Exploration / Exploitation Inference for Statistical Software Testing

Michèle Sebag

TAO : CNRS - INRIA - Université Paris-Sud Orsay

http://tao.lri.fr

Joint work with
Nicolas Baskiotis, Marie-Claude Gaudel, Sandrine Gouraud
Overview

- Software Testing
- Exploration/Exploitation for Statistical Software Testing
- ML for Computer Science
- Autonomic Computing
Software Testing

A key task

Bugs may kill (airplanes, shuttles, stock market,...)  
ST costs 50% percent of the development time

A challenging task

Pesticide Paradox
Every method you use to prevent or find bugs leaves a residue of subtler bugs against which those methods are ineffectual.

Complexity Barrier
Software complexity (and therefore that of bugs) grows to the limits of our ability to manage that complexity.

[Beizer 90]
Software Testing, Classification

By scope
unit testing, component testing, integration testing, system testing.

By life-cycle phase
requirements phase testing, design phase testing, program phase testing, evaluating test results, installation phase testing, acceptance testing, maintenance testing.

By purpose
correctness testing, performance testing, reliability testing, security testing.
Correctness Testing

Black-box  Functional testing
Given: I/O of the program and specifications
Method: partitioning input space, exploring boundary conditions.
Issues: combinatorial explosion; error in specifications (30%)

White-box  Structural testing
Given: the program + oracle
Method: generate test cases (input vectors)
Criteria: coverage wrt program (syntactic or intrusive)
Issues: combinatorial explosion; undecidability.

Annotated-box  Formal testing
Given program + properties (formulas)
Method: Prove that program satisfies properties
Issues: combinatorial explosion; undecidability.
Criteria

Find criteria related to correctness...

- Product
  Tested 80% of the lines of code

- Plan
  Run 80% of the test cases

- Results
  Discovered 417 bugs

- Efforts
  Worked 80h a week for 4 weeks.

Standard: coverage-based

- Percentage of the lines of code
- Percentage of the transitions
- Percentage of the paths with bounded length
Hybrid Statistical/Structural Approach

Principle

- Program $\equiv$ Finite State Automaton
- Path $\rightarrow$ constraint satisfaction pb
- CSP $\rightarrow$ Solution = value of input variables = test case exerting the program path

[Denise et al. 04]
Example

Code

```plaintext
read (x, y) 1
if (x < 0) 2
    then x := −x; y := 1/y; 3
p := 1; 4
while (x > 0) 5
do p := p * y; x := x − 1; 6
print p; 7
```

Path

\[ s = 1.2.4.5.7 \]

Test case

Solution: \( x = 0; \)
Hybrid Statistical/Structural Approach, 2

Program = FSA = \{\text{Nodes } \Sigma, \text{ Edges } \subset \Sigma^2 \}

Assumption: consider strings/paths with length \( \leq T \)

Approach: Uniform distribution in finite structured spaces

[Flajolet et al. 94]

Let

\( v_f \) (resp. \( v_s \)): accepting (resp. starting) node.

\( \text{suc}(v) \): set of nodes \( w \) such that \( v.w \) is an edge.

Define

\[ N(v, t) = \text{Number of paths } v \ldots v_f \text{ of length } t \]

Then:

\[ N(v, 1) = 1 \text{ iff } v_f \in \text{suc}(v) \]

\[ N(v, t + 1) = \sum_{w \in \text{suc}(v)} N(w, t) \]
Uniform sampling of bounded program paths

For $t = 1 \ldots T$

$$N(v, t) = \text{Number of paths } v \ldots v_f \text{ of length } t$$

**Uniform Sampling**

**Init:** $s[0] = v_s$

**For** $i = 1 \ldots T$

- Candidates = $\text{suc}(s[i-1]) = \{w_{i,1}, \ldots w_{i,K_i}\}$
- Select $w_{i,j}$ with probability $\propto N(w_{i,j}, T - i)$
- $s[i+1] = w_{i,j}$

**EndFor**

**Return** $s$
Hybrid Statistical/Structural Approach, 3

**Principle**

Init: Test set = \{\}

Repeat

- Generate program path $s$
- Transform $s$ into a constraint satisfaction pb $\text{CSP}_s$
- Call Oracle (constraint solver)
- If $\text{CSP}_s$ satisfiable
  - Find Solution = test case
  - Test set $\leftarrow$ Solution

// Else $s$ unfeasible path

Until stop criterion

**Criterion**

Pr (feasible path exerted by Test set).
Discussion

PROs
Uniform distribution.
No redundancy: each test case exerts a different program path

CONs
Mild: Undecidability
set a time limit on constraint solver
SEVERE: Syntax is a very poor approximation of semantics
⇒ huge fraction of unfeasible paths
⇒ modify the program by hand
At last, ML comes into play!
1st: Discriminant/Active learning

Given \( \left\{ \Sigma, E \right\} \).
\[
\mathcal{L} = \{(x_i, y_i), x_i \in \Sigma^T, y_i = \pm 1\}
\]

Find: \( \hat{y} \) estimating whether a program path is feasible

Wanted
\[
\left\{ \begin{array}{l}
\text{Now: Save the oracle cost} \\
\text{Later: Facilitate the generation of feasible paths}
\end{array} \right.
\]

ML Settings
- strings — RPNI, RedBlue
- propositionalisation — C4.5, Ripper

Fails!
- Insufficiently many positive examples

Active learning? [Dasgupta 05]
2nd: Generative learning

Given
\[
\begin{align*}
\text{FSA: } & \{\Sigma, E\}. \\
\mathcal{L} & = \{(x_i, y_i), x_i \in \Sigma^T, y_i = \pm 1\}
\end{align*}
\]

Find  The distribution \( \mathcal{D} \) of feasible paths

Principle:
1. Use \( \mathcal{D}_t \) to generate \( x_t \)
2. Oracle: compute \( y_t \) \hspace{1cm} \text{feasible/unfeasible}
3. Update \( \mathcal{D}_t \rightarrow \mathcal{D}_{t+1} \).
Position of the problem

Goal
Find the maximal number of (distinct) feasible paths

Wrt online learning
The criterion is not to minimize the regret

Wrt reinforcement learning or estimation of distribution algorithms
The goal is dynamic: after a feasible path has been found it is not new anymore...
Domain knowledge and search space

What makes a path unfeasible?

Limits on Loops
If there are 17 or 19 uranium beams to be examined
the number of times in the loop is 17 or 19.

Violated dependencies
if \( x \)  
then \( y := \ldots \)  
else \( z := \ldots \)  
[...]  
if \( x \)  
then \( u := \ldots \)  
else \( w := \ldots \)  

\( s = \ldots 12457\ldots \) is unfeasible.

Others
The last time a loop occurs, the closing instruction is executed.

Non Markovian problem
Representation: Parikh map

**Parikh map**: each symbol $u$ in $\Sigma \rightarrow$ integer attribute

$$a_u : X \leftrightarrow \mathbb{N}$$

$$a_u(s) = \text{number of occurrences of } u \text{ in } s$$

**Extended Parikh map**: each $(u, k)$ in $\Sigma \times \mathbb{N} \rightarrow$ categorical attribute

$$a_{u,k} : X \leftrightarrow \Sigma$$

$$a_{u,k}(s) = \text{symbol successor of the } k\text{-th occurrence of } u$$

Captures target concepts
- loops
- dependencies (XOR)
- closing instructions (reverse order on paths)
Distribution search space

Parikh map description

\[ s = v \, w \, v \, w \, v \, x \, y \, w \rightarrow \]

\[ a_{v,1} = w, \ a_{v,2} = w, \ a_{v,3} = x \]
\[ a_{w,1} = v, \ a_{w,2} = v, \ a_{w,3} = \emptyset \]
\[ a_{x,1} = y, \ a_{x,2} = \emptyset, \ a_{x,3} = \emptyset \]
\[ a_{y,1} = w, \ a_{y,2} = \emptyset, \ a_{y,3} = \emptyset \]

Distributions: \( \Sigma \times \Sigma \times N \rightarrow [0, 1] \)

- \( n(v, w, i) \): number of paths st \( a_{v,i} = w \)
- \( f(v, w, i) \): number of feasible paths st \( a_{v,i} = w \)
- \( \mu(v, w, i) = f(v, w, i)/n(v, w, i) \)
EXIST : Exploitation / Exploration Inference for Statistical Testing

Modules

- Initialise the current distribution
- Select the current symbol
- Update the distribution

Criteria

number of feasible NEW paths [and their diversity]
Selection Module

Given

- the current string $v$
- the last symbol $v$
- with $i$-th occurrences in $s$
- frequency of feasible paths $s'$ st $a_{v,i}(s') = w$

Select a node among $suc(v)$

- **Greedy**: $\text{argmax}_w \mu(v, w, i)$

- **BandiST**: $\text{argmax}_w \mu(v, w, i) + \sqrt{\frac{2 \log n(v,*.i)}{n(v, w, i)}}$

  [Auer, Cesa-Bianchi, Fischer 02]

- **Roulette Wheel**: select $w$ proportionally to $\mu(v, w, i)$
Update Module

Global update

Increment \( n(v, w, i) \)
Increment \( f(v, w, i) \) iff \( s \) is feasible and NEW

Local update: look ahead

\[ n(v, w, i) \rightarrow n_s(v, w, i) = \text{number of paths } s' \]
\[ \text{such that } a_{v,i}(s') = w \text{ and } a_{w}(s') \geq a_w(s) \]

Same for \( f(v, w, i) \) and \( \mu(v, w, i) \)
Initialisation Module

Straightforward option
Set \( n(v, w, i) \) and \( f(v, w, i) \) to the initial number of feasible paths.

... fails same problem as finding XORs with decision trees...

Example:
\[
([a_{v,1} = w] \land [a_{v,2} = w]) \lor ([a_{v,1} = z] \land [a_{v,2} = z])
\]

\(vwvwv\ldots\text{feasible} \quad \text{but} \quad vwuzv\ldots\text{unfeasible}\)

\(vzvzv\ldots\text{feasible} \quad \text{but} \quad vzvwv\ldots\text{unfeasible}\)

\(\rightarrow f(v, w, 1) \) and \( f(v, w, 2) \) not informative...
Seeded Initialization

Principle
Extract a subset $E$ of positive paths in the same conjunctive concept
Criterion: the least general generalisation of $E$ must be correct
not covering negative examples
Seeded Initialisation

Seeded Initialization
Randomly order the positive examples \( \{x_1, \ldots, x_n\} \)
Init: \( E' = \{e_1\}, \quad tc_1 = e_1 \);
For \( i = 2 \ldots n \)
\[ tc = lgg(tc_{i-1}, e_i) \]
If \( tc \) is correct, \( tc_i = tc \) and \( E' = E' \cup \{e_i\} \)
Else \( tc_i = tc_{i-1} \)

Uniform Seeded
Same except that the initial order favors the less previously selected examples

Fake Seeded
As in Seeded initialization, but without the correctness test
Summary of EXIST

Init Module

- Global, Seeded, Uniform Seeded, Fake Seeded

Selection Module

- Greedy, BandiST, Roulette Wheel

Update Module

- Global, Local, Restart
Experimental Validation

Real-world problem: FCT4
13 nodes and 26 edges (after pruning)
Length 120 → \( Pr(s \text{ feasible}) = 10^{-5} \).
target concept: loop and XORs.

Artificial problems randomly generated
nodes in [10,20]; length in [60,120]
target concept: loops and XORs
Feasibility:

- cat. I: \( 10^{-3} \leq Pr(s \text{ feasible}) \leq 10^{-2} \)
- cat. II: \( 10^{-5} \leq Pr(s \text{ feasible}) \leq 10^{-3} \)
- cat. III: \( 10^{-15} \leq Pr(s \text{ feasible}) \leq 10^{-12} \)
Experimental setting and goal

Goal
influence of initial size/balance of examples.

<table>
<thead>
<tr>
<th>Training sets:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>feasible</td>
<td>50</td>
<td>200</td>
<td>1000</td>
<td>50</td>
</tr>
<tr>
<td>unfeasible</td>
<td>50</td>
<td>200</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Assessment
For every option and problem,
1 run: 10,000 paths are generated \(\#\{\text{new feasible paths}\}\) recorded averaged on 10 runs.
Best options = BandiST, Seeded Greedy, and Roulette Wheel
**Best options** = Seeded Greedy with restart
Best options = Seeded Greedy with restart
Remark: Seeded $>>$ Fake seeded when nb examples $\nearrow$. 
Discussion

It worked!

- Extended Parikh Map: a flexible and compact representation
- Seeded initialization: getting rid of non-Markovian issues
- Runtime $< 10\text{min}$

Next

- Convergence
- Diversity study
- Adapt EXIST for other coverage-based criteria
- Benchmarks for software testing
Related Works

- Ernst et al. 1999: Program invariants are learned from traces
- Brehelin et al. 2001: HMM are used to generalize test sequences for PLA.
- Vardan et al. 2004: Grammatical Inference is used to characterize paths relevant to constraint checking
- Zheng et al. 2003-6: Use traces to identify bugs (intrusive testing)
- Xiao et al. 05: Active learning for game player modeling (black box)
Overview

- Software Testing
- Exploration/Exploitation for Statistical Software Testing
- ML for Computer Science
- Autonomic Computing
Computers and networks govern communication and information.

Complex systems
Large-scale, heterogeneous components, dynamic interactions.

Number of skilled administrators... doesn’t scale up.

Need for Autonomous Systems

First step Self-Aware Systems

How? ML
"Considering current technologies, we expect that the total number of device administrators will exceed 220 millions by 2010."

-Gartner 6/2001
A case study
(upcoming EGEE-Pascal Challenge)

EGEE, Enabling Grids for e-Science in Europe

- Infrastructure project started in 2001
- 80 partners, 30,000 CPUs all over the world
- Web: www.eu-egee.org
Goal: Grid modelling

**Heterogeneous systems**: processors, storage, network, services. State can at most be estimated

**Mutualisation paradigm**: load depends on collective behavior. ... must be estimated on the fly

**Needed**: a grid model, in order to
- Control and maintain the system
- Predict the application performances
- Optimize the system
detect ill-configured units
dimension the capacities for jobs
refine the scheduler
Modelling the grid: an ML problem

Input data
Traces of the jobs:
800 Ko per job, including specifications and all events
some hundred thousands jobs per trace
spatio-temporal (redundant) structure

Goals
Classification: jobs are done, aborted, or lost
Early detection: predict as early as possible
Clustering: provide the user with model chunks and/or outliers
Call to Arms

ML for Autonomic Computing

- The need
- The data
- The expertise

The big question mark: Learning or Optimization?