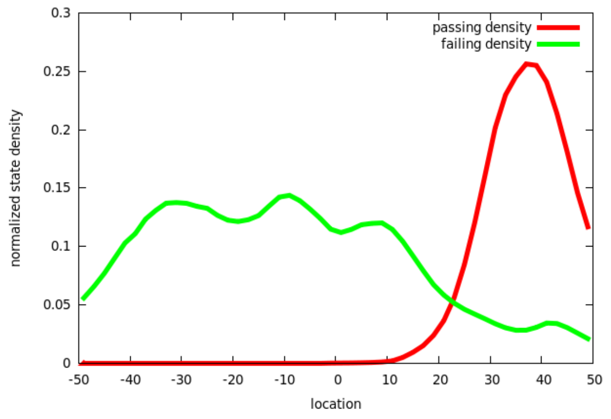

Figure 1: The normalized (see Section 4.2) passing and failing densities for the faulty varsite.

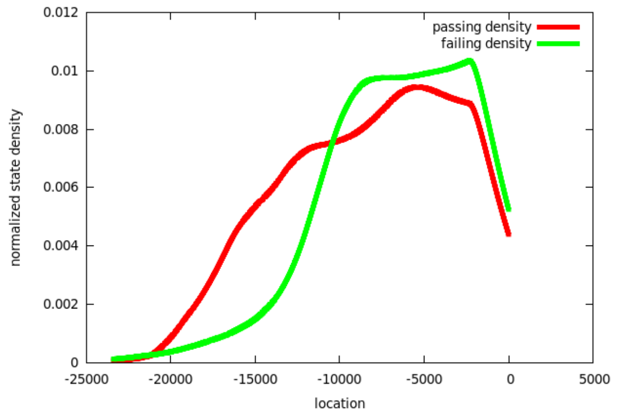

Figure 2: The normalized passing and failing densities for a varsite unrelated to the fault.

This example illustrates how the similarities and differences between ρ_f and ρ_p for different varsites can indicate what varsites are suspicious. In Section 6.2 we further support this intuition by testing a discretization of this approach on a traditional fault localization benchmark. The graphs of ρ_p and ρ_f may also be helpful debugging information, as they point toward the nature of the error. Noting that the accumulation of *currentMagnetization* is associated with unexpected behaviors provides a lead, but the exact error becomes obvious when the density distribution shows that negative values are causing failures and positive values are passing.

6 PREDICATE VECTORS

In this section, we work with arbitrary vector spaces in place of the Hilbert space of density functions. This is a generalization that allows us to test the use of inner products of varticle distributions in cases where the assumption of a continuous density distribution is invalid. Note that the density at some point x is essentially the number of states with location y satisfying the predicate $y \in [x, x+dx]$. The predicate vector approach replaces these infinitesimal bins with some set of arbitrary predicates. A given ordered set of predicates has a passing vector and a failing vector, describing the distribution of passing and failing variable states across the predicates. Defining an inner product for these vectors gives a new method of assigning suspiciousness to the collection of predicates. We define this notion precisely in Section 6.1, then apply a specific formulation of it to benchmarks in Section 6.2.

6.1 Formal Definition

For any ordered set of predicates, there exist two predicate vectors of the same dimensionality: a passing and a failing vector. Component i of the passing vector is the number of variable states occurring in passing tests that satisfy predicate i. The failing vector is defined analogously, but components tally states occurring in failing tests.

The suspiciousness of a collection of predicates is one minus the inner product of its normalized passing and failing vectors. All components of the passing and failing vectors are non-negative, so this product is guaranteed to be in $[0,1]$. Using the dot product, $u_i v_i$ (summation implied), as an inner product is one option. A slightly more general option is to weight predicates with a matrix, A_{ij}, of real, positive values such that the inner product is $u_i A_{ij} v_j$.

6.2 Testing With the Siemens Benchmark Suite

In this section, we use the Siemens Benchmark Suite to test the fault-finding power of the inner product. We compare its rankings of statements with those produced by a variant of Liblit's importance score (Liblit 2004):

$$\text{Importance}(p) = \frac{2}{\frac{\log(NumF)}{\log(F(p))} + \left(\frac{F(p)}{F(p)+S(p)} - \frac{F(p_{obs})}{F(p_{obs})+S(p_{obs})}\right)^{-1}} \tag{2}$$

We modified Liblit's formula slightly to use the data in the predicate vectors. In our tests, $F(p)$ and $S(p)$ counted all true evaluations of p in failed and successful runs rather than just the numbers of failed and successful runs in which p was ever true. To ensure that the score remained in $[0,1]$, we set $NumF$ (originally the number of failed runs) to the largest number of times any predicate occurred in failing test cases. To get the importance score for a varsite, we took the maximum importance score of any predicate associated with the varsite.

6.2.1 TCAS

First, we tested our method on the benchmark suite's faulty versions of the Traffic Collision Avoidance System (TCAS) application. For each version of TCAS, we defined a set of predicates (and thus a passing and a failing predicate vector) for each varsite. The predicate list for a given varsite contained the *elastic predicates* suggested by Gore et al. (2011). We set the elements of the weight matrix, A_{ij}, to δ_{ij}. We display the results of the method following the conventions of Jones and Harrold (2005); we show the percent of faults revealed as a function of the percent of code that must be examined. We consider a fault to be revealed when a varsite collinear to the fault is encountered in a ranked list of varsites. We define the "percent of code that must be examined" as the fraction of the varsite list traversed to reveal the fault. We removed the versions of TCAS with faults in declarations of constants because such faulty statements do not vary in behavior between passing and failing test cases. This makes them undetectable when using inner products of predicate vectors.

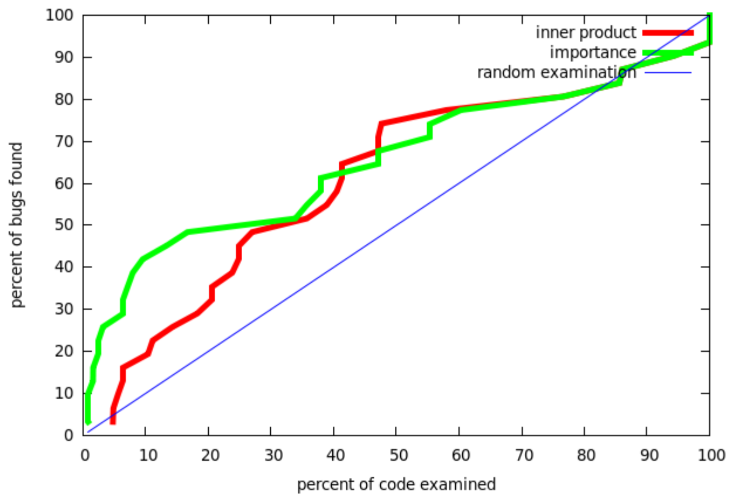

Figure 3: Results of using inner products of predicate vectors to rank statements, compared with importance scoring, for the TCAS benchmark.

The inner product displays some fault-finding capability, as shown in Figure 3, where it is compared with Liblit's importance score. Inner product suspiciousness consistently beats the diagonal line representing random examination of statements but is outperformed in most places by importance scoring. The results show that inner product suspiciousness ranking is especially deficient at getting the fault itself into the top few varsites. We hypothesize that the first few varsites flagged by inner product suspiciousness are likely to be false positives related to the fault – like the highly-ranked output statements in the Ising model (see Section 5). Such related statements may not be complete red herrings. They can provide insight into the nature and effects of a fault, especially if the distribution of varticles is retained and presented, as in Figures 1 and 2.

6.2.2 Stochastic TCAS

We repeated the test described in Section 6.2.1 on a modified stochastic version of the benchmark that injects random noise into certain variables. This version of the benchmark was created and first used by Gore et al. (2011) to represent exploratory simulations. The results of inner product suspiciousness and importance scoring are shown in Figure 4. The inner product approach performs better relative to importance, matching or surpassing it over much of the plot, but still falls short of importance for the first few lines of code examined.

7 DISCUSSION

We observed in Section 5 how the inner product varsite ranking can put false positives at the top of the list. We hypothesized that this leads to the inner product ranking's poor fault-finding performance when only the first few percent of varsites are examined.

A way to compensate for this weaknesses might be to rank varsites with hybrid metrics that combine inner product suspiciousness with other heuristics. Requiring a high score from a second method, for

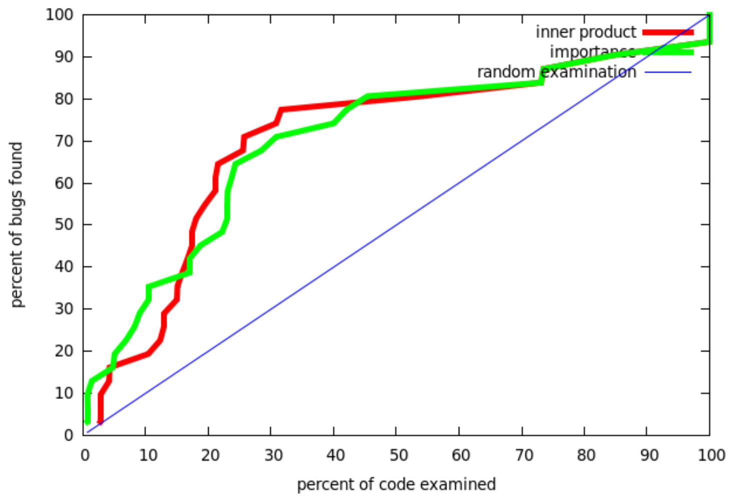

Figure 4: Results for the stochastic version of TCAS.

instance, might eliminate the false positives from the inner product. Liblit (2004) used this approach to derive his importance score: notice that (2) is a harmonic mean of two quantities. Liblit argues that, while each of those quantities can find some faults alone, their combination is stronger because it selects predicates that are suspicious for multiple reasons.

Our initial experiments with such combinations have been inconclusive. Determining the most effective way to apply inner products of predicate vectors will require thorough exploration of variations, guided by testing with additional benchmarks and simulations.

8 CONCLUSIONS

In this paper, we introduced and evaluated three ideas. First was a fuzzy pass/fail distinction for testing non-deterministic simulations. Second was the use of densities of variable states as a diagnostic tool for exploring unexpected behavior in simulations with real-valued random variables. Third was the use of inner products of predicate vectors as a quantitative comparison of variable state distributions in passing and failing tests.

Section 3.2 explained the theoretical advantage of fuzzy passing and the case study in Section 5 showed how the combination of fuzzy passing and variable state densities can help localize faults in Monte Carlo simulations. Section 6.2 demonstrated the effectiveness of predicate vector inner products as a statement ranking mechanism, especially for programs with random variables. Because predicate vectors and density distributions are closely related (see the introduction to Section 6), the successful testing of predicate vectors provides indirect support for using densities.

In addition to exploring the refinements to inner product rankings mentioned in Section 7, further work may consist of devising reliable ways to apply fuzzy passing to simulations with different types of outputs. Section 3.2 showed that there exist choices of a function for which fuzzy passing can improve some calculations, but did not provide a prescription for determining the function. Given some heuristics for choosing this function, we could go on to test fuzzy passing more thoroughly on standard benchmarks. We have also introduced but not yet explored several adjustable parameters, such as variable weights for outputs and components of predicate vectors.

ACKNOWLEDGMENTS

We gratefully acknowledge support from our colleagues at the University of Virginia and funding from Achievement Rewards for College Scientists (ARCS).

REFERENCES

Agrawal, H., and J. R. Horgan. 1990. “Dynamic program slicing”. In *Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation*, edited by K. S. McKinley, PLDI ’90, 246–256. New York, NY, USA: ACM.

Arumuga Nainar, P., T. Chen, J. Rosin, and B. Liblit. 2007. “Statistical debugging using compound boolean predicates”. In *Proceedings of the 2007 international symposium on Software testing and analysis*, edited by D. Rosenblum and S. Elbaum, ISSTA ’07, 5–15. New York, NY, USA: ACM.

Ayewah, N., D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh. 2008, September. “Using Static Analysis to Find Bugs”. *IEEE Softw.* 25:22–29.

Brush, S. 1967. “History of the Lenz-Ising Model”. *Reviews of Modern Physics* 39:883–893.

Chilimbi, T. M., B. Liblit, K. Mehra, A. V. Nori, and K. Vaswani. 2009. “HOLMES: Effective statistical debugging via efficient path profiling”. In *Proceedings of the 31st International Conference on Software Engineering*, edited by S. Fickas, ICSE ’09, 34–44. Washington, DC, USA: IEEE Computer Society.

Cleve, H., and A. Zeller. 2005. “Locating causes of program failures”. In *Proc. of the 27th Int. Conf. on Software Engineering*, edited by G.-C. Roman, W. G. Griswold, and B. Nuseibeh, 342–342.

J. Faber and J.C. Lombardi and F. Rasio 2003. “StarCrash: a parallel smoothed particle hydrodynamics (SPH) code for calculating stellar interactions”. Accessed April 3, 2011. http://ciera.northwestern.edu/StarCrash/manual/usersmanual.pdf.

Gingold, R., and J. Monaghan. 1977. “Smoothed particle hydrodynamics - Theory and application to non-spherical stars”. *Mon. Not. R. Astron. Soc.* 181:375–389.

Gore, R., P. Reynolds, and D. Kamensky. 2011. “Statistical Debugging with Elastic Predicates”. Technical Report No. CS-2011-02, Department of Computer Science, University of Virginia, Charlottesville, Virginia.

He, H., and N. Gupta. 2004. “Automated debugging using path-based weakest preconditions”. In *Proc. of the 2004 Fundamental Approaches to Software Engineering Conf.*, 267–280.

Huang, T., P. Chou, C. Tsai, and H. Chen. 2007. “Automated fault localization with statistically suspicious program states”. In *Proceedings of the 2007 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, edited by S. Pande and Z. Li, 11–10. San Diego, California: Association for Computing Machinery.

Jeffrey, D., N. Gupta, and R. Gupta. 2008. “Fault localization using value replacement”. In *Proceedings of the 2008 international symposium on Software testing and analysis*, edited by B. G. Ryder and A. Zeller, ISSTA ’08, 167–178. New York, NY, USA: ACM.

Jeffrey, D., N. Gupta, and R. Gupta. 2009. “Effective and efficient localization of multiple faults using value replacement”. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, edited by E. Stroulia, 221 –230.

Jones, J. A., and M. J. Harrold. 2005. “Empirical evaluation of the Tarantula automatic fault-localization technique”. In *Proc. of the 20th IEEE/ACM Int. Conf. on Automated Software Engineering*, edited by D. F. Redmiles, T. Ellman, and A. Zisman, 273–282. Long Beach, California: Association for Computing Machinery.

Korel, B., and J. Laski. 1988, October. “Dynamic program slicing”. *Inf. Process. Lett.* 29:155–163.

Liblit, B. 2004. *Cooperative bug isolation*. Ph.D. thesis, Department of Computer Science, University of California at Berkeley, Berkeley, California. Available via http://pages.cs.wisc.edu/~liblit/dissertation/dissertation.pdf [accessed April 3, 2011].

Liblit, B., A. Aiken, A. X. Zheng, and M. I. Jordan. 2003. "Bug isolation via remote program sampling". In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, edited by R. Cytron and R. Gupta, PLDI '03, 141–154. New York, NY, USA: ACM.

Liblit, B., M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. 2005. "Scalable statistical bug isolation". In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, edited by V. Sarkar and M. Hall, PLDI '05, 15–26. New York, NY, USA: ACM.

Liu, C., X. Yan, L. Fei, J. Han, and S. P. Midkiff. 2005. "SOBER: statistical model-based bug localization". In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, edited by M. Wermelinger and H. C. Gall, ESEC/FSE-13, 286–295. New York, NY, USA: ACM.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. 1953. "Equation of state calculations by fast computing machines". *Journal of Chemical Physics* 21:1087–1092.

Nainar, A., Piramanayagam, and B. Liblit. 2010. "Adaptive bug isolation". In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, edited by J. Kramer, J. Bishop, P. Devanbu, and S. Uchitel, ICSE '10, 255–264. New York, NY, USA: ACM.

Onsager, L. 1944. "Crystal statistics. I. A two-dimensional model with an order-disorder transition". *Phys. Rev.* 65:117–149.

Tip, F. 1994. "A Survey of Program Slicing Techniques.". Technical report, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands.

Weimer, W., S. Forrest, C. Le Goues, and T. Nguyen. 2010, May. "Automatic program repair with evolutionary computation". *Commun. ACM* 53:109–116.

Zhang, X., N. Gupta, and R. Gupta. 2006a. "Locating faults through automated predicate switching". In *Proceedings of the 28th international conference on Software engineering*, edited by L. J. Osterweil, D. Rombach, and M. L. Soffa, ICSE '06, 272–281. New York, NY, USA: ACM.

Zhang, X., N. Gupta, and R. Gupta. 2006b. "Pruning dynamic slices with confidence". In *Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, edited by M. Schwartzbach and T. Ball, PLDI '06, 169–180. New York, NY, USA: ACM.

AUTHOR BIOGRAPHIES

DAVID KAMENSKY Is an undergraduate at the University of Virginia. His e-mail is dmk3d@virginia.edu

ROSS GORE is a Ph.D. Candidate in Computer Science at the University of Virginia. His e-mail is rjg7v@virginia.edu

PAUL REYNOLDS is a Professor of Computer Science at the University of Virginia. He has conducted research in Modeling and Simulation for over 30 years, and has published on a variety of topics including parallel and distributed simulation, multi-resolution modeling and simulation. His e-mail is pfr@virginia.edu