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

University of Montreal, May 13, 2016
Acknowledgements

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

Scientists:
• Shiva Nejati
• Andrea Arcuri
• Yvan Labiche
Collaborative Research @ SnT Centre

- Research in context
- Addresses actual needs
- Well-defined problem
- Long-term collaborations
- Our lab is the industry
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, exercising a system such as to maximize chances of uncovering (important) latent faults within time constraints and resources.

- Other forms of verifications are important too (e.g., design time, run-time), but much less present in practice.
Decades of V&V research have not yet significantly and widely impacted engineering practice.
Cyber-Physical Systems: Challenges

- Increasingly complex and critical systems
- Complex environment
- Discrete and dynamic behavior
- Combinatorial and state explosion
- Complex requirements, e.g., temporal, timing, resource usage
- Uncertainty, e.g., about the environment
Scalable? Practical?

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

- **Practical**: Can a technique be efficiently and effectively applied by engineers in realistic conditions?
  - realistic ≠ universal
• **Formal Verification (Wikipedia):** In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

• **Our focus:** How can we, in a practical, effective and efficient manner, uncover as many (critical) faults as possible in software systems, within time constraints, while scaling to artifacts of realistic size.
Metaheuristics

- **Heuristic search (Metaheuristics):** Hill climbing, Tabu search, Simulated Annealing, Genetic algorithms, Ant colony optimisation …

- **Stochastic optimization:** General class of algorithms and techniques which employ some degree of randomness to find optimal (or as optimal as possible) solutions to hard problems

- Many verification and testing problems can be re-expressed as (hard) optimization problems
Talk Outline

• Selected project examples, with industry collaborations
• Similarities and patterns
• Lessons learned
Testing Software Controllers

References:

• R. Matinnejad et al., “Effective Test Suites for Mixed Discrete-Continuous Stateflow Controllers”, ACM ESEC/FSE 2015 (Distinguished paper award)
• R. Matinnejad et al., “MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models”, IEEE/ACM ASE 2014 (Distinguished paper award)
Electronic Control Units (ECUs)

Comfort and variety

More functions

Safety and reliability

Faster time-to-market

Greenhouse gas emission laws

Less fuel consumption
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)
Dynamic Continuous Controllers
Development Process

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

Software-in-the-Loop Stage
- Code Generation and Integration
- SiL Testing

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

- Data flow oriented
- Blocks and lines
- Time continuous and discrete behavior
- Input and outputs signals
MATLAB/Simulink model

Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, …
Testing Controllers at MIL

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

Actual value

Test Input

Test Output

Initial Desired Value

Final Desired Value

$T/2$ $T$

Desired Value

Actual Value
Time-dependent variables

\[
\begin{align*}
\text{output}(t) & \quad \text{desired}(t) \\
\text{Plant Model} & \quad \text{actual}(t) \\
& \quad \sum \\
P & \quad K_P e(t) \\
I & \quad K_I \int e(t) \, dt \\
D & \quad K_D \frac{de(t)}{dt} \\
& \quad e(t)
\end{align*}
\]

Configuration Parameters
Requirements and Test Objectives

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

Desired Value (input)
Actual Value (output)
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
Search-Based Software Testing

- Express test generation problem as a search problem
- Search for test input data with certain properties, i.e., constraints
- Non-linearity of software (if, loops, …): complex, discontinuous, non-linear search spaces (Baresel)
- Many search algorithms (metaheuristics), from local search to global search, e.g., Hill Climbing, Simulated Annealing and Genetic Algorithms

Search-Based Software Testing: Past, Present and Future
Phil McMinn
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
Smoothness Objective Functions: $O_{\text{Smoothness}}$

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

$O_{\text{Smoothness}}(\text{Test Case A}) > O_{\text{Smoothness}}(\text{Test Case B})$
Solution Overview (Simplified Version)

Objective Functions based on Requirements + Controller-plant model

1. Exploration

HeatMap Diagram

List of Critical Regions

Domain Expert

2. Single-State Search

Worst-Case Scenarios

Graph Builder

Final vs. Initial

Smoothness

0.100

0.150

0.200

0.250

0.300

Initial Desired

Final Desired

Desired Value

Actual Value

time

0

1

2
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

Test Results
Workspace Name: SBFC With Derivative Plus Tuning
Requirement: Stability
Heat Map Diagram

Run Model With Initial Desired - 0.8040 and Final Desired - 0.0636

Figure 1
File Edit View Insert Tools Desktop Window Help

Worst-case Test Scenarios

<table>
<thead>
<tr>
<th>Number</th>
<th>Initial Desired</th>
<th>Final Desired</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.80400</td>
<td>0.063627</td>
</tr>
</tbody>
</table>

Run Model With the Selected Test

Close
Analysis – Fitness increase over iterations
Analysis II – Search over different regions

**Average**  
**Random Search Distribution**  
**(1+1) EA Distribution**

Number of Iterations
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
- But not all configuration parameters matter for all objective functions
Modified Process and Technology

Objective Functions + Controller Model (Simulink)

1. Exploration with Dimensionality Reduction
   - Regression Tree
   - Domain Expert

List of Critical Partitions

2. Search with Surrogate Modeling
   - Visualization of the 8-dimension space using regression trees

Surrogate modeling to predict the objective function and speed up the search (Machine learning)

Dimensionality reduction to identify the significant variables (Elementary Effect Analysis)

Worst-Case Scenarios
Dimensionality Reduction

- Sensitivity Analysis: Elementary Effect Analysis (EEA)
- Identify non-influential inputs in computationally costly mathematical models
- Requires less data points than other techniques
- Observations are simulations generated during the Exploration step
- Compute sample mean and standard deviation for each dimension of the distribution of elementary effects
Imagine function $F$ with 2 inputs, $x$ and $y$:

For $X$:
- $F(A1) - F(A)$
- $F(B1) - F(B)$
- $F(C1) - F(C)$
- ...

For $Y$:
- $F(A2) - F(A)$
- $F(B2) - F(B)$
- $F(C2) - F(C)$
- ...

Elementary Effects Analysis Method
Visualization in Inputs & Configuration Space

Regression Tree
Surrogate Modeling During Search

- **Goal:** To predict the value of the objective functions within a critical partition, given a number of observations, and use that to avoid as many simulations as possible and speed up the search.
Surrogate Modeling During Search

- Any supervised learning or statistical technique providing fitness predictions with confidence intervals

1. Predict higher fitness with high confidence: Move to new position, no simulation
2. Predict lower fitness with high confidence: Do not move to new position, no simulation
3. Low confidence in prediction: Simulation
Experiments Results (RQ1)

- The best regression technique to build Surrogate models for all of our three objective functions is Polynomial Regression with $n = 3$

- Other supervised learning techniques, such as SVM

Mean of $R^2$/MRPE values for different surrogate modeling techniques

<table>
<thead>
<tr>
<th></th>
<th>LR $\ R^2$/MRPE</th>
<th>ER $\ R^2$/MRPE</th>
<th>PR($n=2$) $\ R^2$/MRPE</th>
<th>PR($n=3$) $\ R^2$/MRPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{sm}$</td>
<td>0.66/0.0526</td>
<td>0.44/0.0791</td>
<td>0.95/0.0203</td>
<td>0.98/0.0129</td>
</tr>
<tr>
<td>$F_r$</td>
<td>0.78/0.0295</td>
<td>0.49/1.2281</td>
<td>0.85/0.0247</td>
<td>0.85/0.0245</td>
</tr>
<tr>
<td>$F_{st}$</td>
<td>0.26/0.2043</td>
<td>0.22/1.2519</td>
<td>0.46/0.1755</td>
<td>0.54/0.1671</td>
</tr>
</tbody>
</table>
Experiments Results (RQ2)

- Dimensionality reduction helps generate better surrogate models for Smoothness and Responsiveness requirements.

Mean Relative Prediction Errors (MRPE Values)

- Smoothness ($F_{sm}$)
- Responsiveness ($F_r$)
- Stability ($F_{st}$)
Experiments Results (RQ3)

✓ For responsiveness, the search with SM was 8 times faster

✓ For smoothness, the search with SM was much more effective
Our approach is able to identify critical violations of the controller requirements that had neither been found by our earlier work nor by manual testing.

<table>
<thead>
<tr>
<th></th>
<th>MiL-Testing different configurations</th>
<th>MiL-Testing fixed configurations</th>
<th>Manual MiL-Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>2.2% deviation</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Smoothness</td>
<td>24% over/undershoot</td>
<td>20% over/undershoot</td>
<td>5% over/undershoot</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>170 ms response time</td>
<td>80 ms response time</td>
<td>50 ms response time</td>
</tr>
</tbody>
</table>
A Taxonomy of Automotive Functions

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

- Mixed discrete-continuous behavior: Simulink stateflows
- No plant model: 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 Based on Search

- Input diversity
- White-box Structural Coverage
  - State Coverage
  - Transition Coverage
- Output Diversity
- Failure-Based Selection Criteria
  - Domain specific failure patterns
  - Output Stability
  - Output Continuity
Failure-based Test Generation

- Search: Maximizing the likelihood of presence of specific failure patterns in output signals
- Domain-specific failure patterns elicited from engineers

**Instability**

**Discontinuity**
Summary of Results

- The test cases resulting from state/transition coverage algorithms cover the faulty parts of the models.
- However, they fail to generate output signals that are sufficiently distinct from the oracle signal, hence yielding a low fault revealing rate.
- Output-based algorithms are more effective.
Automated Testing of Driver Assistance Systems Through Simulation
Night Vision (NiVi) System

- The NiVi system is a camera-based collision-warning system providing improved vision at night
Testing DA Systems

- Testing DA systems requires complex and comprehensive simulation environments
  - Static objects: roads, weather, etc.
  - Dynamic objects: cars, humans, animals, etc.

- A simulation environment captures the behaviour of dynamic objects as well as constraints and relationships between dynamic and static objects.
Overview

Specification Documents
(Simulation Environment and NiVi System)

(1) Development of Requirements and domain models

Domain model \[\overset{\rightarrow}{\longrightarrow}\] Requirements model

(2) Generation of Test specifications

<table>
<thead>
<tr>
<th>test case specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
</tr>
<tr>
<td>[ranges/values/resolution]</td>
</tr>
<tr>
<td>Dynamic</td>
</tr>
<tr>
<td>[ranges/resolution]</td>
</tr>
</tbody>
</table>
NiVi and Environment Domain Model

**Weather**
- weatherType: Condition

**Road**
- roadType: RT

**SceneLight**
- intensity: Real

**Condition**
- fog
- rain
- snow
- normal

**RT**
- curved
- straight
- ramped

**Position**
- x: Real
- y: Real

**Test Scenario**
- simulationTime: Real
- timeStep: Real

**Camera Sensor**
- field of view: Real

**RoadSide Object**

**Trees**

**Parked Cars**

**Pedestrian**
- x: Real
- y: Real

**Vehicle**
- v:C: Real

**Collision**
- x_0: Real
- y_0: Real
- θ: Real
- v:p: Real

**Detection**
- state: Boolean

**NiVi**
- state: Boolean

**Output Trajectory**

**Dynamic Object**

**uses**

**positioned**
The NiVi system shall detect any person located in the Acute Warning Area of a vehicle.

\[ \exists t. \text{car.awa.pos}_{x1}[t] < \text{car.awa.human.trajecory.pos}_{x}[t] < \text{car.awa.pos}_{x2}[t] \land \text{car.awa.pos}_{y1}[t] < \text{car.awa.human.trajecory.pos}_{y}[t] < \text{car.awa.pos}_{y2}[t] \Rightarrow \text{car.sensor.warning} == \text{true} \]
MiL Testing via Search

Simulator + NiVi

- Fixed during Search
  - Environment Settings (Roads, weather, vehicle type, etc.)

- Manipulated by Search
  - Car Simulator (speed)
  - Human Simulator (initial position, speed, orientation)
  - NiVi

Meta-heuristic Search (multi-objective)

Generate scenarios

Detection or not?

Collision or not?
## Test Case Specification: Static (combinatorial)

<table>
<thead>
<tr>
<th>Situation 1</th>
<th>Type of Road</th>
<th>Type of vehicle</th>
<th>Type of actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Situation 2</td>
<td>Straight</td>
<td>Car</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 3</td>
<td>Straight</td>
<td>Car</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 4</td>
<td>Straight</td>
<td>Car</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 5</td>
<td>Straight</td>
<td>Truck</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 6</td>
<td>Straight</td>
<td>Truck</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 7</td>
<td>Straight</td>
<td>Truck</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 8</td>
<td>Straight</td>
<td>Car</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 9</td>
<td>Straight</td>
<td>Car</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 10</td>
<td>Straight</td>
<td>Truck</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 11</td>
<td>Straight</td>
<td>Track</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 12</td>
<td>Straight</td>
<td>Car</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 13</td>
<td>Straight</td>
<td>Car</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 14</td>
<td>Straight</td>
<td>Car</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 15</td>
<td>Straight</td>
<td>Truck</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 16</td>
<td>Straight</td>
<td>Truck</td>
<td>Child</td>
</tr>
<tr>
<td>Situation 17</td>
<td>Straight</td>
<td>Truck</td>
<td>Cow</td>
</tr>
<tr>
<td>Situation 18</td>
<td>Straight</td>
<td>Car + Cars in parking</td>
<td>Male</td>
</tr>
<tr>
<td>Situation 19</td>
<td>Straight</td>
<td>Car + buildings</td>
<td>Male</td>
</tr>
</tbody>
</table>
Test Case Specification: Dynamic

person : Actor
- PositionX = 74
- Position Y = 37.72
- Position Z = 0
- OrientationHeading = 93.33
- Acceleration = 0
- MaxWalkingSpeed = 14
- height = 1.75

startPerson : StartState
- Time = 0
- Speed = 12.59

profilePerson : Speed Profile
- Start location = 74
- Start location Y = 37.72
- Start location Z = 0
- Orientation = 0

trajectoryPerson : Trajectory
- Start location X = 74
- Start location Y = 37.72
- Start location Z = 0
- Orientation = 0

segmentPerson : Path Segment
- Length = 60
- Type = Straight
- MaxSpeedLimit = 14

pathPerson : Path
- StartPointX = 74
- StartPointY = 37.72
- StartPointZ = 0
- StartAngle = 93.33
- End Angle = 0
- Length = 60

profileCar : Speed Profile
- Start location X = 10
- Start location Y = 50.125
- Start location Z = 0.56
- Orientation = 0

trajectoryCar : Trajectory
- Start location X = 10
- Start location Y = 50.125
- Start location Z = 0.56
- Orientation = 0

segmentCar : Path Segment
- Length = 100
- Type = Straight
- MaxSpeedLimit = 100

pathCar : Path
- StartPointX = 10
- StartPointY = 50.125
- StartPointZ = 0.56
- StartAngle = 0
- End Angle = 0
- Length = 100

profileCar : Speed Profile
- Start location X = 10
- Start location Y = 50.125
- Start location Z = 0.56
- Orientation = 0

slotCar : Slot
- ID

startCar : StartState
- Time = 0
- Speed = 60.66

segmentCar : Path Segment
- Length = 100
- Type = Straight
- MaxSpeedLimit = 100

pathCar : Path
- StartPointX = 10
- StartPointY = 50.125
- StartPointZ = 0.56
- StartAngle = 0
- End Angle = 0
- Length = 100

profileCar : Speed Profile
- Start location X = 10
- Start location Y = 50.125
- Start location Z = 0.56
- Orientation = 0

slotCar : Slot
- ID

startCar : StartState
- Time = 0
- Speed = 60.66

Collision
- MinTTC = 0.3191
Choice of Surrogate Model

- Neural networks (NN) have been trained to learn complex functions predicting fitness values
- NN can be trained using different algorithms such as:
  - LM: Levenberg-Marquardt
  - BR: Bayesian regularization backpropagation
  - SCG: Scaled conjugate gradient backpropagation
- $R^2$ (coefficient of determination) indicates how well data fit a statistical model
- Computed $R^2$ for LM, BR and SCG $\Rightarrow$ BR has the highest $R^2$
Multi-Objective Search

- Search algorithm need objective or fitness functions for guidance
- In our case several independent ones can be interesting
- Objectives functions to minimize during simulation time:
  - Time to collision
  - Distance between car and pedestrian
  - Distance between pedestrian and AWA
- NSGA II algorithm
Pareto Front Projection

Straight Road with Parking

- ▲ Not detected
- ▼ Detected

TTC(s)

D(P/AWA)(m)
Simulation Scenario Execution

- Straight road with parking
- The person appears in the AWA, but is not detected
MO Search with NSGA-II

- Based on Genetic Algorithm
- \(N\): Archive and population size
- **Non-Dominated sorting**: Solutions are ranked according to how far they are from the Pareto front, fitness is based on rank.
- **Crowding Distance**: Individuals in the archive are being spread more evenly across the front (forcing diversity)
- **Runs simulations for close to \(N\) new solutions**
Improving Time Performance

- Individual simulations take on average more than 1 min
- It takes 10 hours to run our search-based test generation (≈ 500 simulations)
- We use surrogate modeling to improve the search
- Neural networks are used to predict fitness values within a confidence interval
- During the search, we use prediction values & prediction errors to run simulations only for the solutions likely to be selected
Search with Surrogate Models

NSGA II

Original Algorithm
- Runs simulations for all new solutions

Our Algorithm
- Uses prediction values & prediction errors to run simulations only for the solutions likely to be selected
Results – Surrogate Modeling

Comparing HV values obtained by (a) 20 runs of NSGAII and NSGAII-SM, (b) 20 runs of random search and NSGAII-SM, (c) the worst runs of NSGAII, NSGAII-SM, and Random search. The GD distributions obtained by running NSGAII-SM after 50 min and until 150 min are significantly better (with a medium confidence level, i.e., $\alpha = 0.95$); (b) 20 runs of NSGAII-SM and RS. The GD values obtained by NSGAII-SM at 140 min and 150 min are significantly better (with a small confidence level, i.e., $\alpha = 0.95$); (c) HV values for worst runs of NSGAII, NSGAII-SM, and Random search.
Results – Random Search

To summarize, we compared NSGAII-SM with Random Search (RS) in terms of HV values obtained by 20 runs of these algorithms up to 150 min. Figure 6(b) shows the HV results over time for NSGAII and NSGAII-SM. The HV values obtained by RS are significantly lower than those obtained by NSGAII-SM at 140 min and 150 min, with a confidence level of 95%. With NSGAII, the tester might encounter worse outcomes than those obtained by NSGAII-SM. With NSGAII-SM, this average is equal to 7.9. Hence, given the same time budget, NSGAII-SM is able to produce more optimal results compared to NSGAII. Therefore, given a fixed execution time, NSGAII-SM yields better HV results compared to Random search. Consequently, we can consider NSGAII-SM to be a safer algorithm to use, especially under tight time budget constraints. We note that as shown in Figure 6(c) the HV values obtained by NSGAII-SM at 140 min and 150 min are significantly better (with a small confidence level, i.e., cl = 0.95, 0.9 and 0.8). The HV and GD results indicated that NSGAII-SM performs best, and this conclusion holds for a wide range of time budgets. As a result, NSGAII-SM appears to be a safer algorithm to use, especially under tight time budget constraints.
Results – Worst Runs

In the initial 50 minutes of the time to run the algorithm only once, testers will likely have obtained solutions by NSGAII. Given the large execution time of our algorithms, the GD results indicated that NSGAII-SM performs best, and as a result, as shown in Figure 6(a), given the same time budget, NSGAII-SM is able to produce more optimal algorithms and parameters before concluding with the main experiment reported in this paper. However, as shown in Figure 6(a), the medians and averages of the HV values obtained by NSGAII-SM at each iteration are likely to be better than those obtained by Random search with a medium confidence level, i.e., $P = 0.95$; (b) 20 runs of NSGAII and NSGAII-SM ($P = 0.95$); (c) the worst runs of NSGAII, NSGAII-SM and Random search.

Further, the GD values obtained by NSGAII-SM at 140 min and 150 min are significantly better (with a small $e$-value) than those obtained by NSGAII at 140 min and 150 min, as shown in Figure 6(b). Similarly, we compared the GD values obtained by NSGAII-SM after 50 min and until 150 min are significantly better (with a medium $e$-value) than those obtained by RS and NSGAII-SM. The GD distributions obtained by running NSGAII-SM after 50 min and until 150 min are significantly better (with a small $e$-value). As shown in the figure, after 50 min execution, NSGAII-SM is able to perform more generations (iterations) than NSGAII within the same execution time. As discussed, NSGAII-SM is able to perform more generations (iterations) than NSGAII with a probability of 97.5%, at a given iteration and provided constraints. We note that as shown in Figure 6(c) the HV results, after 50 min executing these algorithms, the average GD values obtained by NSGAII-SM are higher than the medians and averages of the HV values obtained by NSGAII-SM in this paper.
Minimizing CPU Shortage Risks During Integration

References:

Automotive: Distributed Development
Software Integration
Stakeholders

Car Makers
• Develop software optimized for their specific hardware
• Provide integrator with runnables

Integrator
• 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

Integrator

- 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
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
- **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

![Diagram showing CPU time usage simulations](image)
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
Without optimization

CPU time usage exceeds the size of the slot (5ms)
CPU time usage always remains less than 2.13ms, so more than half of each slot is guaranteed to be free.
Single-objective Search algorithms

Hill Climbing and Tabu Search and their variations

Solution Representation

a vector of offset values: \( o_0=0, o_1=5, o_2=5, o_3=0 \)

Tweak operator

\[ o_0=0, o_1=5, o_2=5, o_3=0 \rightarrow o_0=0, o_1=5, o_2=10, o_3=0 \]

Synchronization Constraints

offset values are modified to satisfy constraints

Fitness Function

max CPU time usage per time slot
Summary of Problem and Solution

**Optimization**
while satisfying synchronization/temporal constraints

**Explicit Time Model**
for real-time embedded systems

**Search**
meta-heuristic single objective search algorithms

**10^27**
an industrial case study with a large search space
Search Solution and Results

- 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**

- Running the system without offsets: 5.34 ms
- Optimized offset assignment: 2.13 ms
Comparing different search algorithms

Best CPU usage

Time to find Best CPU usage
Comparing our best search algorithm with random search

(a) Lowest max CPU usage values computed by HC within 70 ms over 100 different runs

(b) Lowest max CPU usage values computed by Random within 70 ms over 100 different runs

(c) Comparing average behavior of Random and HC in computing lowest max CPU usage values within 70 s and over 100 different runs

HC

Random

Average
Trade-off between Objectives

Car Makers

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

Integrator

Minimize CPU time usage

1 slot

2 slots

3 slots
Trade-off curve

- **# of slots**
  - 1
  - 2
  - 3

- **CPU time usage (ms)**
  - 1.45
  - 1.56
  - 2.04

**Boundary Trade Offs**

**Interesting Solutions**
Multi-objective search

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

Objectives:
- (1) Max CPU time
- (2) Maximum time slots between “dependent” tasks
Trade-Off Analysis Tool

Input.csv:
- runnables
- Periods
- CETs
- Groups
- # of slots per groups

Search

A list of solutions:
- objective 1 (CPU usage)
- objective 2 (# of slots)
- vector of group slots
- vector of offsets

Visualization/Query Analysis

- Visualize solutions
- Retrieve/visualize simulations
- Visualize Pareto Fronts
- Apply queries to the solutions
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:

• S. Di Alesio et al., “Stress Testing of Task Deadlines: A Constraint Programming Approach”, IEEE ISSRE 2013, San Jose, USA
Real-time, concurrent systems (RTCS) have concurrent interdependent tasks which have to finish before their deadlines.

Some task properties depend on the environment, some are design choices.

Tasks can trigger other tasks, and can share computational resources with other tasks.

How can we determine whether tasks meet their deadlines?
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

$j_0$, $j_1$, $j_2$ arrive at $at_0$, $at_1$, $at_2$ and must finish before $dl_0$, $dl_1$, $dl_2$

$J_1$ can miss its deadline $dl_1$ depending on when $at_2$ 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)
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)

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

Deadline Misses Analysis

Constraint Optimization

Constraint Programming (CP)

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

Stress Test Cases

INPUT

OUTPUT
System monitors gas leaks and fire in oil extraction platforms.
Challenges and Solutions

• CP effective on small problems
• **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 (GA) 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

• **Our approach can be used both for stress testing and schedulability analysis (assumption free)**
Combining GA and CP

A:12
S. Di Alesio et al.

Fig. 3: Overview of GA+CP: the solutions $x_1$, $y_1$, and $z_1$ in the initial population of GA evolve into $x_6$, $y_6$, and $z_6$, then CP searches in their neighborhood for the optimal solutions $x^*$, $y^*$ and $z^*$.

Let $\mathcal{J}^*(x)$ be the union of the impacting sets of tasks missing or closest to miss their deadlines.

Let $\mathcal{I}^*(x)$ be the union of the impacting sets of tasks in $\mathcal{J}^*(x)$:

$$
\mathcal{I}^*(x) \overset{\text{def}}{=} \left\{ j^* \in \mathcal{J}^*(x) \mid I_j^*(x) \right\}
$$

By definition, $\mathcal{I}^*(x)$ contains all the tasks that can have an impact over a task that misses a deadline or is closest to a deadline miss.

Constraint Model $M$ implementing a Complete Search Strategy. Let $M$ be the constraint model defined in our previous work [Di Alesio et al. 2014] for the purpose of identifying arrival times for tasks that are likely to lead to deadline misses scenarios.

$M$ models the static and dynamic properties of the software system respectively as constants and variables, and the scheduler of the operating system as a set of constraints among such variables. Note that $M$ implements a complete search strategy over the space of arrival times. This means that $M$ searches for arrival times of all aperiodic tasks within the whole interval $T$.

Our combined GA+CP strategy consists in the following two steps:
Process and Technologies

UML Modeling (e.g., MARTE)

Constraint Optimization

**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)*

**Genetic Algorithms (GA)**

**Constraint Programming (CP)**

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

INPUT

OUTPUT
Environment-Based Testing of Soft Real-Time Systems

References:

Objectives

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

- **RVM**
  - notRoutingFlag: Boolean
  - signal user_inserts_item()
  - signal SUT_item_arrived()
  - signal ITEM_LOST()

- **User**
  - count: Integer
  - signal insertionTime: Integer
  - signal rvm_sends_item()

- **Sorter**
  - signal moveArmTimeLC: Integer
  - signal moveArmTimeCR: Integer
  - destination: String
  - signal POSITION_RIGHT()
  - signal POSITION_CENTRE()
  - signal POSITION_LEFT()
  - signal item_at_destination()
Environment: Behavioral Model
Test Case Generation

• Test objectives: Reach “error” states (critical environment states)
• Test Case: Simulation Configuration
  – 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 functions
  – 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:

V&V Topics Addressed by Search

- Many projects over the last 15 years

- Design-time verification
  - Schedulability
  - Concurrency
  - Resource usage

- Testing
  - Stress/load testing, e.g., task deadlines
  - Robustness testing, e.g., data errors
  - Reachability of safety or business critical states, e.g., collision and no warning
  - Security testing, e.g., XML injections
General Pattern: Using Metaheuristic Search

- Problem = fault model
- Model = system or environment
- Search to optimize objective function(s)
- Metaheuristics
- 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
General Pattern: Using Metaheuristic Search

- Simulation can be time consuming
- Makes the search impractical or ineffective
- Surrogate modeling based on machine learning
- Simulator dedicated to search
Scalability
• Scalability is the most common verification challenge in practice

• Testing closed-loop controllers, DA system
  – Large input and configuration space
  – Expensive simulations
  – Smart heuristics to avoid simulations (machine learning to predict fitness)

• 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?
Practicality
Project examples

• Practicality requires to account for the domain and context

• Testing controllers
  – Relies on Simulink only
  – No additional modeling or complex translation
  – Differences between open versus closed loop controllers

• Minimizing risks of CPU shortage
  – Trade-off between between effective synchronization and CPU usage
  – Trade-off achieved through multiple-objective GA search and appropriate decision tool
In software engineering, and verification in particular, just understanding the real problems in context 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 or type of system.
Discussion

- **Metaheuristic search for verification**
  - Tends to be versatile, tailored to new problems and contexts
  - Can cope with the verification of dynamic behavior
  - Entails acceptable modeling requirements
  - Can provide “best” answers at any time
  - Scalable, practical

**But**

- Not a proof, no certainty
- Effectiveness of search guidance is key and must be experimentally evaluated
- Models are key to provide adequate guidance
- Search must often be combined with other techniques, e.g., machine learning, constraint programming
Discussion II

• **Constraint solvers (e.g., Comet, ILOG CPLEX, SICStus)**
  – Is there an efficient constraint model for the problem at hand?
  – Can effective heuristics be found to order the search?
  – Better if there is a match to a known standard problem, e.g., job shop scheduling
  – Tend to be strongly affected by small changes in the problem, e.g., allowing task pre-emption
  – Often not scalable, e.g., memory

• **Model checking**
  – Detailed operational models (e.g., state models), involving (complex) temporal properties (e.g., CTL)
  – Enough details to analyze statically or execute symbolically
  – These modeling requirements are usually not realistic in actual system development. State explosion problem.
  – Originally designed for checking temporal properties through reachability analysis, as opposed to explicit timing properties
  – Often not scalable
Talk Summary

• Focus: Meta-heuristic Search to enable scalable verification and testing.
• Scalability is the main challenge in practice.
• We 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 also shown to lead to practical solutions.
• Solutions are very context dependent.
• Solutions tend to be multidisciplinary: system modeling, constraint solving, machine learning, statistics.
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

University of Montreal, May 13, 2016

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