When Search Based Software Engineering Meets Membrane Computing

Raluca Lefticaru¹,², Ana Turlea², Florentin Ipate², Marian Gheorghe¹

¹University of Bradford, UK
²University of Bucharest, Romania
Overview

- Brief introduction to SBSE & SBST
- Current state and future of SBST
  - Case studies, challenges and SBST tools
  - SBSE influential papers
- Possible applications to Membrane Computing
  - Solution representation
  - Fitness functions ideas
Search-Based Software Engineering (SBSE)

- Software engineering problems → **optimization** problems + metaheuristic search techniques (GAs, PSO, SA)
- Term of **SBSE**: first coined by Harman and Jones (2001).
- **Applications**: software testing, requirements engineering, automated maintenance, service-oriented SE, compiler optimization, quality assessment, project planning and cost estimation.
- Repository of publications on SBSE: [http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/](http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/)
- More than 1700 papers on SBSE

Number of Publications in SBSE per year (1976 - early 2012)

© Image source
http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/
Ratio of SBSE Publications Number in the World Countries

© Image source
http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/
Ratio of Software Engineering Research Fields that Involved SBSE

1976-2010 Percentage of Paper Number

- Testing and Debugging: 52%
- Management: 10%
- Distribution, Maintenance, and Enhancement: 9%
- Design Tools and Techniques: 9%
- Requirements/Specifications: 5%
- General Aspects: 5%
- Software/Program Verification: 3%
- Others: 7%

© Image source
http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/
Important SBSE Events

- International Workshop The Search-Based Software Testing - SBST: [http://www.searchbasedsoftwaretesting.org](http://www.searchbasedsoftwaretesting.org)
SBST Methodology

- **Search-Based Software Testing (SBST)** is an automated search of a potentially large input space, guided by a problem-specific **fitness function**.

- **Search space = ?**
  - Depending on the problem, the input parameters of the program. Codification?

- **Fitness function = ?**
  - The fitness function guides the search to the **test goal**
  - It *scores* different inputs to the system according to the test goal

- **Which search algorithms to use?**
  - Global, local, hybrid?
  - GA, EGA, SA, ESA, ACO, GP, PSO?
Metaheuristic Search Techniques

- **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*).

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

- **Particle swarm optimization (PSO):** 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.

- **The Alternative Variable Method (AVM):** it optimizes each input variable locally, by taking each variable and taking exploratory moves, adding +1 or -1. If the exploratory move improves the fitness function, then the search continues on that direction until there is no more improvement.
Examples of Fitness Functions in SBST

- Worst case execution time
- Worst case scenario - parking example
- Structural testing: approach level + normalised branch level
- Functional testing: conformance testing - precondition and negated postcondition
Evolutionary Testing with Genetic Algorithms

Insertion → Mutation → Crossover → Selection → Fitness Evaluation → Execution → Monitoring → Test cases → End?

© Phil McMinn: Search-Based Testing. ISSTA 2010
Example of Fitness Function for Structural Testing
## Branch Distance

Tracey’s objective functions for relational predicates and logical connectives

<table>
<thead>
<tr>
<th>Relational predicate or logical connective</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean</th>
<th>$obj(a) + obj(b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \land b$</td>
<td>$\text{min}(obj(a), obj(b))$</td>
</tr>
<tr>
<td>$a \lor b$</td>
<td>$\text{obj}((a \land \neg b) \lor (\neg a \land b))$</td>
</tr>
<tr>
<td>$\neg a$</td>
<td>Negation is moved inwards and propagated over $a$</td>
</tr>
</tbody>
</table>
Fitness = Approach Level + Normalised Branch Distance

```c
void f1(int a, int b, int c, int d)
{
    if (a > b)
    {
        if (b > c)
        {
            if (c > d)
            {
                // target
            }
        }
    }...
```

normalised branch distance between 0 and 1 indicates how close approach level is to being penetrated


M. Harman. The Current State and Future of Search Based Software Engineering. FOSE 2007

P. McMinn. Search-Based Software Testing: Past, Present and Future. ICST Workshops 2011

Tutorials, Useful Presentations


- Phil McMinn. *Search-Based Testing*. ISSTA 2010


SBST Tools

**SBST Contest:** Java Unit Testing at the Class Level


- **EvoSuite:** won the SBST 2017 tool competition

- **JTExpert:** second place in SBST 2017 tool competition
  [https://sites.google.com/site/saktiabdel/JTExpert](https://sites.google.com/site/saktiabdel/JTExpert)

- **jMetal:** Metaheuristic Algorithms in Java

- **T3:** budget-aware random testing tool
  [http://www.qatestingtools.com/testing-tool/T3](http://www.qatestingtools.com/testing-tool/T3)
Challenges and Tool Competitions

- G. Fraser, A. Arcuri. 1600 faults in 100 projects: automatically finding faults while achieving high coverage with EvoSuite. Empirical Software Engineering, 2015


- I. S. W. B. Prasetya. Budget-aware random testing with T3: benchmarking at the SBST2016 testing tool contest. SBST@ICSE 2016

- A. Sakti, G. Pesant, Y.-G. Guéhéneuc. JTeXpert at the SBST 2017 Tool Competition. SBST@ICSE 2017


Case Studies: Automotive, CPS, Mobile Testing

- J. Wegener, O. Bühler. Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System. GECCO 2004
- K. Mao, M. Harman, Y. Jia. Crowd intelligence enhances automated mobile testing. ASE 2017
P Systems Testing
Motivation

- **Why P system testing?**
- Membrane computing: very fast growing field
- Rapid development of many tools and P system simulators
- This issue raises the problem of testing all these implementations of P systems.
- The models are complex: non-deterministic, parallel, can have polarizations (charges), transformation-communication rules, membrane creation/division etc.
P Systems Testing
Previous approaches

- P system **coverage criteria:** RC, RTC, CDRC, CDRTC
- Finite state based testing of P systems
- Mutation-based testing of P systems (Kripe structure)
- Testing non-deterministic stream X-machine models and P system
- Assessing the fault-detection efficiency of previous techniques - **mutation testing**!
- Automating P system test generation
- Test Generation for P Systems using **model checking**
P System Testing Papers (Selection)

- F. Ipate, M. Gheorghe. Finite state based testing of P systems. Natural Computing, 2009
When SBST meets Membrane Computing

- Testing P systems approaches
  - Grammar inspired
  - FSM, X-machine based
  - Mutation testing
  - Model checking based automation

- What about search based testing?
  - Input values for programs → initial multisets (configuration)
  - Search space? Representation
  - Encoding - integer, real? (x1, x2, ...)
  - Fitness function - testing goal!
  - Search technique? GAs, PSO?
Search-based Approach

- Fitness function
- Search space
- Encoding
- Search algorithm
- Test execution
- Instrumentation
Fitness Functions Ideas

- **Worst-case execution time** - maximization problem.

- Testing specification conformance: find **failures** in the implementation

- **Key ingredients:** preconditions (valid inputs - multisets), postconditions (expected result, properties which hold for certain configurations).

- **Failure:** an initial configuration is discovered, that satisfies the preconditions, but for which the results or the intermediary configurations violate the post-conditions.

- **Fitness function:** *closeness* of the test data to uncovering such a situation, and metaheuristic search techniques are then employed to seek failures in the implementation.

- **Fitness:** \( \text{Obj} \ (\text{Precondition AND } \neg \text{Postcondition}) \)
Thank You!