Scalable Software Testing and Verification Through Heuristic Search and Optimization: Experiences and Lessons Learned

Lionel Briand

Interdisciplinary Centre for ICT Security, Reliability, and Trust (SnT) University of Luxembourg, Luxembourg

QRS, Vancouver, August 3, 2015
Acknowledgements

PhD. Students:
• Vahid Garousi
• Marwa Shousha
• Zohaib Iqbal
• Reza Matinnejad
• Stefano Di Alesio

Scientists:
• Shiva Nejati
• Andrea Arcuri
• Yvan Labiche
• Arnaud Gotlieb
Scalable Software Testing and Verification Through Heuristic Search and Optimization
Verification, Testing

- The term “verification” is used in its wider sense: Defect detection.

- Testing is, in practice, the most common verification technique.

- Testing is about systematically, and preferably automatically, exercise a system such as to maximize chances of uncovering (important) latent faults within time constraints.

- Other forms of verifications are important too (e.g., design time, run-time), but much less present in practice.

- Decades of research have not yet significantly and widely impacted software verification practice.
Scalable? Applicable?

- **Scalable**: Can a technology be applied on large artifacts (e.g., models, data sets, input spaces) and still provide useful support within reasonable effort, CPU and memory resources?

- **Applicable**: Can a technology be efficiently and effectively applied by engineers in realistic conditions?
  - realistic ≠ universal
“A metaheuristic is a heuristic method for solving a very general class of computational problems by combining user given black-box procedures — usually heuristics themselves — in a hopefully efficient way.” (Wikipedia)

- Hill climbing, Tabu search, Simulated Annealing, Genetic algorithms, Ant colony optimisation ….

- Our research is agnostic to any specific technology but is driven by problems – the use of metaheuristics is however a recurring pattern. Why?
Talk Outline

• Selected project examples, with industry collaborations

• Similarities and patterns

• Lessons learned
Testing Software Controllers

References:

Electronic Control Units (ECUs)

Comfort and variety

More functions

Safety and reliability

Faster time-to-market

Greenhouse gas emission laws

Less fuel consumption
Dynamic Continuous Controllers
A Taxonomy of Automotive Functions

Computation
- Transforming
  - unit convertors
- Calculating
  - calculating positions, duty cycles, etc

Controlling
- State-Based
  - State machine controllers
- Continuous
  - Closed-loop controllers (PID)

Different testing strategies are required for different types of functions
Development Process

Model-in-the-Loop Stage
- Simulink Modeling
- MIL Testing

Software-in-the-Loop Stage
- Generic Functional Model
- Code Generation and Integration
- SIL Testing

Hardware-in-the-Loop Stage
- Software Running on ECU
- Software Release
- HIL Testing
MATLAB/Simulink model

Fibonacci sequence: 1,1,2,3,5,8,13,21,…
Controller Input and Output at MIL

Desired value + Error Controller (SUT) Plant Model System output

Actual value

Test Input

Test Output

Initial Desired Value

Final Desired Value

Initial Desired Value

Final Desired Value

T/2 T T/2 T

time

time

Desired Value

Actual Value
Controllers at MIL

Inputs: Time-dependent variables

Configuration Parameters
Requirements and Test Objectives

- Desired Value (input)
- Actual Value (output)
- Initial Desired (ID)
- Final Desired (FD)
- Responsiveness
- Smoothness
- Stability

Diagram showing the relationship between desired and actual values over time.
Test Strategy: A Search-Based Approach

- Continuous behavior
- Controller’s behavior can be complex
- Meta-heuristic search in (large) input space: Finding worst case inputs
- Possible because of automated oracle (feedback loop)
- Different worst cases for different requirements
- Worst cases may or may not violate requirements
Smoothness Objective Functions: $O_{\text{Smoothness}}$

$O_{\text{Smoothness}}(\text{Test Case A}) > O_{\text{Smoothness}}(\text{Test Case B})$

We want to find test scenarios which maximize $O_{\text{Smoothness}}$. 
Search Elements

- **Search Space:**
  - Initial and desired values, configuration parameters

- **Search Technique:**
  - (1+1) EA, variants of hill climbing, GAs …

- **Search Objective:**
  - Objective/fitness function for each requirement

- **Evaluation of Solutions**
  - Simulation of Simulink model => fitness computation

- **Result:**
  - Worst case scenarios or values to the input variables that (are more likely to) break the requirement at MiL level
  - Stress test cases based on actual hardware (HiL)
Solution Overview (Simplified Version)

Objective Functions based on Requirements + Controller-plant model

1. Exploration

Heat Map Diagram

Domain Expert

List of Critical Regions

2. Single-State Search

Worst-Case Scenarios

Graph Builder

Final vs. Initial

Smoothness

Initial Desired

Final Desired
Automotive Example

- **Supercharger bypass flap controller**
  - Flap position is bounded within [0..1]
  - Implemented in MATLAB/Simulink
  - 34 sub-components decomposed into 6 abstraction levels
  - The simulation time $T=2$ seconds

Flap position = 0 (open)  
Flap position = 1 (closed)
Finding Seeded Faults

Inject Fault

Figure 1
Analysis – Fitness increase over iterations
Analysis II – Search over different regions

<table>
<thead>
<tr>
<th>Number of Iterations</th>
<th>Average</th>
<th>(1+1) EA Distribution</th>
<th>Random Search Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.315</td>
<td>0.316</td>
<td>0.317</td>
</tr>
<tr>
<td></td>
<td>0.319</td>
<td>0.321</td>
<td>0.323</td>
</tr>
<tr>
<td></td>
<td>0.324</td>
<td>0.326</td>
<td>0.327</td>
</tr>
<tr>
<td></td>
<td>0.328</td>
<td>0.329</td>
<td>0.330</td>
</tr>
</tbody>
</table>

Random Search

(1+1) EA
Conclusions

- We found much worse scenarios during MiL testing than our partner had found so far, and much worse than random search (baseline)
- These scenarios are also run at the HiL level, where testing is much more expensive: MiL results -> test selection for HiL
- But further research was needed:
  - Simulations are expensive
  - Configuration parameters
  - Dynamically adjust search algorithms in different subregions (exploratory <-> exploitative)
Testing in the Configuration Space

- MIL testing for all feasible configurations
- The search space is much larger
- The search is much slower (Simulations of Simulink models are expensive)
- Results are harder to visualize
- Not all configuration parameters matter for all objective functions
Modified Process and Technology

1. Exploration with Dimensionality Reduction

- Visualization of 8-dimensional space using regression trees

2. Search with Surrogate Modeling

- Dimensionality reduction to identify the significant variables

- Surrogate modeling to predict the objective function and speed up the search

List of Critical Partitions

Domain Expert

Objective Functions + Controller Model (Simulink)

Worst-Case Scenarios
A Taxonomy of Automotive Functions

Computation

Transforming
- unit convertors

Calculating
- calculating positions, duty cycles, etc

Controlling

State-Based
- State machine controllers

Continuous
- Closed-loop controllers (PID)

Different testing strategies are required for different types of functions
Differences with Close-Loop Controllers

- Mixed discrete-continuous behavior: Simulink stateflows
- Much quicker simulation time
- No feedback loop -> no automated oracle
- The main testing cost is the manual analysis of output signals
- Goal: Minimize test suites
- Challenge: Test selection
- Entirely different approach to testing
Selection Strategies

- Adaptive Random Selection
- White-box Structural Coverage
  - State Coverage
  - Transition Coverage
- Output Diversity
- Failure-Based Selection Criteria (search)
  - Domain specific failure patterns
  - Output Stability
  - Output Continuity
Stability
Continuity
Minimizing CPU Shortage Risks During Integration

References:

Automotive: Distributed Development
Software Integration
Stakeholders

Car Makers

- Develop software optimized for their specific hardware
- Provide part suppliers with runnables (exe)

Part Suppliers

- Integrate car makers software with their own platform
- Deploy final software on ECUs and send them to car makers
Different Objectives

Car Makers

- Objective: Effective execution and synchronization of runnables
- Some runnables should execute simultaneously or in a certain order

Part Suppliers

- Objective: Effective usage of CPU time
- Max CPU time used by all the runnables should remain as low as possible over time
An overview of an integration process in the automotive domain

OEM

AUTOSAR Models

AUTOSAR Models

Glue

sw runnables

sw runnables

DELPHI

Automotive Systems
CPU time shortage

- **Static cyclic scheduling:** predictable, analyzable
- **Challenge**
  - Many OS tasks and their many runnables run within a limited available CPU time
    - The execution time of the runnables may exceed their time slot
- **Our goal**
  - Reducing the maximum CPU time used per time slot to be able to
    - Minimize the hardware cost
    - Reduce the probability of overloading the CPU in practice
    - Enable addition of new functions incrementally
Using runnable offsets (delay times)

Inserting runnables’ offsets

Offsets have to be chosen such that
the maximum CPU usage per time slot is minimized, and further,
the runnables respect their period
the runnables respect their time slot
the runnables satisfy their synchronization constraints
Search algorithms

- The objective function is the max CPU usage of a 2s-simulation of runnables
- The search modifies one offset at a time, and updates other offsets only if timing constraints are violated
- Single-state search algorithms for discrete spaces (HC, Tabu)

*Case Study: an automotive software system with 430 runnables, search space = $10^{27}$*
Trade-off between Objectives

Car Makers

Part Suppliers

Execute $r_0$ to $r_3$ close to one another.

Minimize CPU time usage
Trade-off curve

Interesting Solutions

Boundary Trade Offs

# of slots

CPU time usage (ms)

21

12

1.45 1.56

1.45 1.56

1.56

2.04

2

3
Multi-objective search

- Multi-objective genetic algorithms (NSGA II)
- Supporting decision making and negotiation between stakeholders

Objectives:
- (1) Max CPU time
- (2) Maximum time slots between "dependent" tasks
Conclusions

- Search algorithms to compute offset values that reduce the max CPU time needed
- Generate reasonably good results for a large automotive system and in a small amount of time
- Used multi-objective search → tool for establishing trade-off between relaxing synchronization constraints and maximum CPU time usage
Schedulability Analysis and Stress Testing

References:

Problem

• **Schedulability analysis** encompasses techniques that try to predict whether all (critical) tasks are schedulable, i.e., meet their deadlines

• **Stress testing** runs carefully selected test cases that have a high probability of leading to deadline misses

• Stress testing is **complementary** to schedulability analysis

• Testing is typically expensive, e.g., hardware in the loop

• Finding stress test cases is difficult
Finding Stress Test Cases is Difficult

\[ j_0, j_1, j_2 \text{ arrive at } at_0, at_1, at_2 \text{ and must finish before } dl_0, dl_1, dl_2 \]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ j_0 \text{ finishes before } dl_0 \]

\[ j_1 \text{ finishes before } dl_1 \]

\[ j_2 \text{ finishes before } dl_2 \]

\[ J_1 \text{ can miss its deadline } dl_1 \text{ depending on when } at_2 \text{ occurs!} \]
Challenges and Solutions

• Ranges for arrival times form a very large input space

• Task interdependencies and properties constrain what parts of the space are feasible

• We re-expressed the problem as a constraint optimisation problem

• Constraint programming (e.g., IBM CPLEX)
System monitors gas leaks and fire in oil extraction platforms.
Constraint Optimization

Constraint Optimization Problem

Static Properties of Tasks
( Constants)

Dynamic Properties of Tasks
( Variables)

OS Scheduler Behaviour
( Constraints)

Performance Requirement
( Objective Function)
Process and Technologies

UML Modeling (e.g., MARTE)

System Design
System Platform

Design Model (Time and Concurrency Information)

Deadline Misses Analysis

Optimization Problem
(Find arrival times that maximize the chance of deadline misses)

Constraint Optimization

Constraint Programming (CP)

Solutions
(Task arrival times likely to lead to deadline misses)

Stress Test Cases

INPUT

OUTPUT

System Platform Solutions
(Task arrival times likely to lead to deadline misses)
Challenges and Solutions

• **Scalability problem:** Constraint programming (e.g., IBM CPLEX) cannot handle such large input spaces (CPU, memory)

• **Solution:** Combine metaheuristic search and constraint programming
  – metaheuristic search identifies high risk regions in the input space
  – constraint programming finds provably worst-case schedules within these (limited) regions
  – Achieve (nearly) GA efficiency and CP effectiveness
Combining CP and GA
Environment-Based Testing of Soft Real-Time Systems

References:

Objectives

- Model-based system testing
  - Independent test team
  - Black-box
  - Environment models
Environment: Domain Model

```plaintext
<<Context>>
RVM
- notRoutingFlag : Boolean
- signal: user_inserts_item()
- signal: SUT_item_arrived()
- signal: ITEM_LOST()

<<Context>>
User
- count : integer
- signal: insertionTime : Integer
- signal: rvm_sends_item()
- signal: insertionTime

<<Context>>
Sorter
- signal: moveArmTimeLC
- lowerBound = 280, upperBound = 320, scope = state
- signal: moveArmTimeCR
- lowerBound = 280, upperBound = 320, scope = state
- signal: POSITION_RIGHT()
- signal: POSITION_CENTRE()
- signal: POSITION_LEFT()
- signal: item_at_destination()

Items: time unique: lowerBound = 300, upperBound = 1100, scope = state
```
Environment: Behavioral Model
Test Case Generation

- Test objectives: Reach “error” states (critical environment states)
- Test Case: (1) Environment and (2) Simulation Configuration
  - (1) Number of instances for each component in domain model, e.g., number of items on conveying belt
  - (2) Setting non-deterministic properties of the environment, e.g., speed of sorter’s left and right arms
- Oracle: Reaching an “error” state
- Metaheuristics: search for test cases getting to error state
- Fitness function
  - Distance from error state
  - Distance from satisfying guard conditions
  - Time distance
  - Time in “risky” states
Stress Testing focused on Concurrency Faults

Reference:

Stress Testing of Distributed Systems

Reference:

General Pattern: Using Metaheuristics

- Search to optimize objective function
- Metaheuristics, constraint programming
- Scalability: A small part of the search space is traversed
- Model: Guidance to worst case, high risk scenarios across space
- Reasonable modeling effort based on standards or extension
- Heuristics: Extensive empirical studies are required
Scalability
Project examples

- Scalability is the most common verification challenge in practice

- Testing closed-loop controllers
  - Large input and configuration space
  - Smart heuristics to avoid simulations (machine learning)

- Schedulability analysis and stress testing
  - Large space of possible arrival times
  - Constraint programming cannot scale by itself
  - CP was carefully combined with genetic algorithms
Scalability: Lessons Learned

- Scalability must be part of the problem definition and solution from the start, not a refinement or an after-thought.
- Meta-heuristic search, by necessity, has been an essential part of the solutions, along with, in some cases, machine learning, statistics, etc.
- Scalability often leads to solutions that offer “best answers” within time constraints, but no guarantees.
- Scalability analysis should be a component of every research project – otherwise it is unlikely to be adopted in practice.
- How many papers research papers do include even a minimal form of scalability analysis?
Applicability
Project examples

• Applicability requires to account for the domain and context

• Testing controllers
  – Relies on Simulink only
  – No additional modeling or complex translation
  – Within domains, differences have huge implications in terms of applicability (open versus closed loop controllers)

• Minimizing risks of CPU shortage
  – Trade-off between effective synchronisation and CPU usage
  – Trade-off achieved through multiple objective GA search and appropriate decision tool

• Schedulability analysis and stress testing
  – Near deadline misses must be identified
Applicability: Lessons Learned

• In software engineering, and verification in particular, just understanding the real problems in real contexts is difficult
• What are the inputs required by the proposed technique?
• How does it fit in development practices?
• Is the output what engineers require to make decisions?
• There is no unique solution to a problem as they tend to be context dependent, but a context is rarely unique and often representative of a domain
Discussion

• **Metaheuristic search**
  – Tends to be versatile, easy to tailor to a new problem
  – Entails acceptable modeling requirements
  – Can provide “best” answers at any time
  – Scalable

**But**

– Not a proof, no certainty
– Though in practice (complex) models are not fully correct, there is no certainty anyway
– Effectiveness of search guidance is key and must be experimented and evaluated
– Models are key to provide adequate guidance
Talk Summary

• Focus: Meta-heuristic Search to enable scalable verification and testing.
• Scalability is the main challenge in practice.
• Drew lessons learned from example projects in collaboration with industry, on real systems and in real verification contexts.
• Results show that meta-heuristic search contributes to mitigate the scalability problem.
• It has shown to lead to applicable solutions in practice.
• Solutions are very context dependent.
• It is usually combined with a variety of other complementary techniques: system modeling, constraint solving, machine learning, statistics.
Scalable Software Testing and Verification Through Heuristic Search and Optimization

Lionel Briand

Interdisciplinary Centre for ICT Security, Reliability, and Trust (SnT)
University of Luxembourg, Luxembourg

QRS, Vancouver, August 3, 2015

SVV lab: svv.lu
SnT: www.securityandtrust.lu