Prioritizing Mutation Operators using Probabilistic Sampling

International Symposium on Software Reliability Engineering (ISSRE 2010)

Mohan Sridharan[1] Akbar Siami Namin[2]

[1] Stochastic Estimation and Autonomous Robotics Lab (SEARL)
Department of Computer Science
Texas Tech University, USA
mohan.sridharan@ttu.edu

[2] AdVanced Empirical Software Testing and Analysis (AVESTA) Research Group
Department of Computer Science
Texas Tech University, USA
akbar.namin@ttu.edu

Outline

- **Introduction:**
  - Motivation.
  - Related Work.
- **Proposed Approach:**
  - Hypothesis.
  - Stochastic sampling.
  - Probabilistic formulation.
- **Experimental Results**:
  - Case studies.
  - Figures and tables.
  - Discussion.
- **Conclusions**:
  - Conclusions.
  - Further Reading.

Outline

- **Introduction:**
  - Motivation.
  - Related Work.
- **Proposed Approach:**
  - Hypothesis.
  - Stochastic sampling.
  - Probabilistic formulation.
- **Experimental Results**:
  - Case studies.
  - Figures and tables.
  - Discussion.
- **Conclusions**:
  - Conclusions.
  - Further Reading.

Mutation Testing

- Fault-based testing technique for assessing the adequacy of test suites.
- Artificially inject syntactic faults in the program.
- *Mutation operators*: mathematical transformations to synthesize faults, i.e., mutants.
- E.g. Binary operator to binary operator (BoB):

$$\begin{array}{ccc} \dots & & \dots \\ if(a > b)\ then & \rightarrow & if(a < b)\ then \\ \dots & & \dots \end{array}$$

- Augment test suites to detect unexposed faults.

Mutation Generation

- Each operator represents a specific type of fault.
- Several mutation operators have been created.
- Proteum:
  - A mutation tool for C with 108 operators.
- MuJava:
  - A mutation tool for Java.
  - 12 traditional mutation operators. 28 class mutation operators.

Feasibility of Mutation Testing

- Each mutant has to be compiled and executed for the entire pool of test cases.
- The number of mutants generated for the target program can be large.
    - For instance, Proteum (108 operators) generated 4935 mutants on a program with 137 lines of code.
- Examining all mutants can be intractable!

Mutation Score

- Mutation score definition:

$$MS = AM = \frac{\#killed\ mutants}{\#Total\ non-equivalent\ mutants} \tag{1}$$

- Two possible options:
  - Mutants are exposed by the existing test suite.
  - Mutants are hard to kill and the test suite needs to be augmented.

- Drawback: *The mutation score can grow without reflecting the quality of testing.*

Reducing Cost of Mutation Testing

- Standard mutation testing [DeMillo et al, 1978; Hamlet 1977].
- Meta-mutants [Offutt and Untch, 2000]:
  - Creating and compiling one file to include all mutants.
- Selective mutation [Offutt et al.,1996]:
  - Considering only a subset of mutation operators.
- Sufficient mutation operators [Namin et al., 2008]:
  - Identifying a sufficient set of mutation operators using regression and other estimation methods.

Outline

- **Introduction:**
  - Motivation.
  - Related Work.
- **Proposed Approach:**
  - Hypothesis.
  - Stochastic sampling.
  - Probabilistic formulation.
- **Experimental Results**:
  - Case studies.
  - Figures and tables.
  - Discussion.
- **Conclusions**:
  - Conclusions.
  - Further Reading.

Our Hypothesis:

A mutation operator, most of whose mutants have been caught by the existing test suite, is likely to produce mutants that can be exposed using the existing test suite.

Intuitive Idea

- Focus on operators that are *likely* to generate mutants that will need an augmentation of the test suite.
- Definition of *important* operators:
    - Operators that tend to generate mutants that need an augmentation of the test suite.
- Prioritize mutation operators based on their importance.
- Consider the top few operators for detailed analysis.

Stochastic Sampling

- Used to track multiple hypotheses in practical domains.
- Probabilistic representation for each hypothesis.
- Iteratively focus on the more important hypotheses.

- Sampling algorithms: rejection sampling, importance sampling, Gibbs sampling, MCMC.
- Several applications:
  - Tracking multiple humans in image sequences.
  - Finding likely locations of celestial objects, i.e., in astronomy.

Stochastic Sampling

- Used to track multiple hypotheses in practical domains.
- Probabilistic representation for each hypothesis.
- Iteratively focus on the more important hypotheses.

- Sampling algorithms: rejection sampling, importance sampling, Gibbs sampling, MCMC.
- Several applications:
  - Tracking multiple humans in image sequences.
  - Finding likely locations of celestial objects, i.e., in astronomy.

Importance Sampling Overview

- Assign probability to each hypothesis.
- Generate initial set of *samples* of each hypothesis based on the corresponding probabilities.
- In each of a set of iterations:
  - Adjust samples to account for dynamic changes in the system: *prediction* step.
  - Use observations of the system to update probabilities of the samples: *correction* step.
  - Resample, i.e., generate samples of each hypothesis proportional to the updated probabilities.

Importance Sampling Overview

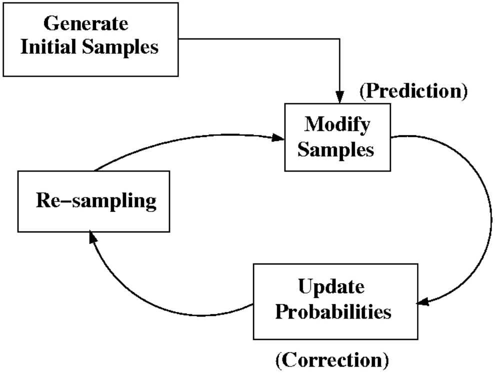
- Assign probability to each hypothesis.
- Generate initial set of *samples* of each hypothesis based on the corresponding probabilities.
- In each of a set of iterations:
  - Adjust samples to account for dynamic changes in the system: *prediction* step.
  - Use observations of the system to update probabilities of the samples: *correction* step.
  - Resample, i.e., generate samples of each hypothesis proportional to the updated probabilities.

Importance Sampling Overview

- Assign probability to each hypothesis.
- Generate initial set of *samples* of each hypothesis based on the corresponding probabilities.
- In each of a set of iterations:
    - Adjust samples to account for dynamic changes in the system: *prediction* step.
    - Use observations of the system to update probabilities of the samples: *correction* step.
    - Resample, i.e., generate samples of each hypothesis proportional to the updated probabilities.

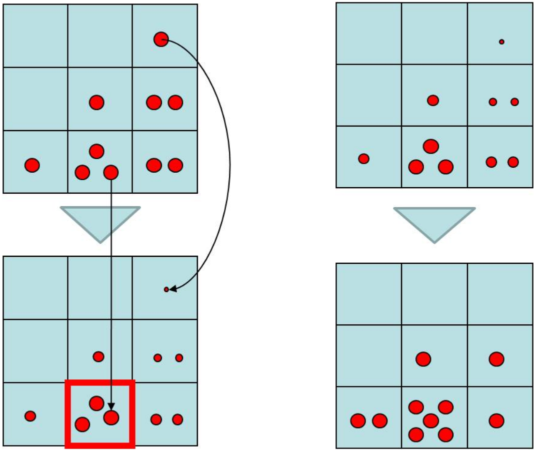
Importance Sampling

- The typical importance sampling framework:



Pictorial Representation

- Probabilistic update and re-sampling:



Importance Sampling for Mutation Testing

- Probability for each mutation operator: p_i for $\mu_i \forall i \in [1, N]$.
- Select initial (small) set of mutant samples of each operator, choosing *uniformly* or *proportional* to operator probabilities:

$$numMutantSamps_i^0 \simeq \begin{cases} c & \text{uniform} \\ \propto \frac{Nm_i}{NM} & \text{proportional} \end{cases} \qquad (2)$$

Importance Sampling for Mutation Testing

- Probability for each mutation operator: p_i for $\mu_i \, \forall i \in [1, N]$.
- Select initial (small) set of mutant samples of each operator, choosing *uniformly* or *proportional* to operator probabilities:

$$numMutantSamps_i^0 \simeq \begin{cases} c & \text{uniform} \\ \propto \frac{Nm_i}{NM} & \text{proportional} \end{cases} \quad (2)$$

Sampling Iterations

- Examine the ability of existing test suites to expose selected mutants.
- Increase probabilities of operators whose mutants are unexposed.

$$p_i^t = p_i^{t-1} + \frac{\delta p_i^t}{totalMutantSamps^t} \tag{3}$$

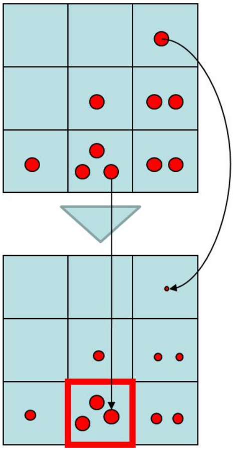
$$\delta p_i^t = -1.0 + 2.0 \frac{numAlive_i^t}{numMutantSamps_i^t} \quad : \in [-1.0, 1.0]$$

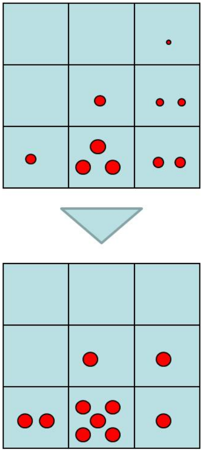
$$totalMutantSamps^t = \sum_{i=0}^{N-1} numMutantSamps_i^t$$

- Generate samples of each operator proportional to probabilities.

Recap: Pictorial Representation

- Probabilistic update and re-sampling:





Innovations in Sampling

- Sample without replacement: *stationary* system.
- *Adapt* number of samples based on uncertainty:

$$\mathcal{Y}^{t+1} = \frac{1}{2\epsilon}\chi^2_{q^t-1,1-\delta} \tag{4}$$
$$\simeq \frac{q^t-1}{2\epsilon}\left\{1 - \frac{2}{9(q^t-1)} + \sqrt{\frac{2}{9(q^t-1)}}z_{1-\delta}\right\}^3$$

- Entropy in operator probability distribution:

$$E^t = -\sum_{j=0}^{N-1} p_j^t \cdot ln(p_j^t) \tag{5}$$

- *Termination condition:* $E^t - E^{t-1} \leq \text{threshold}$

Innovations in Sampling

- Sample without replacement: *stationary* system.
- *Adapt* number of samples based on uncertainty:

$$\begin{aligned}\mathcal{Y}^{t+1} &= \frac{1}{2\epsilon}\chi^2_{q^t-1,1-\delta} \quad (4)\\ &\simeq \frac{q^t-1}{2\epsilon}\left\{1-\frac{2}{9(q^t-1)}+\sqrt{\frac{2}{9(q^t-1)}}z_{1-\delta}\right\}^3\end{aligned}$$

- *Entropy* in operator probability distribution:

$$E^t = -\sum_{j=0}^{N-1} p_j^t \cdot ln(p_j^t) \quad (5)$$

- *Termination condition:* $E^t - E^{t-1} \leq \text{threshold}$

Innovations in Sampling

- Sample without replacement: *stationary* system.
- *Adapt* number of samples based on uncertainty:

$$\mathcal{Y}^{t+1} = \frac{1}{2\epsilon}\chi^2_{q^t-1,1-\delta} \simeq \frac{q^t-1}{2\epsilon}\left\{1 - \frac{2}{9(q^t-1)} + \sqrt{\frac{2}{9(q^t-1)}}z_{1-\delta}\right\}^3 \quad (4)$$

- Entropy in operator probability distribution:

$$E^t = -\sum_{j=0}^{N-1} p_j^t \cdot ln(p_j^t) \quad (5)$$

- *Termination condition:* $E^t - E^{t-1} \leq \text{threshold}$

Innovations in Sampling

- Sample without replacement: *stationary* system.
- *Adapt* number of samples based on uncertainty:

$$\mathcal{Y}^{t+1} = \frac{1}{2\epsilon}\chi^2_{q^t-1,1-\delta} \tag{4}$$
$$\simeq \frac{q^t-1}{2\epsilon}\left\{1 - \frac{2}{9(q^t-1)} + \sqrt{\frac{2}{9(q^t-1)}}z_{1-\delta}\right\}^3$$

- Entropy in operator probability distribution:

$$E^t = -\sum_{j=0}^{N-1} p_j^t \cdot ln(p_j^t) \tag{5}$$

- *Termination condition:* $E^t - E^{t-1} \leq \text{threshold}$

Outline

- **Introduction:**
  - Motivation.
  - Related Work.
- **Proposed Approach:**
  - Hypothesis.
  - Stochastic sampling.
  - Probabilistic formulation.
- **Experimental Results**:
  - Case studies.
  - Figures and tables.
  - Discussion.
- **Conclusions**:
  - Conclusions.
  - Further Reading.

Siemens Programs

Siemens Dataset:

Program	NLOC	NTC	NMG	NMS	NM
`printtokens`	343	4130	11741	2000	1551
`printtokens2`	355	4115	10266	2000	1942
`replace`	513	5542	23847	2000	1969
`schedule`	296	2650	4130	2000	1760
`schedule2`	263	2710	6552	2000	1497
`tcas`	137	1608	4935	4935	4935
`totinfo`	281	1052	8767	2000	1740

Table: Description of subject programs. NLOC: Net lines of code. NTC: Number of test cases. NMG: Number of mutants generated by all mutation operators. NMS: Number of randomly selected mutants. NM: Number of selected mutants that were non-equivalent.

Larger Programs

- Evaluated performance on `gzip` and `space`.
- Program `gzip`:
  - NLOC: 5680, 212 test cases.
  - 43 mutation operators, 28 based on sufficient set.
  - 317 non-equivalent mutants used.
  - *Study effect of sufficient set of mutation operators.*
- Program `space`:
  - NLOC: 5905, 13585 test cases.
  - 108 mutation operators.
  - 23708 non-equivalent mutants used.
  - *Study effect of larger set of test cases.*

Data Collection

- Proteum generates mutants with 108 mutant operators.
- Test suites of size $1 - 50$ by choosing randomly from the available test cases.
- Use test suites on the mutants to record mutants killed and left alive.
- *This provides the ground truth.*

- Compare sampling-based approach against ground truth.
- Compare two schemes for initial operator probabilities: *uniform* and *proportional*.

Measures Used

- Mutation Importance Measure: $IM = 1 - AM$

$$IM = \frac{\#alive\ mutants}{\#Total\ non - equivalent\ mutants} \qquad (6)$$

- Measures fraction of examined mutants left alive.

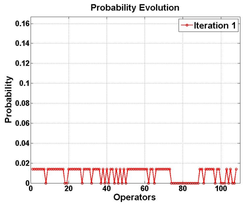
- Operator Overlap Measure:

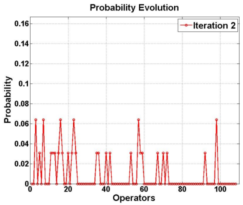
$$OpOverlap(\mathcal{P}, \mathcal{S}, T) = Overlap(Gt(\mathcal{P}, \mathcal{S}), Obs(\mathcal{P}, \mathcal{S})) \qquad (7)$$

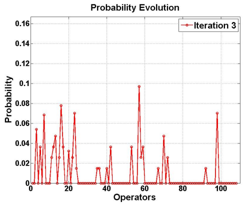
- Measures fraction of top mutants in ground truth list (*Gt*) that exist in the observed list (*Obs*).

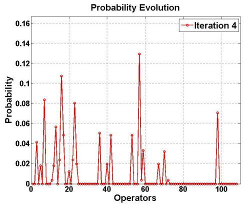
Probability Updates

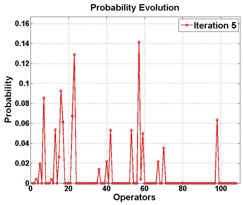
Sampling converges on important mutation operators.





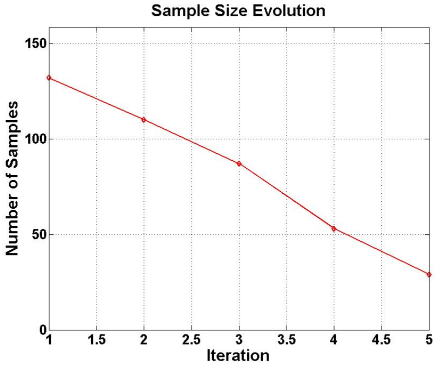






Samples Examined

- Entropy and adaptive sampling enable convergence while examining a small set of samples.



Operator Overlap Scores I

Siemens: Uniform Initial Probability

Programs	Operator Overlap			
	Max	Min	Average	Dynamic
`printtokens`	0.93	0.56	0.76 ± 0.07	0.9
`printtokens2`	0.92	0.48	0.69 ± 0.11	0.94
`replace`	1.0	0.80	0.87 ± 0.06	0.9
`schedule`	0.90	0.48	0.68 ± 0.1	0.9
`schedule2`	0.91	0.68	0.77 ± 0.04	0.89
`tcas`	1.0	0.74	0.85 ± 0.06	0.89
`totinfo`	1.0	0.51	0.69 ± 0.10	0.95

Table: Operator overlap with uniform initial operator probability assignment. Dynamic selection of “T” provides better results.

Operator Overlap Scores II

Siemens: Proportional Initial Probability

Programs	Operator Overlap			
	Max	Min	Average	Dynamic
`printtokens`	0.94	0.60	0.77 ± 0.09	0.92
`printtokens2`	0.97	0.47	0.66 ± 0.13	0.9
`replace`	1.0	0.65	0.86 ± 0.07	0.94
`schedule`	0.90	0.48	0.67 ± 0.11	0.98
`schedule2`	0.92	0.66	0.78 ± 0.05	0.91
`tcas`	0.98	0.85	0.91 ± 0.04	0.92
`totinfo`	0.96	0.51	0.74 ± 0.09	0.96

Table: Operator overlap with proportional initial operator probability distribution. Dynamic selection of “T” provides better results.

Operator Overlap Scores III

OpOverlap for `gzip` and `space`:

Programs	Operator Overlap			
	Max	Min	Average	Dynamic
Uniform				
`gzip`	0.98	0.95	0.96 ± 0.01	0.97
`space`	0.91	0.35	0.64 ± 0.17	0.81
Proportional				
`gzip`	0.98	0.95	0.96 ± 0.01	0.97
`space`	0.96	0.34	0.67 ± 0.19	0.87

Table: Operator overlap for `gzip` and `space` with uniform and proportional initial operator probability distributions.

Observations I

- The proposed approach is able to identify 90% of the important operators by examining $\leq$ 20% of the mutants over $3 - 4$ sampling iterations.
- There is a 6% increase in *IM*, i.e., *larger proportion of examined mutants remain unexposed*.
- Automatic tuning of parameters to match the specific program and test suite.
- Performance generalizes across different programs and test suite sizes.

Observations II

- Performance generalizes to program with sufficient set of mutants (`gzip`) and large number of test cases (`space`).
- Similar performance with different schemes of initial probability assignment.
- The operators identified as being important make logical sense—see paper for detailed results.
- Initial hypothesis justified!

Threats to Validity

- Small to medium-sized C programs.
- Do not investigate object-oriented programs.
- Random selection of mutants for each program.
- Correctness of the data collection process.

Outline

- **Introduction:**
  - Motivation.
  - Related Work.
- **Proposed Approach:**
  - Hypothesis.
  - Stochastic sampling.
  - Probabilistic formulation.
- **Experimental Results**:
  - Case studies.
  - Figures and tables.
  - Discussion.
- **Conclusions**:
  - Conclusions.
  - Further Reading.

Conclusions

- Mutation testing formulated as a importance sampling (i.e., probabilistic) problem.

- Adaptive sampling and information theoretic measures enable reliable and efficient mutation testing.
- Detects $\geq$ 90% of the truly important mutation operators by examining $\leq$ 20% of the available mutants.
- Generalizes to medium-sized programs.

- Sampling is well-suited for many other software testing applications!

Conclusions

- Mutation testing formulated as a importance sampling (i.e., probabilistic) problem.

- Adaptive sampling and information theoretic measures enable reliable and efficient mutation testing.
- Detects $\geq$ 90% of the truly important mutation operators by examining $\leq$ 20% of the available mutants.
- Generalizes to medium-sized programs.

- Sampling is well-suited for many other software testing applications!

Conclusions

- Mutation testing formulated as a importance sampling (i.e., probabilistic) problem.
- Adaptive sampling and information theoretic measures enable reliable and efficient mutation testing.
- Detects $\geq$ 90% of the truly important mutation operators by examining $\leq$ 20% of the available mutants.
- Generalizes to medium-sized programs.
- Sampling is well-suited for many other software testing applications!

Conclusions

- Mutation testing formulated as a importance sampling (i.e., probabilistic) problem.

- Adaptive sampling and information theoretic measures enable reliable and efficient mutation testing.
- Detects $\geq$ 90% of the truly important mutation operators by examining $\leq$ 20% of the available mutants.
- Generalizes to medium-sized programs.

- Sampling is well-suited for many other software testing applications!

Conclusions

- Mutation testing formulated as a importance sampling (i.e., probabilistic) problem.
- Adaptive sampling and information theoretic measures enable reliable and efficient mutation testing.
- Detects $\geq$ 90% of the truly important mutation operators by examining $\leq$ 20% of the available mutants.
- Generalizes to medium-sized programs.
- Sampling is well-suited for many other software testing applications!

Further Reading

- Mohan Sridharan and Akbar Siami Namin. *Bayesian Methods for Data Analysis in Software Engineering*. Tutorial at ICSE-2010.
- C. Bishop. *Pattern Recognition and Machine Learning*. Springer publishing house, 2007.
- S. Thrun and W. Burgard and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
- Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Second Edition, Wiley-Interscience, 2005.