Testing Robotic Systems:

Arnaud Gotlieb
Simula Research Laboratory
Lysaker, 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
# How Software Development of Industrial Robots Has Evolved...

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-core, single application system</td>
<td>Multi-core, complex distributed system</td>
</tr>
<tr>
<td>All source code maintained by a small team located at the same place</td>
<td>Subsystems developed by distinct teams located at distinct places in the world</td>
</tr>
<tr>
<td>Manual system testing only handled in a single place, on actual robots</td>
<td>Automated software testing handled in a continuous integration process</td>
</tr>
</tbody>
</table>
A Typical Cycle of Continuous Integration:

- Developer commit
  - Software building
    - Software Deployment
      - Software Testing
        - Developer feedback
          - Test Case Selection/Generation
            - Test Suite Reduction
              - Test Case Prioritization
                - Test Execution Scheduling
                  - Test Execution

Timeline
Our Focus: Artificial Intelligence for Testing of Robotic Systems

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
A Typical Robot Painting Scenario

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

Current practice:

Main issue: Can we generate automatically test scenarios and check results using sensors?

SetBrush 1 at \(x:=300\)

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)

Paint Valve=On at \(x:=50\)
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]

- Constraint model: 2KLOC of Prolog, finite domains constraint solver (clpfd + home-made heuristics)
  - Time-aware constraint-based optimization
  - Integrated through ABB’s Continuous Integration process
  - Constraint model is solved ~15 times per day

But, still working on maximizing the diversity among test scenarios.

E: Efficiency factor
\[ E = \frac{SeqLen}{(t_s + t_N)} \]

- It founds 5 re-introduced (already corrected) critical bugs
- It founds dozens of (non-critical) new bugs
2. Test Suite Reduction

Global Constraints
Test Suite Reduction: the core problem

$F_i$: Features
TC: Test Cases

Similar to the Vertex Cover problem in a bipartite graph
Test Suite Reduction: existing approaches

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

  \[ \text{[Hsu Orso ICSE 2009, Campos Abreu QSIC 2013,...]} \]

- Approximation algorithms (greedy, search-based methods)
  \[ \text{[Harrold et al. TOSEM 1993, ...]} \]

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

- Constraint Programming with global constraints \[ \text{[Gotlieb et al. ISSTA 2014, AI Magazine 2016, ...]} \]
Constraint Programming (CP)

• Routinely used in Validation & Verification, CP handles hundreds of thousands of constraints

• CP is versatile: user-defined constraints, dedicated solvers, programming search heuristics but it is not a silver bullet (developing efficient CP models requires expertise)

→ Global constraints: relations over a non-fixed number of variables, implementing dedicated filtering algorithms
The **nvalue** global constraint

[Pachet Roy 1999, Beldiceanu 01]

**nvalue**(N, V)

Where:

N is a finite-domain variable
V = [V₁, ..., Vₖ] is a vector of variables

**nvalue**(N, V) holds iff N = \( \text{card}(\{V_i\}_{i \in 1..k}) \)

**nvalue**(N, [3, 1, 3]) entails N = 2
**nvalue**(3, [X₁, X₂]) fails
**nvalue**(1, [X₁, X₂, X₃]) entails \( X_1 = X_2 = X_3 \)
N in 1..2, **nvalue**(N, [4, 7, X₃]) entails \( X_3 \in \{4,7\} \), N=2
Optimal Test Suite Reduction with $nvalue$

However, only $F_1$, $F_2$, $F_3$ are available for labeling!

$F_1 \in \{1, 2, 6\}, \; F_2 \in \{3, 4\}, \; F_3 \in \{2, 5\}$

$nvalue(\ MaxNvalue, [F_1, F_2, F_3] )$

Minimize($MaxNvalue$)
The global_cardinality constraint (gcc)

Where
\[ T = [T_1, \ldots, T_N] \] is a vector of N variables
\[ d = [d_1, \ldots, d_k] \] is a vector of k values
\[ V = [V_1, \ldots, V_k] \] is a vector of k variables

\[
gcc(T, d, V) \text{ holds iff } \forall i \ in 1..k, \ V_i = \text{card}\{j \mid T_j = d_i\}\]

Filtering algorithms for gcc are based on max flow computations
Example

gcc( \{F_1, F_2, F_3\}, \{1,2,3,4,5,6\}, \{V_1,V_2,V_3,V_4,V_5,V_6\})
means that:

In the solution-set,
TC1 is used to cover exactly \( V_1 \) features in \( \{F_1, F_2, F_3\} \)
TC2          ‘’ \( V_2 \) ‘’
TC3          ‘’ \( V_3 \) ‘’
...

\( F_1 \) in \{1, 2, 6\}, \( F_2 \) in \{3, 4\}, \( F_3 \) in \{2, 5\}
\( V_1 \) in \{0, 1\}, \( V_2 \) in \{0, 1, 2\}, \( V_3 \) in \{0, 1\}, \( V_4 \) in \{0, 1\}, \( V_5 \) in \{0, 1\}, \( V_6 \) in \{0, 1\}

Here, \( V_1=1, V_2=1, V_3=1, V_4=0, V_5=0, V_6=0 \) is a feasible solution

But, not an optimal one!
CP model using **gcc** and **nvalue**

\[ F_1 \text{ in } \{1, 2, 6\}, \ F_2 \text{ in } \{3, 4\}, \ F_3 \text{ in } \{2, 5\} \]

\[
gcc( [F_1, F_2, F_3], [1,2,3,4,5,6], [V_1, V_2, V_3, V_4, V_5, V_6] )
\]

\[
nvalue(\text{MaxNvalue}, [F_1, F_2, F_3])
\]

Minimize(\text{MaxNvalue})
Model pre-processing

\( F_1 \text{ in } \{1, 2, 6\} \rightarrow F_1 = 2 \)

as \( \text{cov}(TC_1) \subset \text{cov}(TC_2) \) and \( \text{cov}(TC_6) \subset \text{cov}(TC_2) \)
withdraw TC_1 and TC_6

\( F_3 \text{ is covered } \rightarrow \text{withdraw TC}_5 \)

\( F_2 \text{ in } \{3, 4\} \rightarrow \text{e.g., } F_2 = 3, \text{ withdraw TC}_4 \)

Pre-processing rules can be expressed once and then applied iteratively
Other criteria to minimize

Feature coverage is always a prerequisite

Execution time!
Feature coverage is always a prerequisite

Fault revealing capabilities!
Proposed approaches

1. Actual multi-objectives optimization with search-based algorithms (Pareto Front)  
   \[\text{Wang et al. JSS'15}\]
   Aggregated cost function using weights for each objective

2. Cost-based single-objective constrained optimization
   Based on a CP model with global constraints

   Approximate solutions
   No constraint model!

   Exact solutions
   Constrained optimization model!
Optimal Test Suite Reduction with Costs

[Gotlieb et al. ICSOFT-EA’16]

\[ F_1, \ldots, F_n: \text{Features} \]
\[ t_1, \ldots, t_m: \text{Test cases} \]
\[ c_1, \ldots, c_m: \text{Unit cost for each test case} \]

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

\[
\text{Minimize TotalCost}
\]
\[
\text{s.t}
\]
\[
gcc([F_1, \ldots, F_n], [t_1, \ldots, t_m], [O_1, \ldots, O_m])
\]
\[
\text{for } i=1 \text{ to } m \text{ do } B_i = (O_i > 0)
\]
\[
\text{scalar_product}([B_1, \ldots, B_m], [c_1, \ldots, c_m], \text{TotalCost})
\]

where \text{scalar_product} encodes \( B_1 \times c_1 + \ldots + B_m \times c_m = \text{TotalCost} \)
TITAN [Marijan, Gotlieb ICST’17]

Variability model to describe a product line

Diagnostic views, feature coverage

Unoptimized test suite

Optimized (reduced) test suite
Model comparison on random instances (uniform costs)

(Reduced Test Suite percentage in 30sec of search)
Comparison with CPLEX, MiniSAT, Greedy (uniform costs)
(Reduced Test Suite percentage in 60 sec)

But, less encouraging results when non-uniform costs are used!
(CPLEX always better than TITAN)
3. Test Execution Scheduling

Constraint-based Scheduling
Test Execution Scheduling

Test Cases with distinct characteristics

Test Agents (Robots) with limited (time or resources) capacity

Schedule

Assignment of Test Cases to Agents which
1. Satisfies capacity constraints
2. Optimizes some cost function

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

Test Cases: - duration [- priority] [- history]

SIMULA’s SWMOD

Constraint-based scheduling Models
1. Greedy approach
2. Constraint-based scheduling
3. Advanced scheduling based on global constraints / Labelling heuristics

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Deployed at ABB / « good enough »</td>
</tr>
<tr>
<td>2</td>
<td>Evaluated / Needs Improvements</td>
</tr>
<tr>
<td>3</td>
<td>Evaluation in progress / Not yet deployed</td>
</tr>
</tbody>
</table>

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

10..30 code changes per Day
Formally speaking

Variables:
- $t$: a set of Test Cases to schedule with their (known) duration
- $r$: a set of (shareable) resources
- $m$: a set of Test Agents and a relation $f: t \rightarrow m$

Constraints:
- Each Test Case must be executed (exactly) once, without possible preemption;
- None shared resource is used by two Test Cases at the same time;
- $f$ has to be satisfied;
- At most $\text{card}(m)$ Test Cases can be executed at any moment;

Function to optimize:
- $\text{Timespan}$: the overall duration of the schedule
  (in order to minimize the round-trip time)

NP-hard problem!
A realistic example

<table>
<thead>
<tr>
<th>Test</th>
<th>Duration</th>
<th>Executable on</th>
<th>Use of global resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>2</td>
<td>m1, m2, m3</td>
<td>-</td>
</tr>
<tr>
<td>t2</td>
<td>4</td>
<td>m1, m2, m3</td>
<td>r1</td>
</tr>
<tr>
<td>t3</td>
<td>3</td>
<td>m1, m2, m3</td>
<td>r1</td>
</tr>
<tr>
<td>t4</td>
<td>4</td>
<td>m1, m2, m3</td>
<td>r1</td>
</tr>
<tr>
<td>t5</td>
<td>3</td>
<td>m1, m2, m3</td>
<td>r1</td>
</tr>
<tr>
<td>t6</td>
<td>2</td>
<td>m1, m2, m3</td>
<td>-</td>
</tr>
<tr>
<td>t7</td>
<td>1</td>
<td>m1</td>
<td>-</td>
</tr>
<tr>
<td>t8</td>
<td>2</td>
<td>m2</td>
<td>-</td>
</tr>
<tr>
<td>t9</td>
<td>3</td>
<td>m3</td>
<td>-</td>
</tr>
<tr>
<td>t10</td>
<td>5</td>
<td>m1, m3</td>
<td>-</td>
</tr>
</tbody>
</table>

Diagram showing time intervals and resource usage.
The cumulative global constraint  

\[ \text{cumulative}(t, d, r, m) \]

Where

- \( t = (t_1, \ldots, t_N) \) is a vector of tasks, each \( t_i \) in \( \text{EST}_i \ldots \text{LST}_i \)
- \( d = (d_1, \ldots, d_N) \) is a vector of task duration
- \( r = (r_1, \ldots, r_N) \) is a vector of resource consumption rates
- \( m \) is a scalar

\[ \text{cumulative} (t, d, r, m) \] holds iff

\[ \sum_{i=1}^{N} r_i \leq m \]

\[ t_i \leq t \leq t_i + d_i \]

Filtering algorithms based on disjunctive reasoning
Time-Aware Test Execution Scheduling

\[
\text{cumulative}((t_1, \ldots, t_{10}), (d_1, \ldots, d_{10}), (1, \ldots, 1), 3),
M_1, \ldots, M_6 \text{ in } 1..3,
M_7 = 1, M_8 = 2, M_9 = 3, M_{10} \text{ in } \{1,3\},
(E_2 \leq S_3 \text{ or } E_3 \leq S_2), (E_2 \leq S_4 \text{ or } E_4 \leq S_2),
(E_3 \leq S_4 \text{ or } E_4 \leq S_3),
\max(\text{MaxTime}, (E_1, \ldots, E_{10})),
\text{label(minimize(}\text{MaxTime}, (S_1, \ldots, S_{10}), (M_1, \ldots, M_{10})))
\]

An optimal solution:
\[
S_1 = 0, S_2 = 4, S_3 = 8, S_4 = 0, S_5 = 4, S_6 = 7, S_7 = 2, S_8 = 9, S_{10} = 3,
M_1 = 1, M_2 = 1, M_3 = 1, M_4 = 2, M_5 = 2, M_6 = 2, M_7 = 1, M_8 = 2, M_9 = 3, M_{10} = 3
\]
\[
\text{MaxTime} = 11
\]
Experimental results

But, how to handle priorities and execution history?
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)
RETECS: 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
- No training, very limited memory of past executions → unsupervised ML

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

\[ r_{\text{fail}}^i(t) = |TS_{\text{fail}}^i| \quad (\forall t \in T_i) \]

Reward Function 2. Test Case Failure Reward

\[ r_{\text{tcfail}}^i(t) = \begin{cases} 1 - t.\text{verdict}_i & \text{if } t \in TS_i \\ 0 & \text{otherwise} \end{cases} \]

Reward Function 3. Time-ranked Reward

\[ r_{\text{time}}^i(t) = |TS_{\text{fail}}^i| - t.\text{verdict}_i \times \sum_{t_k \in TS_{\text{fail}}^i, \text{rank}(t) < \text{rank}(t_k)} 1 \]
Lessons Learned and Further Work
Lessons learned

• Industrial Robotics is an interesting application field for automated software testing research

• More automation is highly desired by engineers in industrial robots testing. Release better, release faster, release cheaper
  It’s a highly competitive market!

• Adoption of (robust) AI techniques is possible provided that their benefice is demonstrated on real settings. Validated on real robots.

• Adoption of AI techniques in industrial robotics testing is not easy
  (don’t want to see emerging behaviors or non-deterministic behaviors, good-enough practices, higher cognition for industrial robots is not yet a top-priority!)

A New Battlefield!
Further Work

• Automated Testing of Robot Synchronisation, Multi-Robots interactions

• Human Perception of Robot Safety

• Testing Learning Robots

Thanks to:

Mats Carlsson (SICS, Sweden)
Dusica Marijan (SIMULA, Norway)
Hein Meling (U. of Stavanger, Norway)
Morten Mossige (ABB Robotics, Norway)
Helge Spieker (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.  