Evolutionary Methods for State-based Testing

PhD Student Raluca Lefticaru
Supervised by Florentin Ipate

University of Piteşti, Romania
Department of Computer Science
Outline

- Motivation
- Search-based software engineering
- Metaheuristic search techniques
- Test data generation for state-based testing
- Experiments and results obtained
- Conclusions
- Questions
Motivation

- A lot of research has been done in the field of state-based testing:
  - test selection methods for FSMs – the **W-method**
  - **coverage criteria** for state machines diagrams (like all transitions, full predicate, transition pair, complete sequence, disjunct coverage)

- The automatic generation of test cases from extended state machines (state machines diagrams, X-machines, etc.) is not straightforward.
- For example: *which input values to choose for a sequence of methods calls, representing a path in a state machine diagram?*
- Idea: *Why not to use the power of evolutionary algorithms or other metaheuristics to solve state-based testing problems?*
- This answer is given by new field of **Search-based Software Engineering**
Search-based Software Engineering (SBSE)

Search-based software engineering (SBSE) is a relatively new approach to transform the software engineering problems into optimization problems, which can be further solved by applying metaheuristic search techniques.

The term of SBSE was first used by Harman and Jones (2001), although there are some previous papers on this topic.

Repository of publications on SBSE: http://www.sebase.org/sbse/publications/repository.html

In 2008 there were published more than 100 papers on SBSE.

Important events:
- The International Workshop on Search-Based Software Testing, held in conjunction with ICST: 2008 Lillehammer, 2009 Denver
- The International Symposium on Search Based Software Engineering, 2009 Windsor
Search-based software engineering (SBSE)

- **Main idea**: Transform the software engineering problems into *optimization problems*, which can be further solved by applying metaheuristics.

- **Search techniques**: *genetic algorithms*, hill climbing, alternating variable method, simulated annealing, genetic programming, particle swarm optimization, artificial immune systems, tabu search etc.

- **Applications**: *software testing*, requirements engineering, automated maintenance, service-oriented software engineering, compiler optimization, quality assessment, project planning and cost estimation.
Number of publications in SBSE, 1976 - 2009

http://www.sebase.org/sbse/publications/
Ratio of SE research fields involved in SBSE

http://www.sebase.org/sbse/publications/
Search-based Software Testing (SBST)

- Characterized by the usage of guided search techniques for test generation

- The test aim (cover all branches, obtain the WCET) is first transformed into an optimization problem with respect to some fitness (cost or objective) function.

- **Search space** = the input domain of the test object (program, function).

- The search spaces obtained are usually complex, discontinuous, and non-linear, due to the non-linearity of software

- Therefore metaheuristic search methods are recommended.
Search-based Software Testing (SBST)

- **Structural testing:** automatically generate input data for programs written mainly in a procedural paradigm.
  - Program = directed graph
  - Cover the desired graph elements (nodes, branches or paths)

- **Functional testing:** from Z specification, conformance testing, automatic parking system
  - Testing of grey-box properties, for example safety constraints
  - Testing non-functional properties, such as worst-case execution time
Metaheuristic search techniques

- **Simulated annealing (SA):** a random generated "neighbour" replaces the current solution if it has a better objective value; otherwise, with the probability \( p = \exp(-\delta / t) \).

- **Genetic algorithms (GAs):** a class of evolutionary algorithms, that use selection, recombination (crossover) and mutation, applied on a population of potential solutions, called chromosomes (or individuals).

- **Particle swarm optimization (PS):** a population of random solutions, called particles, which maintain their current position, velocity and best position explored so far, fly through the problem space by following the current optimum particles.
Test data generation for state-based testing

- Given: some paths in the state machine (obtained according to a certain coverage criteria)

- Use metaheuristic search techniques to find for each method sequence the input values for the parameters, which satisfy the corresponding guards (pre-conditions)

- Questions:
  - Encoding: $x = (x_1, x_2, \ldots, x_n)$
  - Fitness function
  - Search technique
Fitness function calculation

\[
\text{fitness} = \text{approach\_level} + \text{normalized\_branch\_level}
\]

\[
\text{approach\_level} \in \{0,1,\ldots,m-1\}
\]

\[
\text{normalized\_branch\_level} \in [0,1]
\]
Tracey’s objective functions

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Objective function ( obj )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = b )</td>
<td>if ( \text{abs}(a - b) = 0 ) then 0 else ( \text{abs}(a - b) + K )</td>
</tr>
<tr>
<td>( a \neq b )</td>
<td>if ( \text{abs}(a - b) \neq 0 ) then 0 else ( K )</td>
</tr>
<tr>
<td>( a &lt; b )</td>
<td>if ( a - b &lt; 0 ) then 0 else ( (a - b) + K )</td>
</tr>
<tr>
<td>( a \leq b )</td>
<td>if ( a - b \leq 0 ) then 0 else ( (a - b) + K )</td>
</tr>
<tr>
<td>( a &gt; b )</td>
<td>if ( b - a &lt; 0 ) then 0 else ( (b - a) + K )</td>
</tr>
<tr>
<td>( a \geq b )</td>
<td>if ( b - a \leq 0 ) then 0 else ( (b - a) + K )</td>
</tr>
<tr>
<td>\text{Boolean}</td>
<td>if TRUE then 0 else ( K )</td>
</tr>
<tr>
<td>( a \land b )</td>
<td>( \text{obj}(a) + \text{obj}(b) )</td>
</tr>
<tr>
<td>( a \lor b )</td>
<td>( \text{min}(\text{obj}(a), \text{obj}(b)) )</td>
</tr>
</tbody>
</table>
State machine diagram of a *Book* class
Fitness function landscape

$x_1, x_2 \in [-50,50]$

$x_1 > 0$ and $x_2 = x_1$

Tracey's constant $K = 1$

$norm : [0,101] \rightarrow [0,1)$

$norm(d) = 1 - 1.05^{-d}$
Experiments employing genetic algorithms

- Generating numerical input values for given paths in the state machine, with the *approach level + normalized branch level* fitness function
- GAs had a good convergence rate (max. allowed generations 100)
- Heuristic real value crossover inspired from Michalewicz: $z_i = \alpha \cdot (x_i - y_i) + x_i$, $0 < \alpha < 1$, where $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, $x$ fitter than $y$.

**Conformance testing**: experiments that used mutation testing and a fitness function of form *pre-condition* ∧ ¬ *post-condition*.
- Discovered 72-82% of the mutants in 100 generations. After increasing the maximum allowed evolutions to 200 the rest of unequivalent mutants were detected also.

Average number of generations, related to: solutions number / possible solutions number
Average number of failures, related to: solutions number / possible solutions number
Experiments employing GAs, PSO, SA

- Aim: generate input values for different state machine paths
- Search techniques: GAs, SA, PSO
- The corresponding fitness landscapes have different complexity: simple (only one minimum), complex (many local minima).

R. Lefticaru, F. Ipate, "Functional Search-based Testing from State Machines", ICST2008
GAs (with heuristic crossover) achieved better results (9 out of 9 cases) than PSO and SA.
Complex objects (more local minima)

For more complex landscapes, having more local minima, PSO outperformed GAs and the difference was statistically significant for 17 out of 21 test objects (confidence 95%).

**Success rates for complex objects**

- PS
- GA
Measures to Characterize Search Problems

- Used to characterize the search landscapes, to predict or explain the behavior of search algorithms, to guide the implementation choices to be made.
- Intuitively, some fitness functions might present plateaux along which they provide no guidance.

**Diameter:** maximal distance between two points in S, in terms of successive applications of a neighbourhood operator N.

**Autocorrelation:** measures the variation of fitness for points that are at the same distance. It characterizes the ruggedness of a landscape: *smooth* (the neighbours have nearly the same fitness value) or *rugged* (the fitness values are dissimilar).

**Fitness Distance Correlation (FDC):** joint variation of distances and costs.
Analysing the fitness landscape

- The general fitness functions approach level + normalized branch level \((al + nbl)\) may produce results comparable to those produced by fitness functions, designed especially for a particular situation.

- The best results were obtained when using the \(al + nbl\) functions with GA and PSO.


*R. Lefticaru, F. Ipate, "A Comparative Landscape Analysis of Fitness Functions for Search-based Testing", SYNASC 2008*
Fitness-distance scatterplots

- **cost2**: Fitness vs. Distance plot showing a linear relationship.
- **cost3**: Fitness vs. Distance plot showing a linear relationship.
- **cost4**: Fitness vs. Distance plot showing a linear relationship.
- **cost_dym**: Fitness vs. Distance plot showing a non-linear relationship.
- **cost_dmy**: Fitness vs. Distance plot showing a non-linear relationship.
- **cost_ymd**: Fitness vs. Distance plot showing a non-linear relationship.
The test data obtained with the $al+nbl$ functions can cover difficult paths in the machine.

A slightly different design ($pre-condition \land \neg post-condition$) of the fitness function can be used for specification conformance testing.

For more complex landscapes, having more local minima, PSO outperformed GAs.

For simpler function, having only one minimum GAs achieved better results.

Hybridizing SA or GAs with local search techniques improved their effectiveness.

The general fitness functions $al + nbl$ may produce results comparable to those produced by fitness functions, designed especially for a particular situation.
Future work

- Analyzing, on a larger benchmark of classes, the effectiveness and the efficiency of several search techniques.
- Finding categories of problems for which certain search algorithms achieve better results.
- Extending the strategy presented for method parameters with more complex types and multi-class testing.
- Analyzing other variants of fitness functions.
- Derivation of the fitness function from hierarchical and concurrent state machine diagrams.
Question & Answers

Thank you for your attention!