Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

INVESTIGATING UNEXPECTED OUTCOMES THROUGH THE APPLICATION OF STATISTICAL DEBUGGERS

Kelsey Dutton
Ross Gore
Paul F. Reynolds, Jr.

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

ABSTRACT

Predictions from simulations with inherent uncertainty have entered the mainstream of public policy decision-making practices. Unfortunately, methods for gaining insight into unexpected simulation outcomes have not kept pace. Subject matter experts (SMEs) need to understand if the unexpected outcomes reflect a fault in the simulation or new knowledge. Recent work has adapted statistical debuggers, used in software engineering, to automatically identify simulation faults via extensive profiling of executions. The adapted debuggers have been shown to be effective, but have only been applied to simulations with large test suites and known faults. Here we employ these debuggers in a different manner. We investigate how they facilitate a SME exploring an unexpected outcome that reflects new knowledge. We also evaluate the debuggers in the face of smaller test suites and sparse execution profiling. These novel applications and evaluations show that these debuggers are more effective and robust than previously realized.

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. SMEs need to understand and explain unexpected simulation outcomes to determine if the behaviors reflect an error or if they represent new knowledge in the discipline. 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 conditions most relevant to the unexpected outcome that would assist the SME in understanding and explaining the behavior (Davis 2005).

Prior work in automatically localizing sources of unexpected outcomes in software, not necessarily simulations, has focused on statistical debugging. Statistical debugging is the process of profiling software executions to narrow or guide the search through source code to help a developer find statements containing faults that cause software failures (Liblit, Aiken, Zheng, and Jordan 2003; Liblit, Naik, Zheng, Aiken, and Jordan 2005). Recently, statistical debuggers have been adapted to localize faults specifically in simulations (Gore, Reynolds, and Kamensky 2011; Kamensky, Gore, and Reynolds Jr. 2011).

While statistical debuggers have been shown to be effective for identifying faults in simulations, their ability to assist SME explanation and understanding of a valid unexpected outcome as a reflection of new knowledge has not been evaluated. This capability is important. Validation of unexpected outcomes requires accumulation of insight into the outcome and the conditions under which it arises. In this work we perform a case study to evaluate if statistical debuggers can provide this capability to a SME attempting

to understand under-utilization in two different H2/D/1 queueing simulations. Ultimately, our case study shows that the adaptations made to statistical debuggers for simulations are crucial for accumulating insight and assisting SME explanation and understanding of under utilization in the simulations.

Next, we evaluate how much of the effectiveness of the statistical debuggers adapted for simulations is dependent on large test suites and full data profiling. Recall, these debuggers use source code instrumentation to profile the simulation during execution. The instrumentation, profiling and subsequent analysis can add significant computational overhead to the native execution time of the simulation. Furthermore, the debuggers rely on large test suites which are more common for traditional software than they are for simulations and also contribute to the overhead. We propose to improve the efficiency of the statistical debuggers adapted for simulations by reducing the amount of data profiled via sampling and reducing the size of the test suite the debugger employs. Our results show that these strategies can reduce computational time required by the adapted statistical debugging techniques and that the techniques remain effective, even in the face of aggressive sampling rates and test suites that are only 20% of their original size.

Our work has three novel contributions: (1) a case study evaluation of the ability of traditional and adapted statistical debuggers to facilitate SME understanding and explanation of unexpected simulation outcomes, (2) an evaluation of the application of sparse sampling to improve the efficiency of adapted statistical debuggers and (3) an evaluation of adapted statistical debuggers employing smaller, but carefully chosen, test suites to improve efficiency.

Section 2 provides an overview of statistical debuggers and work related to exploring unexpected simulation outcomes. Section 3 describes our case study and evaluation of the ability of traditional and adapted debuggers to facilitate SME understanding and explanation of unexpected simulation outcomes. Section 4 explores the effectiveness and efficiency of traditional and adapted statistical debuggers under sparse sampling and smaller test suites and Section 5 summarizes our contributions.

2 RELATED WORK

In this section, we present a summary of the statistical debuggers being employed and evaluated in our work and we review other work related to understanding and explaining unexpected simulation outcomes.

2.1 Statistical Debugging

Statistical debugging uses data collected during the execution of passing and failing test cases to rank predicates (conditional propositions) and statements based on the likelihood they contain a fault. This likelihood is referred to as the *suspiciousness* of the statement or predicate. The ranked statements and predicates are presented to the SME in descending order of *suspiciousness* to facilitate localizing the fault. Recently, three statistical debugging approaches stand out in terms of efficiency and effectiveness: Tarantula, Cooperative Bug Isolation (CBI), and Exploratory Software Predictor (ESP) (Jones, Harrold, and Stasko 2002; Jones and Harrold 2005; Liblit, Naik, Zheng, Aiken, and Jordan 2005; Gore, Reynolds, and Kamensky 2011). Here we describe each approach.

2.1.1 Tarantula

Tarantula is a statement-level statistical debugging approach. It analyzes how frequently a program statement is executed in passing and failing test cases. The more frequently a statement appears in failing test cases as opposed to passing test cases, the more suspicious it is. The approach is efficient because it only takes into account statement coverage information. However, its effectiveness is limited due to its inability to analyze values of variables within the faulty program (Jones, Harrold, and Stasko 2002; Jones and Harrold 2005).

2.1.2 CBI and ESP

Opposed to Tarantula, CBI and ESP are predicate-level statistical debuggers. All predicate-level statistical debuggers share a common structure. Each debugger uses a set of conditional propositions, or predicates, which are inserted into a program and tested at particular points. A single predicate can be thought of as partitioning the space of all test cases into two subspaces: those satisfying the predicate and those not. Better predicates create partitions that more closely match where the fault is expressed.

In the canonical predicate-level statistical debugger Cooperative Bug Isolation (CBI), three predicates are inserted and tested for each variable x within a program statement: $(x > 0)$, $(x = 0)$ and $(x < 0)$ (Liblit, Aiken, Zheng, and Jordan 2003; Liblit, Naik, Zheng, Aiken, and Jordan 2005). In the statistical debugger Exploratory Software Predictor (ESP), these three predicates are complemented with elastic predicates. Elastic predicates use profiling to compute the mean, μ_x, and standard deviation, σ_x, of the values of variable x. These summary statistics are used to form the elastic predicates: $(x > \mu_x + \sigma_x)$, $(\mu_x + \sigma_x \geq x \geq \mu_x - \sigma_x)$ and $(x < \mu_x - \sigma_x)$ (Gore, Reynolds, and Kamensky 2011).

Elastic predicate-level debuggers, such as ESP, are designed to target faults in simulations. Elastic predicates are effective for such faults because the predicates: (1) expand or contract based on observed variable values and (2) do not employ a rigid notion of equality (Gore, Reynolds, and Kamensky 2011). Further discussion of applying statistical debuggers to simulations, as opposed to general purpose software, is provided in (Gore, Reynolds, and Kamensky 2011; Gore and Reynolds Jr. 2012).

2.1.3 Suspiciousness Metrics

Predicate-level statistical debuggers require two data structures for each executed test case to compute suspiciousness for predicates. The two data structures are: (1) a feedback report, R, indicating if the test case was passed or failed and (2) a vector, V, with one entry for each instrumented predicate. Within V each entry indicates if the corresponding predicate is true at some point during execution of the test case (Liblit, Naik, Zheng, Aiken, and Jordan 2005; Liblit 2008). Once all test cases have been executed, the suspiciousness of each predicate can be calculated from these data structures.

Several different suspiciousness metrics for computing suspiciousness exist. The most effective estimates include a measure of *sensitivity* and *specificity*. Sensitive predicates account for a high percentage of failed test cases and specific predicates are less likely to be found in successful test cases (Manning, Raghavan, and Schütze 2008). The sensitivity and specificity for a predicate p are computed from each feedback report R and each corresponding vector V through four measures (Liblit, Naik, Zheng, Aiken, and Jordan 2005; Liblit 2008):

1. $s_{p\,obs}$ - the number of successful test cases in which p was evaluated (true or false).
2. $f_{p\,obs}$ - the number of failed test cases in which p was evaluated (true or false).
3. s_p - successful test cases in which p was evaluated and found to be true.
4. f_p - the number failed test cases in which p was evaluated and found to be true.

Sensitivity is computed using Equation 1, where *NumF* is the total number of failing runs. *Specificity* is reflected through the quantity *Increase* and is computed using Equation 2. *Increase* is the amount by which p being true increases the probability of failure. The first term in *Increase* is identical to the standard *specificity* measure used in information retrieval ($\frac{f_p}{f_p+s_p}$) . However, the second term in *Increase* is meant to ensure that predicate p is scored, not by the chance that p implies failure, but by how much difference it makes that p is true versus simply reaching the statement where p is evaluated (true or false). The suspiciousness metric balancing sensitivity and specificity via their harmonic mean, is shown in Equation 3. While the metric used to measure suspiciousness for CBI and ESP predicates is the same, the suspiciousness scores are not. The case study in Section 3 highlights the differences between ESP and CBI.

$$Sensitivity = \frac{f_p}{NumF} \tag{1}$$

$$Specificity = Increase = \frac{f_p}{f_p + s_p} - \frac{f_{pobs}}{f_{pobs} + s_{pobs}} \tag{2}$$

$$Suspiciousness = \frac{2}{1/\frac{f_p}{f} + 1/Increase} \tag{3}$$

2.2 Other Methods for Exploring Unexpected Outcomes

Our work is not the first to address facilitating SME explanation and understanding of unexpected outcomes. Here, we review existing research in this area. *Sensitivity analysis* has been proposed as a methodology to explore the robustness of the behavior in a simulation (Cacuci 2003). The principle behind sensitivity analysis is to vary the initial parameters of the model by a small amount and rerun the simulation. This allows the SME to understand how sensitive the model is to the initial parameters. However, even with a small number of parameters, the number of combinations of parameter values quickly becomes large and the resources required to perform the analysis can be excessive (Cacuci 2003).

Sensitivity analysis has been refined to *exploratory analysis*. Exploratory analysis can be viewed as sensitivity analysis done efficiently (Davis 2000). Exploratory analysis relies on user insight to limit the number of parameters that need to be explored to gain a broad understanding of the potential behaviors of a model. The approach is characterized in part by parametric exploration. Parametric exploration involves conducting model runs across cases defined by discrete values of the parameters within their plausible domains. The differences in the outcomes of the model runs are examined to determine the parameters or inputs that affect the behavior (Davis 2000).

Similarly, *Explanation Exploration* (EE) is a method for increasing insight into unexpected emergent outcomes in simulations, and providing a path to validation of those behaviors that are valid. EE is a process for increasing confidence and insight into an unexpected outcome; it is not a complete validation method or strategy. The method incorporates semi-automated exploration of conditions in which a user can test hypotheses about emergent behaviors, and thereby increase confidence about their assessment of the meaning of the emergent behaviors. Validation is a goal, but not necessarily an outcome of the method (Gore, Reynolds Jr., Tang, and Brogan 2007; Gore and Reynolds 2008; Reynolds, Spiegel, Liu, and Gore 2007).

While sensitivity analysis, exploratory analysis and explanation exploration can facilitate a SME in need of explaining an unexpected outcome, none of the approaches offers a fully automated method to identify conditions with the simulation source code that are strongly correlated with the outcome in question. The statistical debuggers reviewed in Section 2.1, especially ESP, offer this capability. As a result we focus on the use of these debuggers in the remainder of this paper.

3 CASE STUDY

The M/M/1 queueing model is the canonical single server queueing system described by the arrival rate and service rate of jobs in the model. The arrival rate is sampled from a Poisson distribution and the service rate is sampled from an exponential distribution. We use the H2/D/1 queueing model in the case study presented in this section to evaluate how effective statistical debuggers are at facilitating SME understanding and explanation of valid, but unexpected behaviors. The properties of the H2/D/1 model mirror the M/M/1 model, except that the H2/D/1 model allows for experimentation of the burstiness of arrivals through the manipulation of a covariance coefficient input parameter (Christensen 2012).

One measure of performance in the H2/D/1 model is *server utilization*. *Server utilization* is the proportion of time the server is busy divided by the amount of time the system's resources are in use (Lemis

and Park 2006). In our case study we explore H2/D/1 configurations (test cases) in which we expected server utilization to be > 0. However, we found for some of the H2/D/1 configurations (test cases) the server utilization ≤ 0. The implementation of the H2/D/1 model we employ is publicly available and widely tested (Christensen 2012). As a result the unexpected outcome (server utilization ≤ 0) for several of the H2/D/1 test cases is not due to a fault in the implementation but instead the result of an unexpected assumption within the model.

We employ the statistical debuggers ESP and CBI to facilitate explanation of those H2/D/1 test cases where the server utilization is ≤ 0. Recall, CBI employs static predicates where the conditional propositions that are tested to gain insight are specified before execution, while ESP complements these static predicates with elastic predicates that adapt to observed variables profiled during execution.

The H2/D/1 model is run over two test suites, each consisting of ten test cases. The first suite held service time constant at 1.0 to observe the effect of arrival time variance. This test suite is shown in Table 1. The second set introduced variance in the rate of service in an effort to expose the relationship between the arrival and service rates of jobs. This test suite is shown in Table 2. Values for the test cases were generated randomly. Input values for arrival rates are drawn from a standard normal distribution with a mean of one and a standard deviation of 0.1. Similarly, in the second test suite, service time is drawn from a standard normal distribution with a mean of 0.5 and a standard deviation of 0.1. Covariance is bounded between (-5, 5) in an effort to limit its influence on system results. Test cases which produce an expected outcome of server utilization > 0 are classified as passing, while those that produce a server utilization ≤ 0 are unexpected and classified as failing.

Table 1: H2/D/1 Test Suite I.

Simulation Time	Arrival Time	Covariance	Service Time	Expected/Unexpected
50.221	3.959	1.721	1.0	Passing
89.386	17.233	0.727	1.0	Passing
76.231	27.218	0.535	1.0	Passing
87.784	22.725	-4.157	1.0	Passing
81.769	28.317	-2.768	1.0	Failing
85.785	30.593	4.011	1.0	Passing
11.334	19.354	-3.086	1.0	Passing
69.466	83.471	-1.371	1.0	Failing
14.563	18.423	4.060	1.0	Passing
9.403	250.484	1.361	1.0	Failing

Table 2: H2/D/1 Test Suite II.

Simulation Time	Arrival Time	Covariance	Service Time	Expected/Unexpected
50.532	2.031	1.257	1.801	Failing
23.547	1.891	1.256	1.800	Passing
75.496	1.881	1.251	1.512	Failing
95.346	1.101	1.552	1.601	Failing
15.365	1.102	3.552	1.629	Passing
35.365	1.202	1.552	1.199	Passing
55.365	1.302	2.552	1.299	Passing
95.365	1.291	2.983	1.399	Passing
35.365	1.291	2.983	1.099	Passing

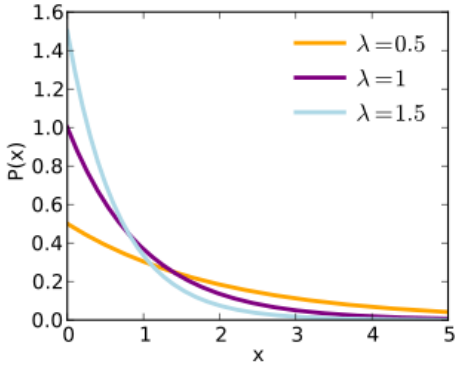

Figure 1: The hyper-exponential distribution.

Test cases with a constant service time suffer from poor sampling of the hyper-exponential distribution used in the H2/D/1 simulation to define the arrival rate of jobs. The arrival rate is an input parameter provided by the user upon execution. The covariance, also an input parameter, defines the burstiness of arrivals. These two parameters are used to compute the arrival rate of jobs through the sampling of the hyper-exponential distribution. Figure 1 shows the shape of the curve as a function of the covariance. If chosen unwisely these parameters will allow the arrival rate to vary in a way such that server utilization is zero. The hyper-exponential function uses a squared covariance to measure the randomness in its parameters, in this case the arrival rate (Allen 1990). This squared covariance is used to compute a probability, which determines how the exponential distribution of the first parameter, the arrival rate, is drawn from for sampling. The algorithm below details this sampling.

HYPEREXPONETIAL$(arrTime, cov)$

1 $z_1 \leftarrow 0$
2 $z_2 \leftarrow 0$
3 **while** $z_1 = 0$ **or** $z_1 = 1$
4 **do**
5 $z_1 \leftarrow$ RAND() **comment:** Draw uniform random number 0 -1
6 **while** $z_2 = 0$ **or** $z_2 = 1$
7 **do**
8 $z_2 \leftarrow$ RAND() **comment:** Draw uniform random number 0 -1
9 $temp \leftarrow cov \times cov$
10 $p \leftarrow \frac{1}{2} \times 1 - \sqrt{\frac{temp-1}{temp+1}}$
11 **if** $z_1 > p$
12 **then**
13 $temp \leftarrow \frac{arrTime}{1-p}$
14 **else**
15 $temp \leftarrow \frac{arrTime}{p}$
16 $hyperValue \leftarrow \frac{1}{2} \times temp \times \log(z_2)$
17 **return** $hyperValue$

Through the investigation of the most suspicious elastic predicates this function was localized as the source of the unexpected outcome. Predicates associated with server utilization, covariance, and arrival rate were among the top ranked predicates for analysis. Server utilization was calculated as the quotient

of the time since the last start of server use, or *busytime* as defined in the H2/D/1 model, divided by the simulation time. This *busytime* is determined by the arrival rates of jobs. Arrivals rates are drawn from an exponential distribution determined by the hyper-exponential function. Arrival rates are drawn from a distribution centered around the the user-provided arrival time. When the arrival time is large the exponential distribution has greater likelihood of producing a number near a positive infinity, or at least near values that are larger than the time specified for the simulation duration. When this is the case and this larger distribution is randomly sampled, no jobs arrive for processing before the simulation's end, thus resulting in a server utilization of zero.

The use of elastic predicates was better than static in this analysis. Elastic predicates directed the SME to the comparison of the square of the covariance to the randomly generated number in the hyper-exponential function. This was significant because this comparison was used to calculate how the distribution is sampled for the job arrival rates. The random number was generated to fall between zero and one. When the covariance is large and the random number falls closer to one than to zero the exponential distribution centered around job arrival rate is sampled with probability 1-p. However, if covariance is small and the random number falls closer to zero than one, the exponential distribution centered around the arrival rate is sampled with probability p. If the user chooses a large input parameter for the arrival rate, as was the case in the event of all the unexpected outcomes, it is likely that a large wait time will be assigned to at least one job. This wait time could be larger than the simulation time, which results in zero job completions and a server utilization of zero. While static predicates also marked the calculation of covariance as highly suspicious, they did not implicate the random number generation, and thus required further investigation of additional predicates to localize the root of the unexpected behavior.

The second instance of unexpected, but valid system behavior is found in those test cases with variance in job arrival and service rates. The results reflect the importance of the relationship between the two. In the first test suite the covariance was an important factor in fault localization. Large values for job arrival rate led to a wide distribution and unexpected system behavior. However, the second test suite cases shows how poor selection of service time can also lead to poor sampling and thus unexpected behavior.

Again predicates associated with server utilization, arrival time, and service time were identified as highly suspicious. However this time the addition of the arrival rates to the simulation resulted in the unexpected system behavior. In the simulation the values for arrival and service rate are compared to the simulation time to schedule the arrival and departure of jobs. When the service rate exceeds the arrival time the server is not utilized and no customers are processed by the system, resulting in an unexpected result. In some cases the arrival rate was slow (a large value was provided by the user), which resulted in scheduling of jobs after the completion of the simulation. Jobs are scheduled for departure based on the sum of their arrival and service rates. The simulation time is updated based on these events. When these values are large they exceed the total available time and the system produces server utilization of zero, despite valid response to user-provided values.

Elastic predicates were found to be particularly helpful. Service time was flagged as a candidate for exploration; a flag that was missing in the static analysis. The flagging of the predicate for service time suggests that the choice for service time greater than arrival time is not appropriate. Additionally, the comparison of job arrival and service time is marked as a highly suspicious elastic predicate, indicating the importance of the relationship between the two.

In both case studies the use of the elastic statistical predicates in ESP cut down analysis time in comparison to traditional manual methods of simulation understanding and the static predicates used in other debuggers such as CBI.

The generality and pervasiveness of the M/M/1 and H2/D/1 queueing models make then more than small test cases for the application of statistical debugging. Both the M/M/1 and H2/D/1 models are representative of a large class of simulations that draw random variables from continuous stochastic distributions to model time and make extensive use of floating-point data types to compute measures of efficiency and effectiveness.

We expect that the results of larger case studies employing more complex models of multiple queues will mirror our reported results.

4 SPARSE SAMPLING AND INCOMPLETE TEST SUITES

Recall the predicates that enabled explanation and understanding of the unexpected outcomes of the H2/D/1 queueing simulations require source code instrumentation. While the instrumentation creates effective analysis it also incurs significant additional computational time compared to the native execution time for the simulation. For example, the H2/D/1 queueing simulation takes more than 45x longer to execute when ESP is applied than it does natively. If this 45x overhead can be reduced, then SMEs will be able to explain and understand unexpected outcomes using statistical debuggers more efficiently.

Previous evaluations have shown that the efficiency of similar predicate-level statistical debuggers such as CBI can be improved without a significant reduction in effectiveness by employing sparse execution profiling and smaller test suites (Liblit 2008). Here we explore if ESP is also robust enough to be effective under these conditions. Sections 4.1 and 4.2 explore how effective ESP is in the face of sparse sampling rates and smaller test suites.

4.1 Sparse Sampling

Here, we explore if the computational overhead added by the instrumentation in ESP can be limited by employing sparse, unbiased random sampling of the instrumented predicates rather than always profiling every predicate. The sampling collects a representative subset of all of the predicates across the simulation test suite. While sampling in CBI has been previously explored, exploring sampling in ESP is a novel contribution. The goal is to determine the extent to which the sampling improves the efficiency of ESP and extent to which the uncertainty introduced by sampling reduces the effectiveness of ESP.

Our evaluation of employing sampling in ESP consists of 18 different faulty versions of two different M/M/1 queueing simulations adapted from two fault localization benchmarks which implement priority queue schedulers (SIR 2012). Each of the simulations is a system with a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution. Each simulation has at least 2,500 test cases with the correct or *expected* output specified for each test case. If a faulty version produces the expected output it passes the test case, otherwise it fails.

All of the faults included in the queueing simulations are computation-related, as opposed to memory-related. These faults reflect operator and operand mutations, missing and extraneous code, and constant value mutations. The faults were included with the source code for the schedulers we adapted into M/M/1 queueing simulations and reflect actual faults made by software developers (Jones and Harrold 2005; Abreu, Zoeteweij, and van Gemund 2007; Jeffrey, Gupta, and Gupta 2008; Jeffrey, Gupta, and Gupta 2010; SIR 2012). For faulty versions with a faulty constant assignment statement, the assignment statement is considered to be localized by any predicate using the constant. For subject program versions where the fault reflects a missing statement, predicates corresponding to statements directly adjacent to the missing code qualify as localizing the missing statement. These issues are handled the same way they are in the published work of others in the statistical debugging community (Jones and Harrold 2005; Abreu, Zoeteweij, and van Gemund 2007; Jeffrey, Gupta, and Gupta 2008; Jeffrey, Gupta, and Gupta 2010).

To study the effectiveness of ESP in the face of sparse sampling, an established cost metric is employed (Cleve and Zeller 2005). Given the ranked set of predicates, the metric *Cost*, measures the percentage of predicates a SME must examine before encountering the fault. If there are ties, it is assumed that the developer must examine all of the tied predicates. For example, if there are n instrumented predicates and all n predicates have the same suspiciousness score, it is assumed that the developer must examine all of the n predicates. Therefore the *Cost* of finding the fault in this scenario is 100%. A lower score is preferable because it means that fewer of the predicates must be considered before the faulty statement is found.

Table 3: Average instrumentation overhead for CBI and ESP at different sampling rates for the 18 faulty versions of the queueing simulation.

Approach	Always	$\frac{1}{10}$	$\frac{1}{100}$	$\frac{1}{1{,}000}$	$\frac{1}{10{,}000}$
CBI	570%	2,708%	340%	118%	21%
ESP	2,568%	10,621%	1,413%	419%	86%

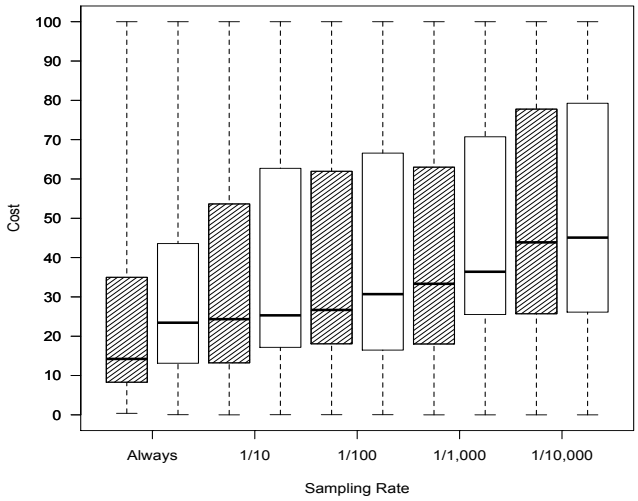

Figure 2: CBI (non-striped) and ESP (striped) under sparse sampling rates for the queueing simulations.

The *Cost* of ten executions of the 18 faulty versions of the queueing simulations included in the evaluation under ESP (striped) and CBI (non-striped) with sampling rates from $\frac{1}{10}$ to $\frac{1}{10{,}000}$ are plotted in Figure 2. The bottom and top of each box in Figure 2 represent the lower and upper quartile *Cost* and the black band is the median *Cost*. The whiskers extend to the lowest and highest *Cost*. In general less frequent sampling results in less complete data, less effective analysis and higher Cost. The effectiveness of CBI has already been shown to degrade gracefully in the face of sparse sampling. It is included here to serve as a baseline for the effectiveness of ESP. The sampling rate which causes ESP to become as or less effective than CBI denotes the rate at which it is no longer useful to explore employing sampling to improve the efficiency of ESP.

Figure 2 shows that the effectiveness of CBI remains more stable under sampling rates of $\frac{1}{10}$ and $\frac{1}{100}$ than the effectiveness of ESP. Once sampling is introduced the median *Cost* of localizing a fault in ESP significantly increases. While ESP continues to remain more effective than CBI by an absolute margin, the relative difference in effectiveness between the two approaches narrows. At a sampling rate of $\frac{1}{1{,}000}$ both ESP and CBI begin to become significantly less effective.

The performance of ESP at a sampling rate of $\frac{1}{10{,}000}$ reveals a trend: sufficiently infrequent sampling rates will reduce the effectiveness of ESP to that of CBI. In these cases the mean and standard deviation of each program point is based on so little data that the resulting elastic predicates are no better, and often worse, than the static and uniform predicates at predicting program failure. However, ESPs performance under more frequent sampling rates shows that elastic predicates do not require an exact calculation of the mean and standard deviation of values at each program point. Even at infrequent rates like $\frac{1}{1{,}000}$, estimations of the mean and standard deviation in the ESP elastic predicates result are effective.

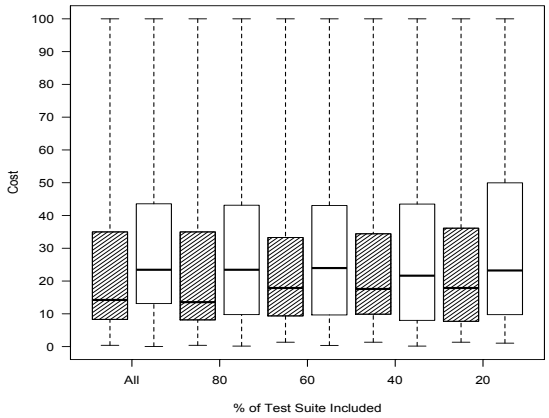

Figure 3: CBI (non-striped) and ESP (striped) with sparse test suites for the queueing simulations.

This is significant. As Table 3 shows, the computation required to perform unbiased sampling at less aggressive rates ($\frac{1}{10}$) can actually add additional overhead to ESP and CBI. However, at sufficiently aggressive rates ($\frac{1}{100}$, $\frac{1}{1{,}000}$) the overhead can be significantly reduced and result in efficient and effective localization of the sources of unexpected simulation outcomes due to faults. Next we explore how significantly the size of the simulation test suites used with ESP can be reduced to allow the approach to remain effective.

4.2 Incomplete Test Suites

Here we evaluate if the efficiency of ESP can also be improved by using a smaller test suite without significantly reducing the debugger's effectiveness. Test cases from the original test suite for each faulty version of the queueing simulations are included in the evaluation. These test cases were chosen at random, forming sparse test suites at $1/5^{\text{th}}$, $2/5^{\text{th}}$, $3/5^{\text{th}}$ and $4/5^{\text{th}}$ the size of the original suite. Each test case was chosen with uniform random probability without replacement and if the resulting sparse test suite did not contain at least 10 failing test cases and at least twenty successful (passing) test cases it was dissolved and the sparse test suite was reformed. Figure 3 summarizes the effect of each of these smaller test suites on ESP (striped). Once again, since CBI (non-striped) has been shown to be stable under smaller test suites it is included in the evaluation as a performance baseline.

The effectiveness of ESP is stable across sparse test suites of different sizes for the subject programs included in the evaluation. Under the sparsest test suite included in the evaluation, $1/5^{\text{th}}$ of the complete test suite, both ESP and CBI show larger variation in effectiveness compared to the other test suite sizes. However, the median effectiveness of each technique at this test suite size is similar to the median effectiveness of each technique when all test cases are included. Overall, Figure 3 reveals that for the subject programs included in the evaluation, if the test suite formed features at least twenty successful (passing) test cases and ten failing test cases then the overall size of the test suite has little effect on the *Cost* incurred when applying ESP. In this regard ESP is similar to CBI and can benefit from the same improvements in efficiency (Abreu, Zoeteweij, and van Gemund 2007).

Table 4: Average instrumentation overhead for CBI and ESP with different test suite sizes.

Approach	Full	$4/5^{\text{th}}$	$3/5^{\text{th}}$	$2/5^{\text{th}}$	$1/5^{\text{th}}$
CBI	570%	473%	359%	239%	119%
ESP	2,568%	2,208%	1,741%	1,272%	770%

Table 4 shows the efficiency results of employing ESP and CBI with smaller test suites. It is important to note that there is significantly less computational overhead required to identify a smaller test suite than required to perform unbiased sampling. With CBI the size of the test suite almost *linearly* reduces the time required to apply the approach to the queueing simulations. This is not true with ESP where the computation and insertion of elastic predicates prevents linear speedup. However, ESP still becomes significantly faster with smaller test suites and remains effective, demonstrating that it is a more robust approach to identifying sources of unexpected simulation outcomes due to faults than previously realized.

5 CONCLUSION

Methodology to improve SME understanding and explanation of unexpected outcomes in exploratory simulations is needed. Our investigation has three novel contributions: (1) a case study evaluation of the ability of traditional and adapted statistical debuggers to facilitate SME understanding and explanation of unexpected simulation outcomes, (2) an evaluation of the application of sparse sampling rate to improve the efficiency of adapted statistical debuggers and (3) an evaluation of adapted statistical debuggers employing smaller, but carefully chosen, test suites to improve efficiency. In future work, we will continue to explore additional enhancements for statistical debuggers that can improve the process of localizing the source of unexpected outcomes in exploratory simulations for SMEs.

REFERENCES

Abreu, R., P. Zoeteweij, and A. J. C. van Gemund. 2007. "On the Accuracy of Spectrum-based Fault Localization". In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, edited by M. Harman and P. McMinn, 89–98. Washington, DC, USA: IEEE Computer Society.

Allen, A. O. 1990. *Probability, Statistics, and Queueing Theory with Computer Science Applications*. 2nd ed. San Diego: Academic Press Inc.

Cacuci, D. 2003. *Sensitivity and Uncertainty Analysis: Theory*. Sensitivity and Uncertainty Analysis. Chapman & Hall/CRC.

Ken Christensen 2012. "Christensen Tools Page". Accessed Mar. 12, 2012. http://www.csee.usf.edu/~christen/tools/toolpage.html.

Cleve, H., and A. Zeller. 2005. "Locating causes of program failures". In *Proceedings of the 27th international conference on Software engineering*, edited by G.-C. Roman, W. G. Griswold, and B. Nuseibeh, ICSE '05, 342–351. New York, NY, USA: ACM.

Davis, P. K. 2000, December. "Dealing with complexity: exploratory analysis enabled by multiresolultion, multiperspective modeling". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 293–302. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Davis, P. K. 2005, December. "New paradigms and new challenges". In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 1067–1076. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gore, R., and P. F. Reynolds. 2008, December. "Applying causal inference to understand emergent behavior". In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Moench, O. Rose, T. Jefferson, and J. W. Fowler, 712–721. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gore, R., P. F. Reynolds, Jr., and D. Kamensky. 2011. "Statistical Debugging with Elastic predicates". In *Proceedings of the 26th IEEE/ACM international Conference on Automated software engineering*, edited by N. Rungta, ASE '11, 273–282. New York, NY, USA: ACM.

Gore, R., and P. F. Reynolds Jr.. 2012. "Modifying Test Suite Composition to Enable Effective Predicate-Level Statistical Debugging". In *NASA Formal Methods*, edited by A. Goodloe and S. Person, Volume 7226 of *Lecture Notes in Computer Science*, 70–84. Springer Berlin / Heidelberg.

Gore, R., P. F. Reynolds Jr., L. Tang, and D. C. Brogan. 2007. "Explanation Exploration: Exploring Emergent Behavior". In *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, edited by K. S. Perumalla and G. F. Riley, PADS '07, 113–122. Washington, DC, USA: IEEE Computer Society.

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. 2010. "Effective and Efficient Localization of Multiple Faults Using Value Replacement".

Jones, J. A., and M. J. Harrold. 2005. "Empirical evaluation of the tarantula automatic fault-localization technique". In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, edited by D. F. Redmiles, T. Ellman, and A. Zisman, ASE '05, 273–282. New York, NY, USA: ACM.

Jones, J. A., M. J. Harrold, and J. Stasko. 2002. "Visualization of test information to assist fault localization". In *Proceedings of the 24th International Conference on Software Engineering*, edited by W. Tracz, M. Young, and J. Magee, ICSE '02, 467–477. New York, NY, USA: ACM.

Kamensky, D., R. Gore, and P. F. Reynolds Jr.. 2011, December. "Applying enhanced fault localization technology to Monte Carlo simulations". In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 2798–2809. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lemis, L., and S. Park. 2006. *Discrete Event Simulation: A First Course*. New York: Pearson Prentice Hall.

Liblit, B. 2008. "Cooperative debugging with five hundred million test cases". In *Proceedings of the 2008 international symposium on Software testing and analysis*, edited by B. G. Ryder and A. Zeller, ISSTA '08, 119–120. New York, NY, USA: ACM.

Liblit, B., A. Aiken, A. X. Zheng, and M. I. Jordan. 2003, May. "Bug isolation via remote program sampling". *SIGPLAN Not.* 38:141–154.

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. W. Hall, PLDI '05, 15–26. New York, NY, USA: ACM.

Manning, C. D., P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press.

Reynolds, Jr., P. F., M. Spiegel, X. Liu, and R. Gore. 2007. "Validating Evolving Simulations in COERCE". In *Proceedings of the 7th international conference on Computational Science, Part I: ICCS 2007*, edited by D. van Albada, ICCS '07, 1238–1245. Berlin, Heidelberg: Springer-Verlag.

SIR 2012. "Software Infrastructure Repository". Accessed Mar. 12, 2012. http://sir.unl.edu/portal/index.php.

AUTHOR BIOGRAPHIES

KELSEY DUTTON is an undergraduate at the University of Virginia. Her e-mail is kjd9ub@virginia.edu.

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

PAUL REYNOLDS is a Professor of Computer Science at the University of Virginia. His e-mail is pfr@virginia.edu.