Artificial Intelligence in Software Testing: An Overview

Application to Industrial Robotics

Arnaud Gotlieb
Simula Research Laboratory
Norway
The Certus Centre

Software Validation and Verification

Cisco Systems Norway

Cancer Registry of Norway

ABB Robotics

Kongsberg Maritime

www.certus-sfi.no
Industrial Robotics Evolves Very Fast!

Industrial robots are now complex cyber-physical systems (motion control and perception systems, multi-robots sync., remote control, Inter-connected for predictive maintenance, ...)

They are used to perform safety-critical tasks in complete autonomy (high-voltage component, on-demand painting with color/brush change, ..)

And to collaborate with human co-workers
Testing Robotic Systems is Crucial and Challenging

- The validation of industrial robots still involve too much human labour
- “Hurry-up, the robots are uncaged!”: Failures are not anymore handled using fences
- Robot behaviours evolve with changing working conditions
- Today, industrial robots can be taught by imitation. Tomorrow, they will learn by themselves

More automation in testing
More diversity in testing
More efficiency in testing
A Typical Cycle of Continuous Integration:

- Developer commit
- Software building
- Software Deployment
- Developer feedback
- Software Testing
- Test Case Selection/Generation
- Test Suite Reduction
- Test Case Prioritization
- Test Execution Scheduling
- Test Execution
Artificial Intelligence in a Nutshell

Perception
- Computer Vision
- Pattern Recognition
- Natural Language Processing

Representation
- Conceptual Graphs
- Conditional Preference Networks

Cognition
- Deep Learning
- Reinforcement Learning
- Constraint Programming
- Multi-Criteria Decision

Interaction
- Multi-Agent Systems
- Human-Machine Interactions

Execution
- Planification
- Optimization
- Scheduling

AI for Improved Software Engineering / Software Testing

Explainable AI - Verified AI - Certifiable AI
Our Focus: Artificial Intelligence for Improving Software Testing

1. Automatic Test Case Generation
2. Test Suite Reduction
3. Test Execution Scheduling
4. Test Case Prioritization

Constraint Modelling

Reinforcement Learning

Global Constraints

Constraint-based Scheduling
1. Automatic Test Case Generation

Constraint Modelling
A Typical Robot Painting Scenario

Crucial test objective: to validate that physical outputs are triggered on expected time

Current practice:

Main issue:
Can we automate this testing process?
Can we integrate an AI model into Continuous Testing?

Paint Valve=On at x:=50
Set Fluid=100 at x:=100 (Pump, mL/min)
Set Atom=15000 at x:=180 (Air flow, L/min)
Set Shape=7500 at x:=250 (Air flow, L/min)

Set Brush 1 at x:=300
Need to call 4 physical subprocesses
Start of ‘brush’
Spray pattern to be painted
X=300ms
Issues for deployment:

1. Can we control the solving time wrt the test execution time?
2. Is this Constraint-based Testing approach interesting to find bugs?
3. Can we ensure enough diversity in the generated test scenarii?
Industrial Deployment [Mossige et al. CP’14, IST’15]

- Integrated through ABB’s Continuous Integration process
- Constraint model is solved ~15 times per day

During initial deployment, it found 5 critical bugs + dozens of (non-critical) new bugs

But, since then, bug discovery has decreased! still working on

1. Maximizing the diversity among test scenarii
2. Generating test scenarios for multi-robots

\[
E = \frac{\text{SeqLen}}{t_s + t_N}
\]

\[
E = \frac{\text{SeqLen}}{t_s + t_N}
\]

- Size of the Brush Table
  - SeqLen =
    - 50
    - 100
    - 150
    - 200
    - 250
    - 300
2. Test Suite Reduction

Global Constraints
Test Suite Reduction: the core problem

$F_i$: Features / User Requirements
TC: Test Cases

Finding a near-optimal solution in a given contract of time is sufficient!

NP-hard problem!
Other criteria to minimize

Feature coverage is always a prerequisite

Execution time!
Other criteria to minimize

Feature coverage is always a prerequisite

Fault revealing capabilities!
Test Suite Reduction: Existing Approaches

- Exact methods: Integer Linear Programming  
  
  \[
  \text{Minimize } \sum_{i=1..6} x_i
  \]
  \[
  \text{subject to } \begin{cases}
  x_1 + x_2 + x_6 \geq 1 \\
  x_3 + x_4 \geq 1 \\
  x_2 + x_5 \geq 1
  \end{cases}
  \]
  (minimize the number of test cases)
  (cover every feature. at least once)

- Approximation algorithms (greedy, search-based methods)  
  
  \[
  F = \text{Set of reqs}, \ \text{Current} = \emptyset
  \]
  \[
  \text{while} (\text{Current} \neq F)
  \]
  \[
  \text{Select a test case that covers the most uncovered features;}
  \]
  \[
  \text{Add covered features to Current;}
  \]
  \[
  \text{return Current}
  \]

- AI-powered method:
  
  **Constraint Programming with Global Constraints**  
  [Gotlieb et al. ISSTA 2014, AI Magazine 2016, ...]

  **Multi-Criteria Test Minimization**  
  [Wang et al. JSS 2015, ESE 2015, ...]
**gcc**: global cardinality constraint
Powerfull AI combinatorial tool

Variability model to describe a product line

Minimize TotalCost
\[ s.t. \quad \text{gcc}(F_1, \ldots, F_n, [t_1, \ldots, t_m], [O_1, \ldots, O_m]) \]
for \( i = 1 \) to \( m \) do \( B_i = (O > 0) \)
\[ \text{scalar}_\text{product}(B_1, \ldots, B_n; [c_1, \ldots, c_m], \text{TotalCost}) \]

Where \( \text{scalar}_\text{product} \) encodes \( B_1 \cdot c_1 + \ldots + B_n \cdot c_n = \text{TotalCost} \)

Unoptimized test suite

Optimized (reduced) test suite

Diagnostic views, feature coverage

This cost value aggregates different criteria (e.g., execution time, ...)

**SICS**

IRB 5500  IRB 5400-22  IRB 580  IRB 540  IRB 52

Rail sys  IRB 5400-12  IRB 58  IRB 52
Comparison with CPLEX, MiniSAT, Greedy (uniform costs)

(Reduced Test Suite percentage in 60 sec)
3. Test Execution Scheduling

Constraint-based Scheduling
Test Execution Scheduling

Assignment of Test Cases to Agents such that:

1. Capacity constraints are not exceeded
2. Test Agents are well occupied
3. Test Execution Time is minimized

Additionally, there can be some shared global resources among test cases (e.g., flow meter, oscilloscope, camera, …)
Constraint Models for Test Scheduling

Constraint-based scheduling Models
1. Greedy approach
2. Constraint-based scheduling
3. Advanced Constraint-based scheduling using bin-packing

Test Cases Repository:
~10,000 Test Cases (TC)
~25 distinct Test Robots
Diverse tested features

- 10..30 code changes per Day

<table>
<thead>
<tr>
<th></th>
<th>Deployed at ABB in CI / «Good Enough»</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Evaluated / Needs Improvements</td>
</tr>
<tr>
<td>3</td>
<td>Deployment in progress</td>
</tr>
</tbody>
</table>

- T2, T5, T34
- T3, T6, T45, T78
- T7, T23
- T4, T56, T67
- T45, T55
Experimental results (Comparing model 3 vs model 1)

But, handling test case diversity is challenging!
4. Test Case Prioritization

Reinforcement Learning
Motivation: Learning from previous test runs of the robot control systems

- Adapt testing to focus on the more error-prone parts of the tested system
- Adapt testing to the execution environment (available robots and devices, limited testing time and resources, experiences from previous cycles in continuous integration)
Using Reinforcement Learning to prioritize test case execution

- Considering test case meta-data only (test verdicts, tested robots, execution time, ...) → lightweight method
- Reward function based on test verdicts from the previous CI-cycles → online ML
- Limited memory of past executions / test results

Implemented with distinct memory models and reward functions
Does it learn?

3 Industrial data sets (1 year of CI cycles)
NAPFD: Normalized Average Percentage of Faults Detected

**Reward Function 1. Failure Count Reward**

\[ \text{reward}_{i}^{\text{fail}}(t) = |T S_{i}^{\text{fail}}| \quad (\forall t \in T_{i}) \]

**Reward Function 2. Test Case Failure Reward**

\[ \text{reward}_{i}^{\text{tc fail}}(t) = \begin{cases} 1 - t.\text{verdict} & \text{if } t \in T S_{i} \\ 0 & \text{otherwise} \end{cases} \]

**Reward Function 3. Time-ranked Reward**

\[ \text{reward}_{i}^{\text{time}}(t) = |T S_{i}^{\text{fail}}| - t.\text{verdict}_{i} \times \sum_{t_{k} \in T S_{i}^{\text{fail}}} 1 \quad \text{if } rank(t) < rank(t_{k}) \]
Lessons Learned and Emerging Topics
Lessons learned

• Industrial Robotics is an interesting application field for AI-powered software testing approaches

• More automation is highly desired in industrial robotics
  AI is a key-enabler for  Release better, release faster, release cheaper!

• Adoption of (robust) AI techniques beneficial in test automation and optimization:
  Constraint Programming, Scheduling, Reinforcement Learning, ...

Many Emerging Challenges!
Emerging Topics

• Testing Learning Robots (RCN T-LARGO Project)

• Machine Learning in Continuous Testing Processes (Collaboration Smartesting)

• AI-on-demand platform for performance testing of industrial robots (AI4EU H2020 Proposal)

• Testing Human Perception of Robot Safety

Thanks to:

Dusica Marijan (SIMULA, Norway)
Morten Mossige (ABB Robotics, Norway)
Helge Spieker (SIMULA, Norway)
Shuai Wang (SIMULA, Norway)
Marius Liaeen (CISCO Systems, Norway)
Mats Carlsson (SICS, Sweden)
Carlo Ieva (SIMULA, Norway)
....
References

   Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration

2. [Gotlieb Marijan 2017] A. Gotlieb and D. Marijan
   Using Global Constraints to Automate Regression Testing
   AI Magazine 38, Spring, 2017.

   TITAN: Test Suite Optimization for Highly Configurable Software
   In Int. Conf. on Soft. Testing, Verification and Validation (ICST-17), Tools Track, Tokyo, Japan, 2017.

   Time-aware Test Case Execution Scheduling for Cyber-Physical Systems
   In Principles and Practice of Constraint Programming (CP-17) – Application Track, Melbourne, Australia, Aug. 2017

5. [Gotlieb et al., 2016] A. Gotlieb, M. Carlsson, D. Marijan and A. Petillon
   A New Approach to Feature-based Test Suite Reduction in Software Product Line Testing
   In 11th Int. Conf. on Software Engineering and Applications (ICSOFT-16), Lisbon, July 2016, Awarded Best Paper

   Testing robot controllers using constraint programming and continuous integration.

   Cost-effective test suite minimization in product lines using search techniques.

   Flower: Optimal test suite reduction as a network maximum flow.
   In Proc. of Int. Symp. on Soft. Testing and Analysis (ISSTA-14), San José, CA, USA, Jul. 2014.

   Using CP in automatic test generation for ABB robotics' paint control system.