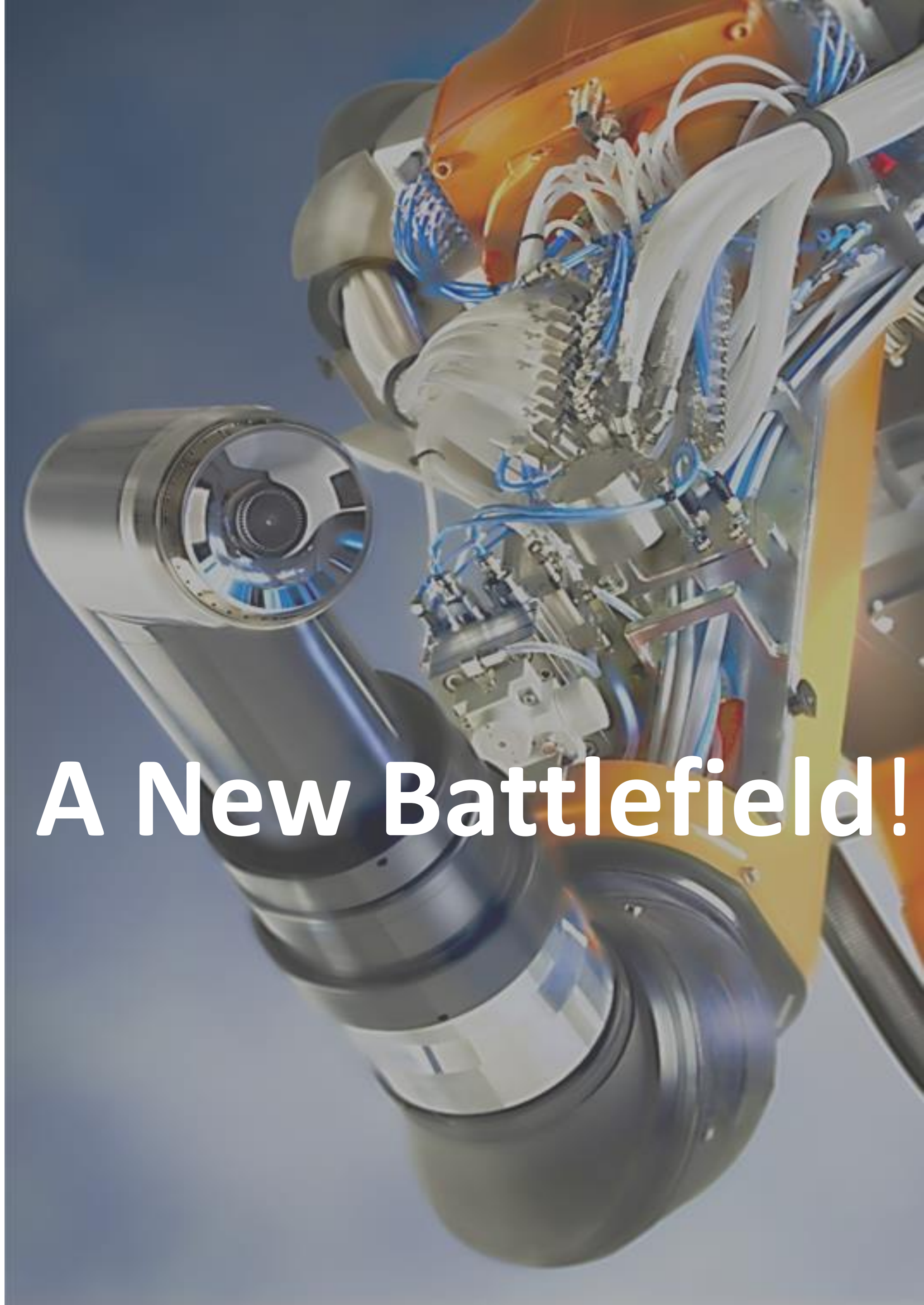
simula

Testing Robotic Systems: A New Battlefield!



Arnaud Gotlieb
Simula Research Laboratory
Lysaker, Norway

GDR-GPL 17 – National Days – 13 June 2017







The Certus Centre

Software Validation and Verification

[simula]

www.certus-sfi.no

Cisco Systems Norway

Cancer Registry of Norway

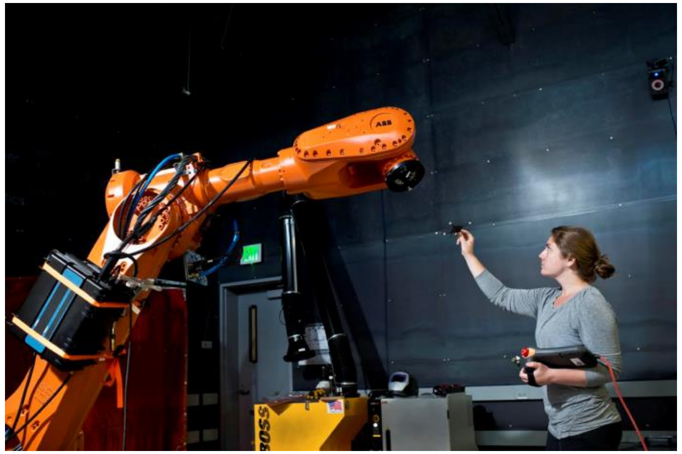
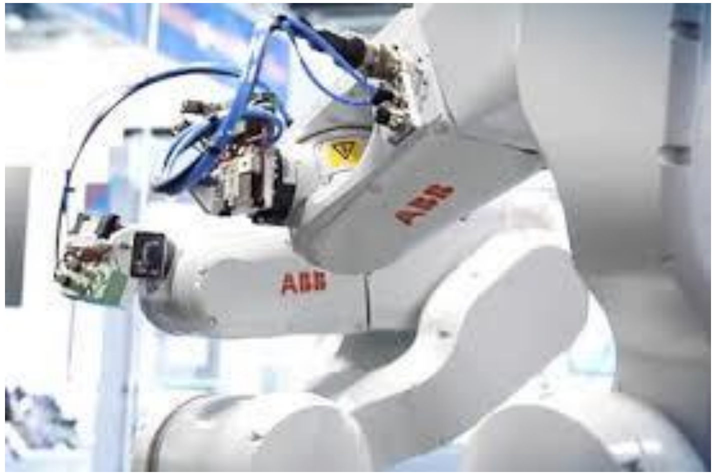


ABB Robotics

Kongsberg Maritime

esito

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...

From....

Single-core, single application system

All source code maintained by a small team located at the same place

Manual system testing only handled in a single place, on actual robots

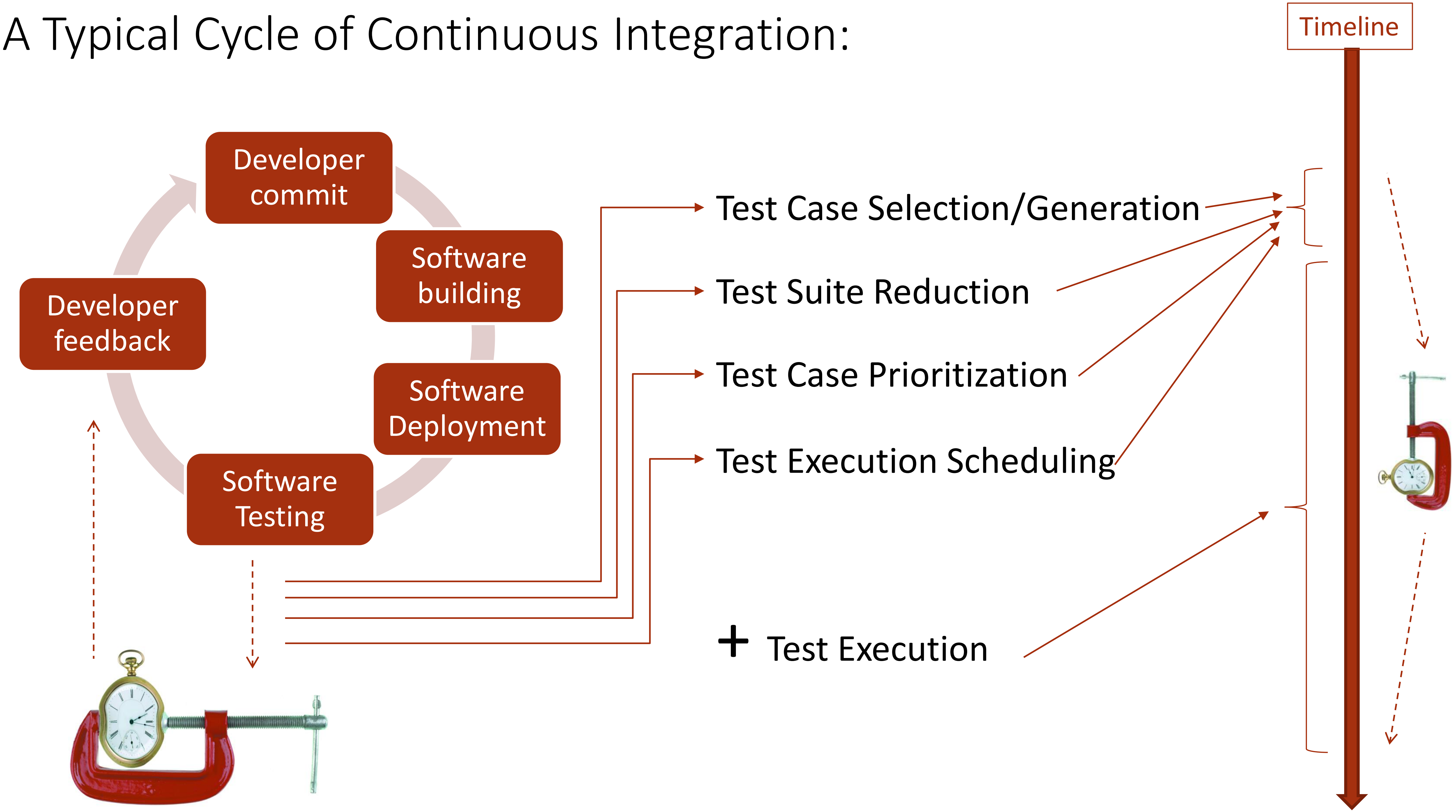
To...

Multi-core, complex distributed system

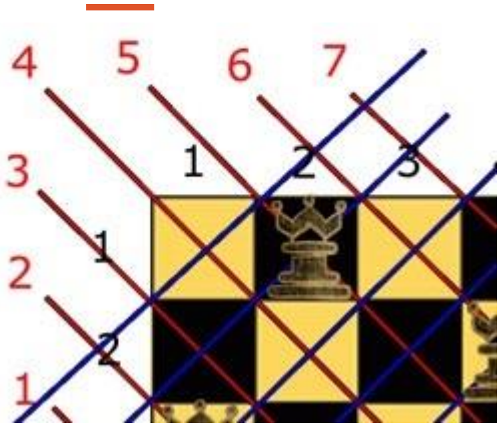
Subsystems developed by distinct teams located at distinct places in the world

Automated software testing handled in a continuous integration process

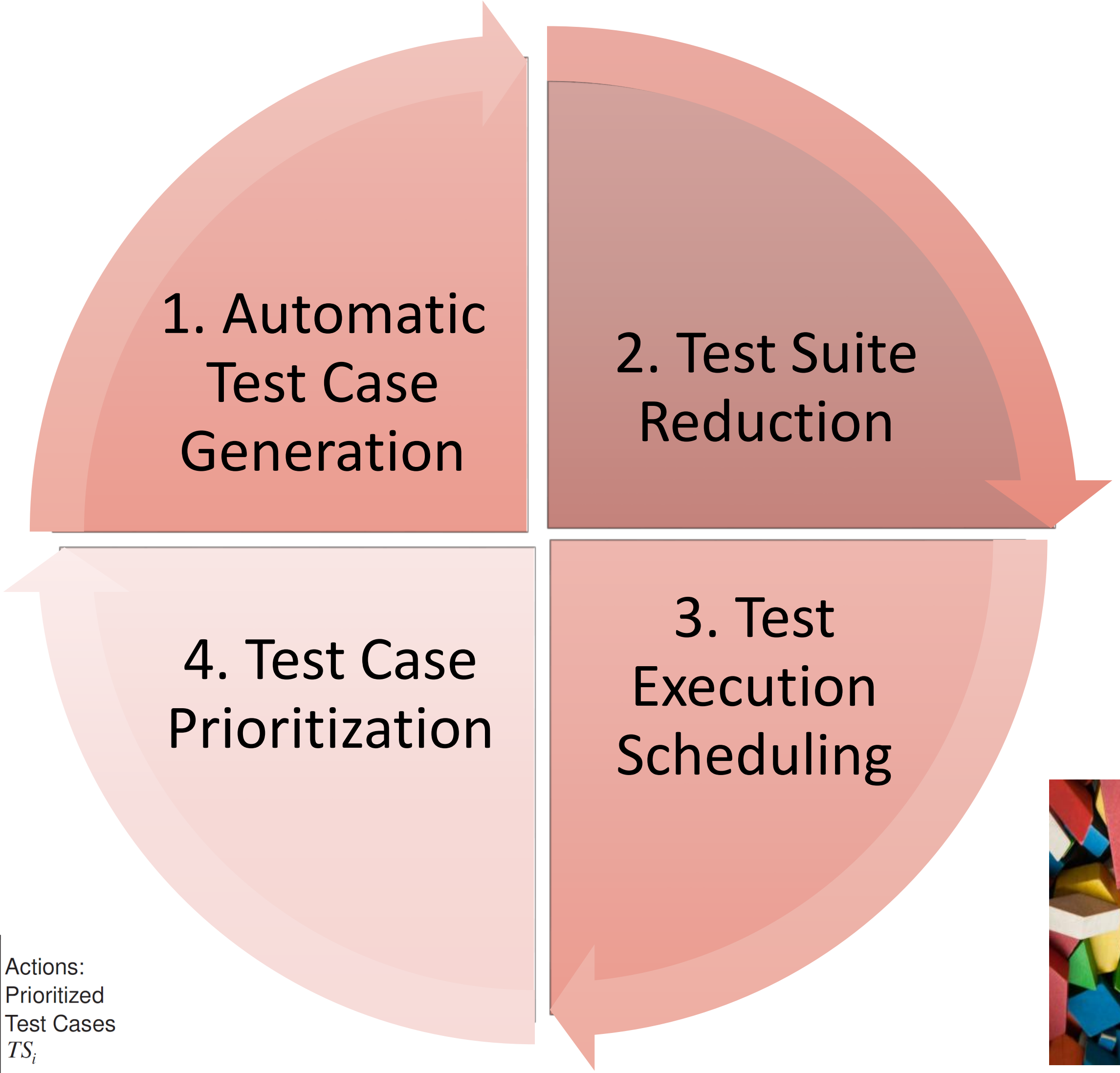
A Typical Cycle of Continuous Integration:

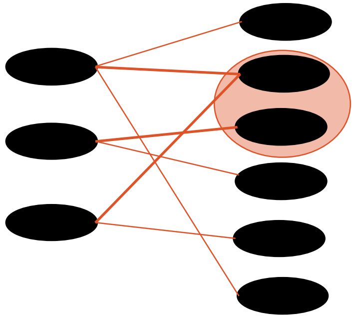


Our Focus : Artificial Intelligence for Testing of Robotic Systems

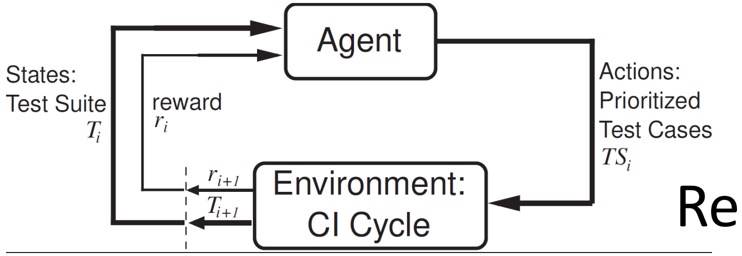


Constraint Modelling

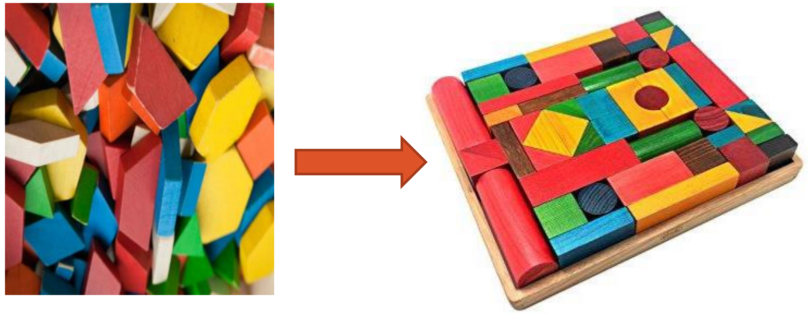




Global Constraints

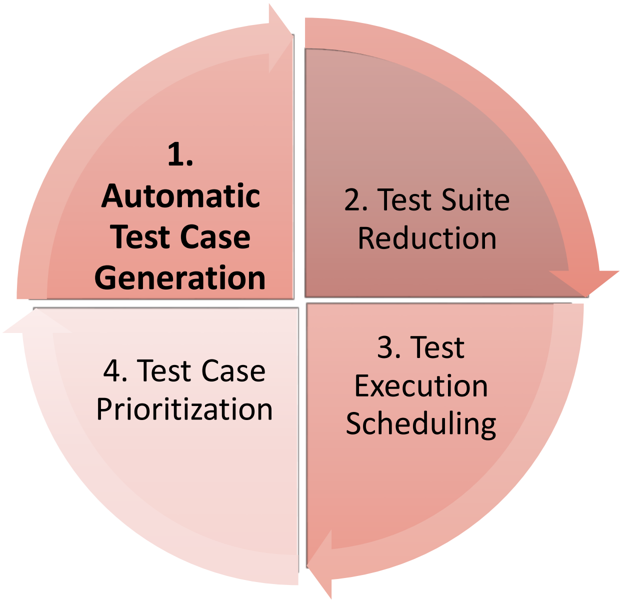


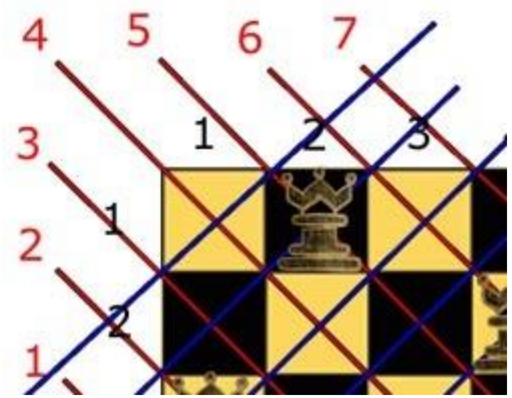
Reinforcement Learning



Constraint-based Scheduling

simula



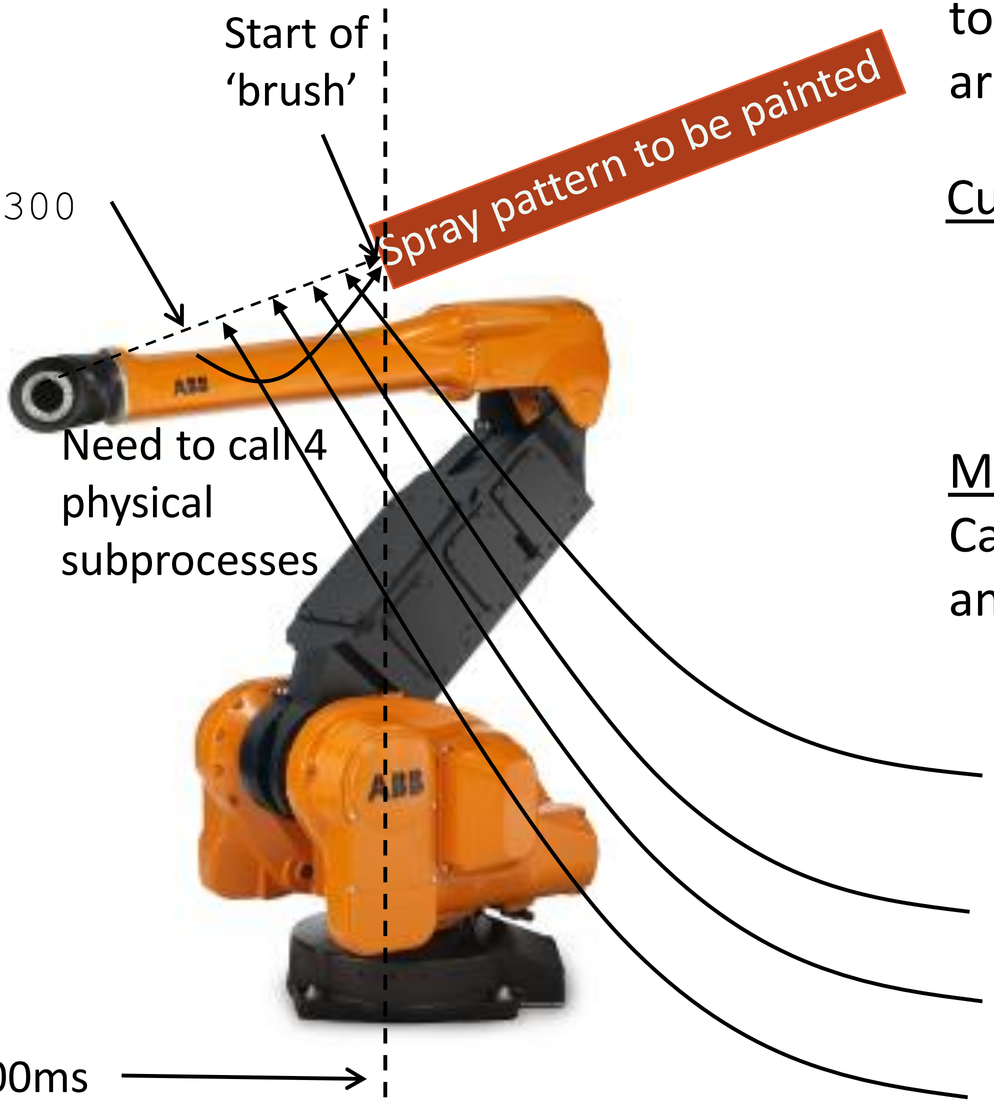


Constraint Modelling

1. Automatic Test Case Generation

simula

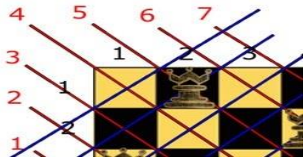
A Typical Robot Painting Scenario



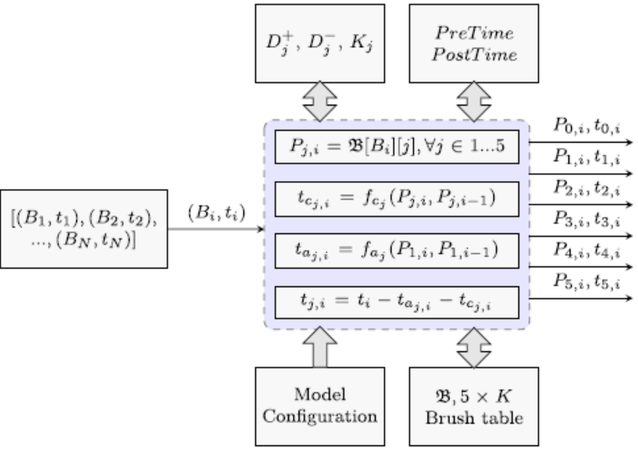
Crucial test objective:
to validate that the four physical outputs are triggered on expected time

Current practice:

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



Constraint Model of IPS



Test oracle

t_t	I/O-1	t_t	I/0-2	t_t	I/O-3
295	75	120	150	205	75
579	500	500	175	585	150
879	75	780	150	881	75
1195	0	1130	0	1231	0

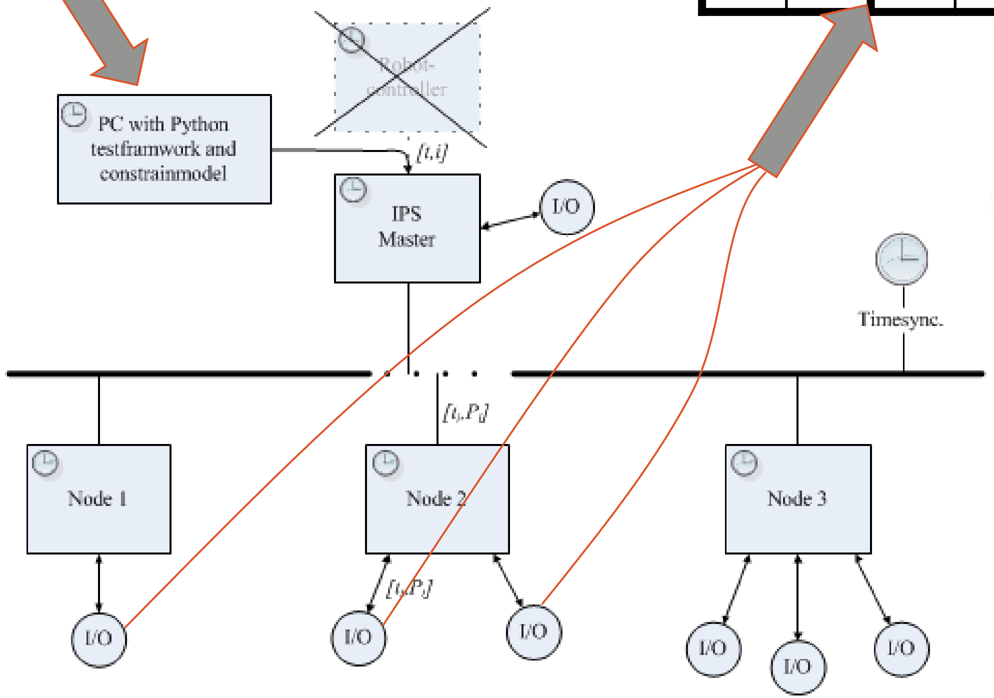
Test sequence

t_i	B_i
300	1
600	2
900	1
1200	0

Compare

Test results

t_t	I/O-1	t_t	I/O-2	t_t	I/O-3
294	75	121	150	205	75
579	500	501	175	585	150
880	75	792	150	880	75
1197	0	1131	0	1232	0



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?

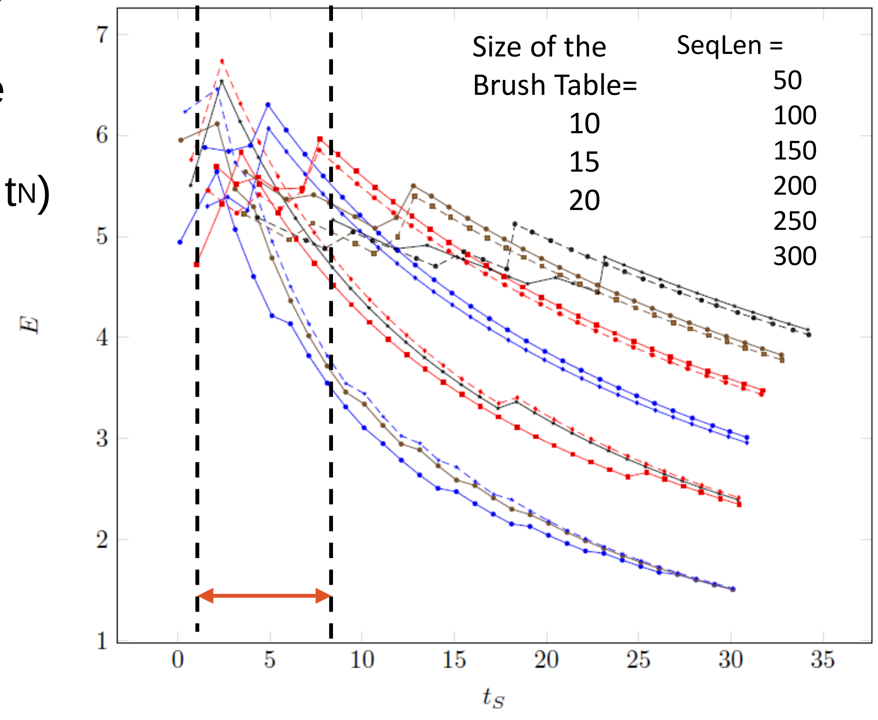
simula

Industrial Deployment

[Mossige et al. CP’14, IST’15]

E: Efficiency factor
t_S : Solving time
t_N : Test exec. time

$E = SeqLen / (t_S + t_N)$



But, still working on maximizing the diversity among test scenarii

SICStus

python

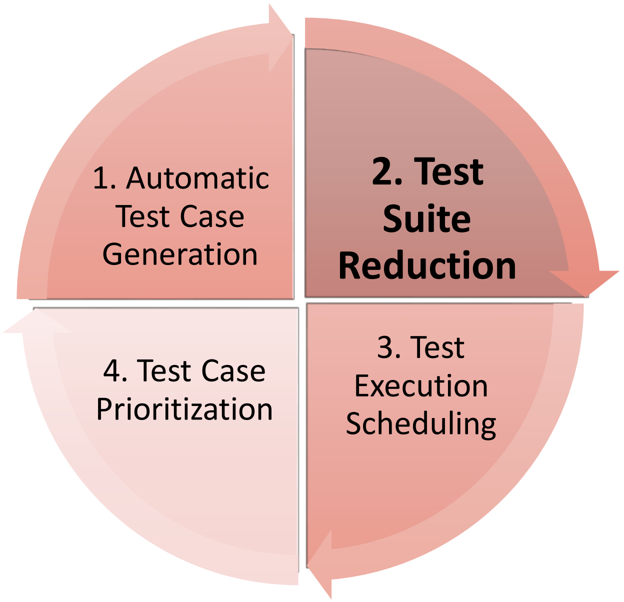
Microsoft Visual Studio Team Foundation Server

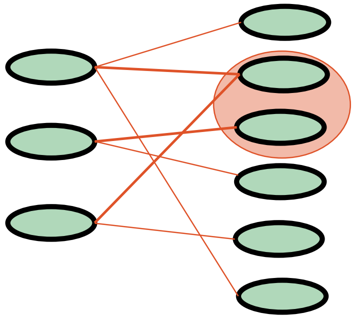
Constraint model: 2KLOC of Prolog, finite domains constraint solver (clpfd + home-made heuristics)

- Time-aware constraint-based optimization
- Integrated throug ABB’s Continuous Integration process
- Constraint model is solved ~15 times per day

- It founds 5 re-introduced (already corrected) critical bugs
- It founds dozens of (non-critical) new bugs

simula





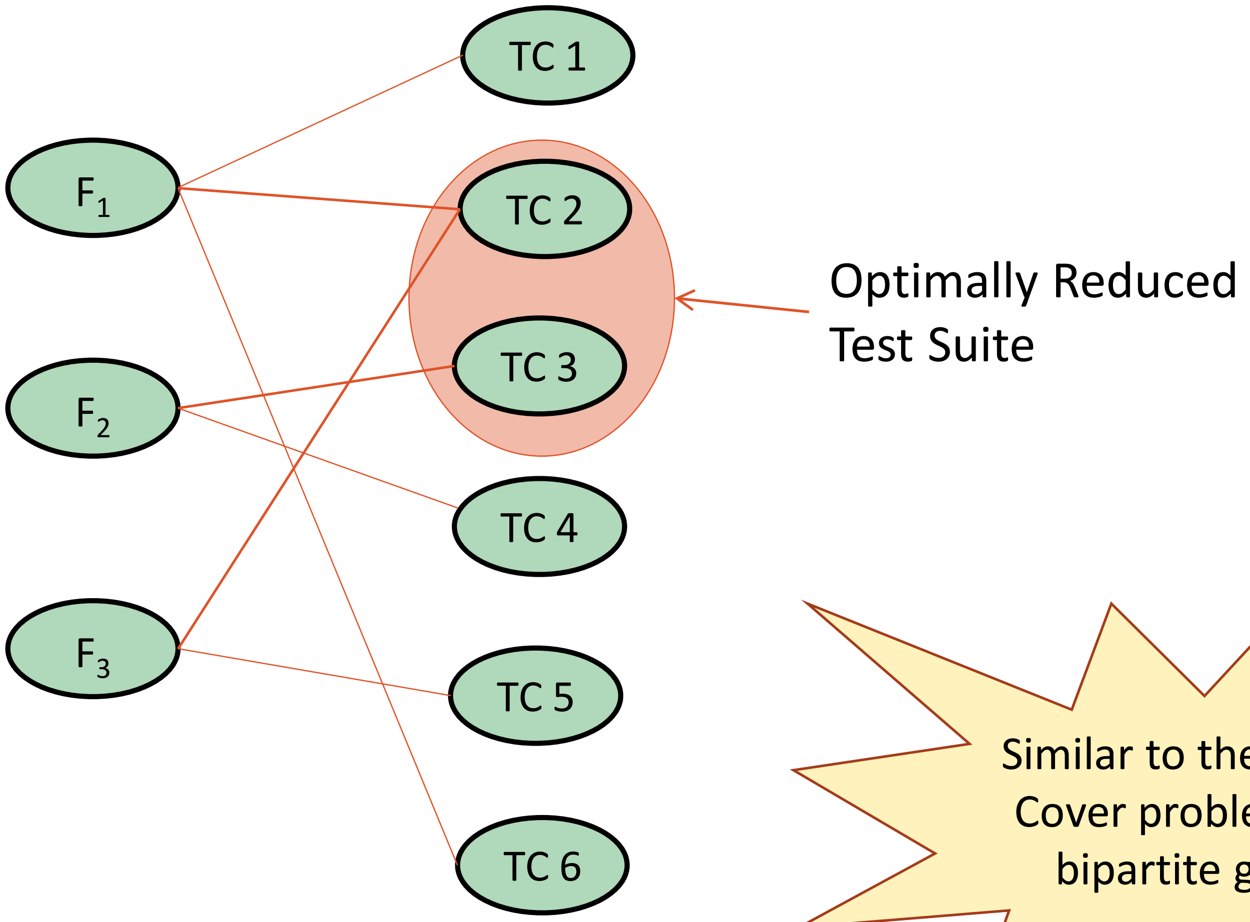
Global Constraints

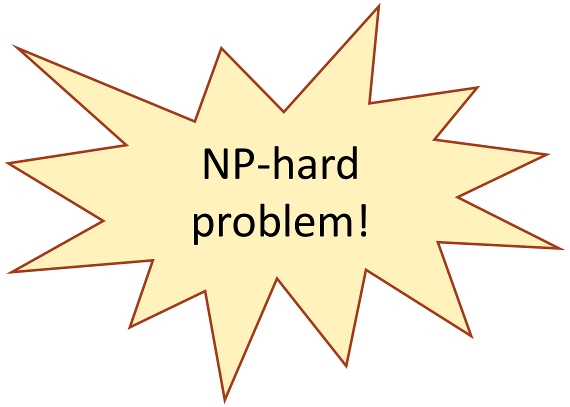
2. Test Suite Reduction

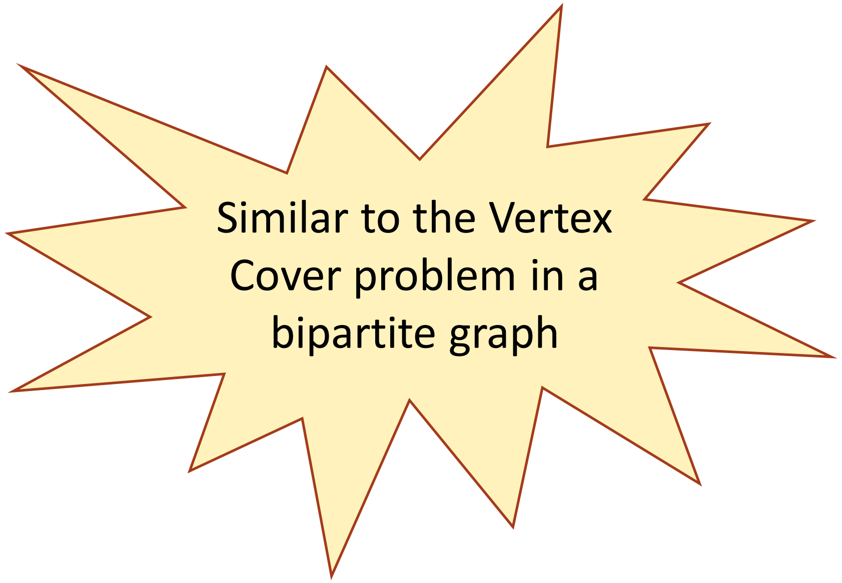
simula

Test Suite Reduction: the core problem

F_i: Features
TC: Test Cases







Test Suite Reduction: existing approaches

- Exact methods: Integer Linear Programming

[Hsu Orso ICSE 2009, Campos Abreu QSIC 2013,…]

Minimize $\sum_{i=1..6} xi$ (minimize the number of test cases)

subject to $\begin{Bmatrix} x_1 + x_2 + x_6 \geq 1 \\ x_3 + x_4 \geq 1 \\ x_2 + x_5 \geq 1 \end{Bmatrix}$ (cover every feature. at least once)

- Approximation algorithms (greedy, search-based methods)

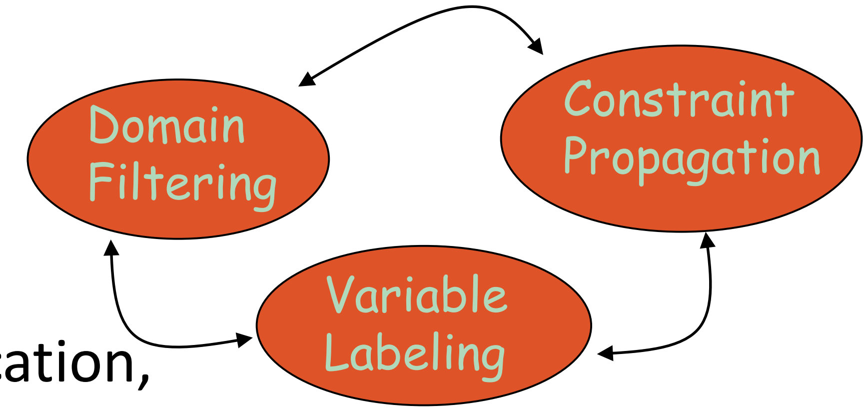
[Harrold et al. TOSEM 1993, …]

```
F = Set of reqs, Current = Ø
    while( Current ǂ F)
        Select a test case that covers the most uncovered features ;
        Add covered features to Current ;
    return Current
```

- Constraint Programming with global constraints *[Gotlieb et al. ISSTA 2014, AI Magazine 2016, …]*

Constraint Programming (CP)

simula



- 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

$V = [V_1, \ldots, V_k]$ is a vector of variables

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

nvalue(N, [3, 1, 3]) entails N = 2

nvalue(3, $[X_1, X_2]$) fails

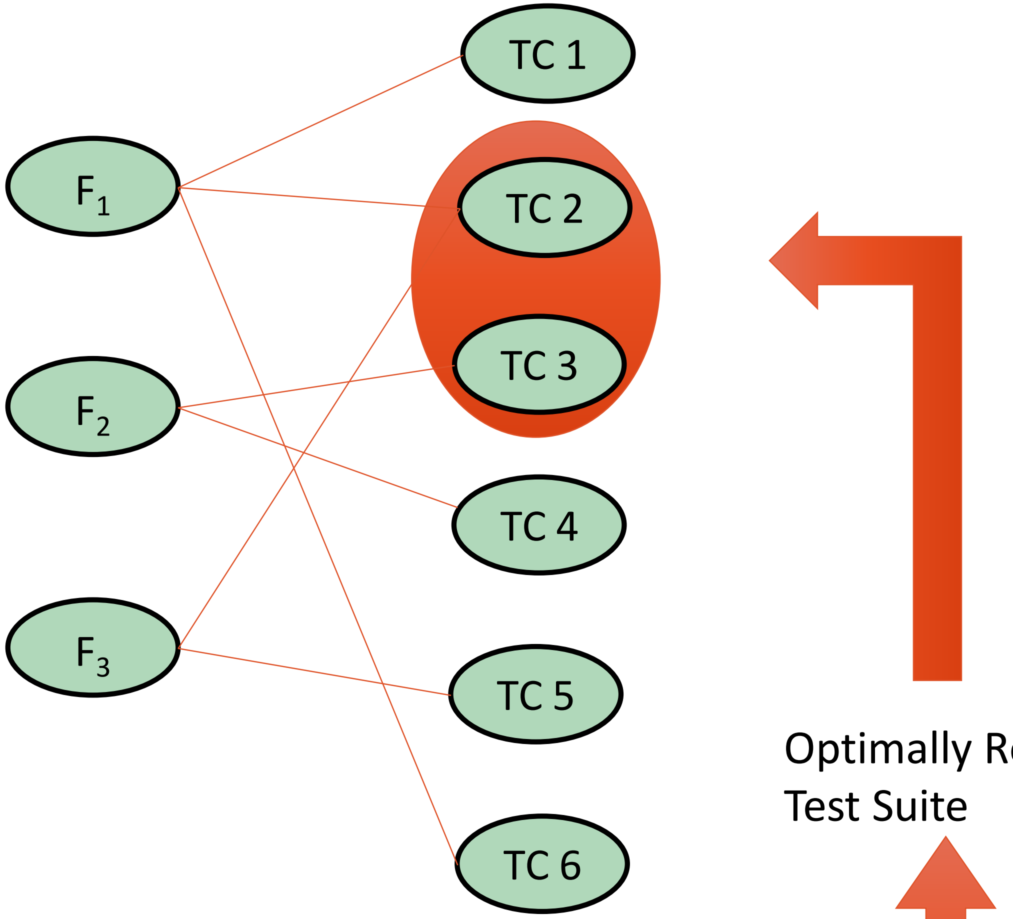
nvalue(1, $[X_1, X_2, X_3]$) entails $X_1 = X_2 = X_3$

N in 1..2, **nvalue**(N, $[4, 7, X_3]$) entails X_3 in {4,7}, N=2

simula

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**)

[Regin AAAI'96]

gcc(T, d, V)

Where

$T = [T_1, ..., T_N]$ is a vector of N variables

$d = [d_1, ..., d_k]$ is a vector of k values

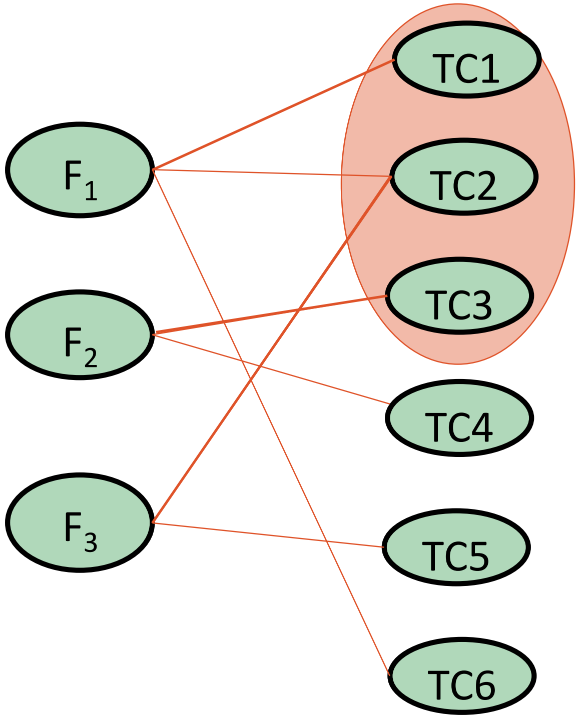
$V = [V_1, ..., V_k]$ is a vector of k variables

gcc(T, d, V) holds iff $\forall i \; in \; 1..k,$ $V_i = card(\{j \mid T_j = di\})$

Filtering algorithms for **gcc** are based on max flow computations

simula

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:

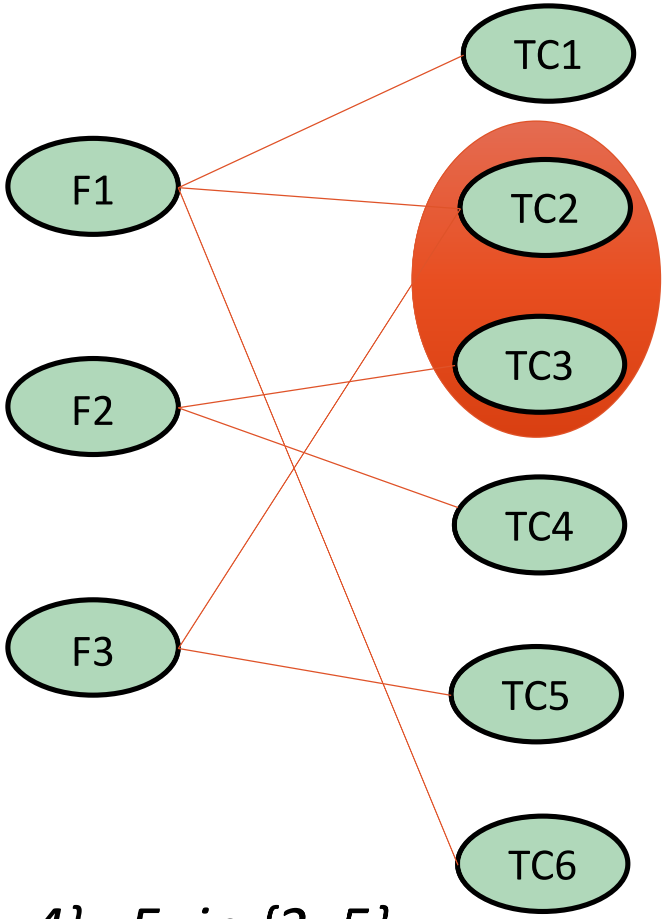
In the solution-set,
TC1 is used to cover exactly V_1 features in [F_1, F_2, F_3]
TC2 " V_2 "
TC3 " V_3 "
...

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 in {1, 2, 6}, F_2 in {3, 4}, F_3 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(MaxNvalue, [F_1, F_2, F_3])

Minimize(MaxNvalue)

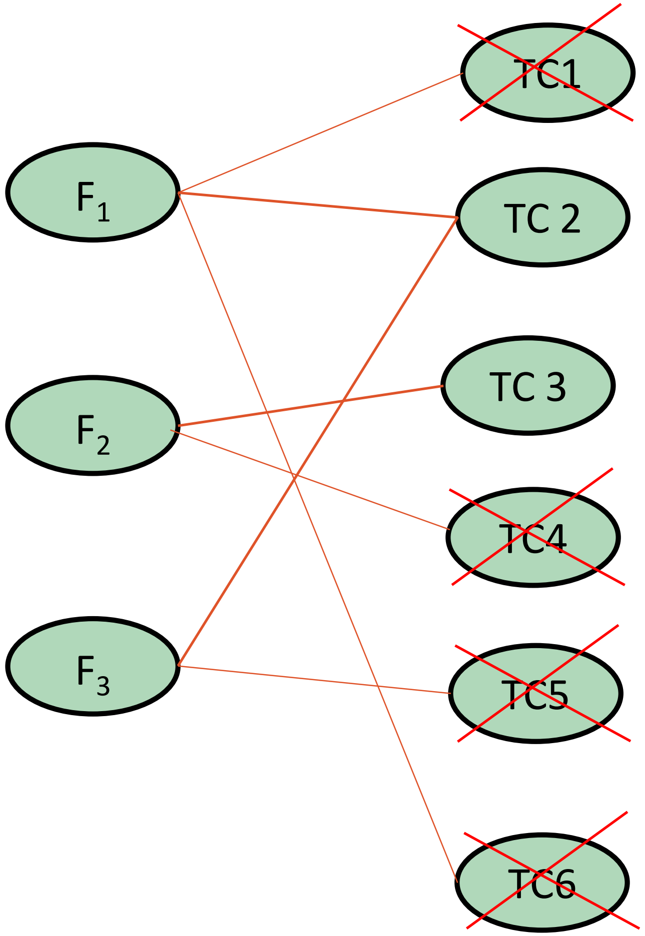
Model pre-processing

F_1 in {1, 2, 6} → $F_1 = 2$
as $cov(TC_1) \subset cov(TC_2)$ and $cov(TC_6) \subset cov(TC_2)$
withdraw TC_1 and TC_6

F_3 is covered → withdraw TC_5

F_2 in {3,4} → e.g., $F_2 = 3$, withdraw TC_4

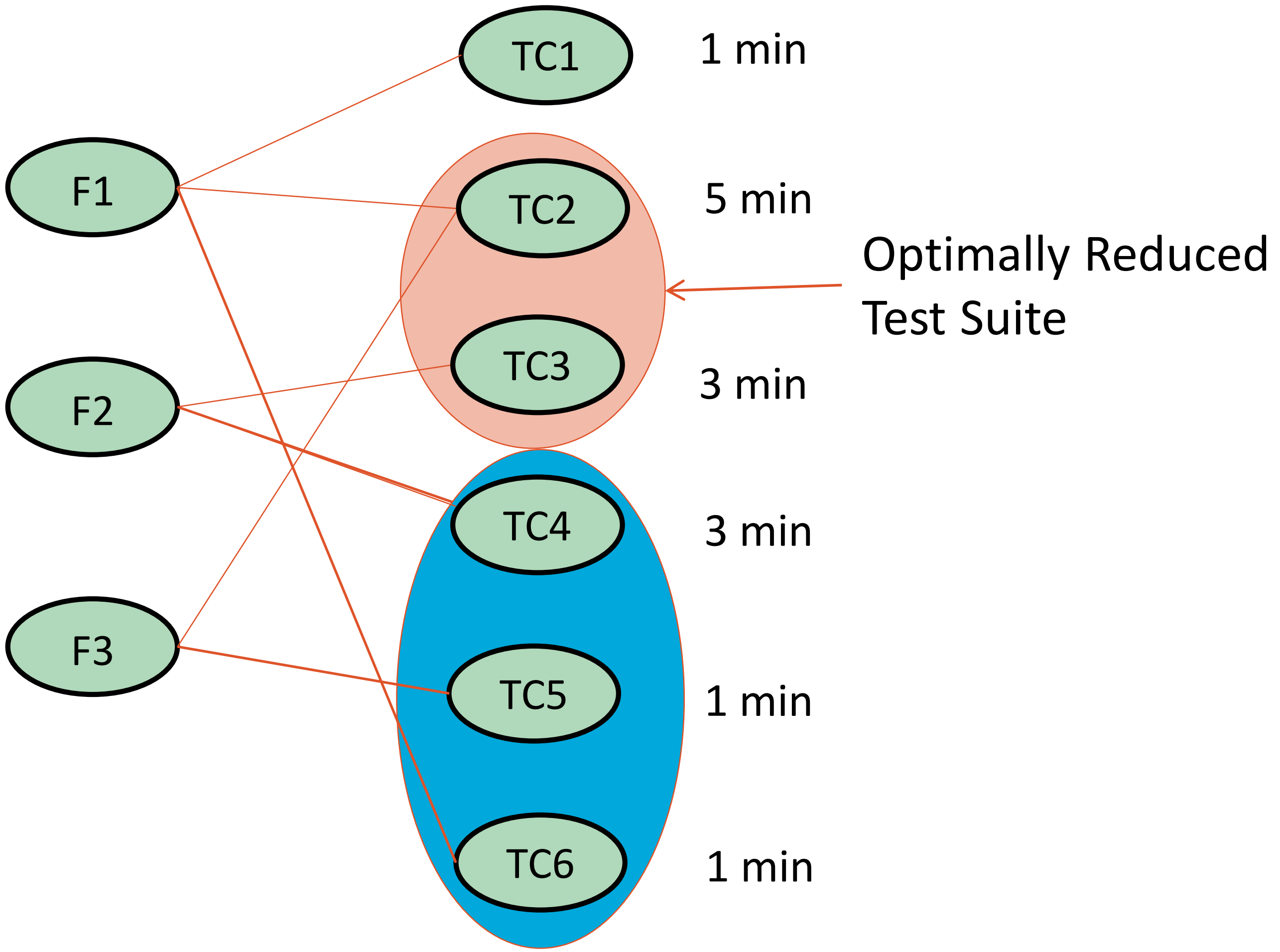
Pre-processing rules can be expressed once and then applied iteratively



simula

Other criteria to minimize

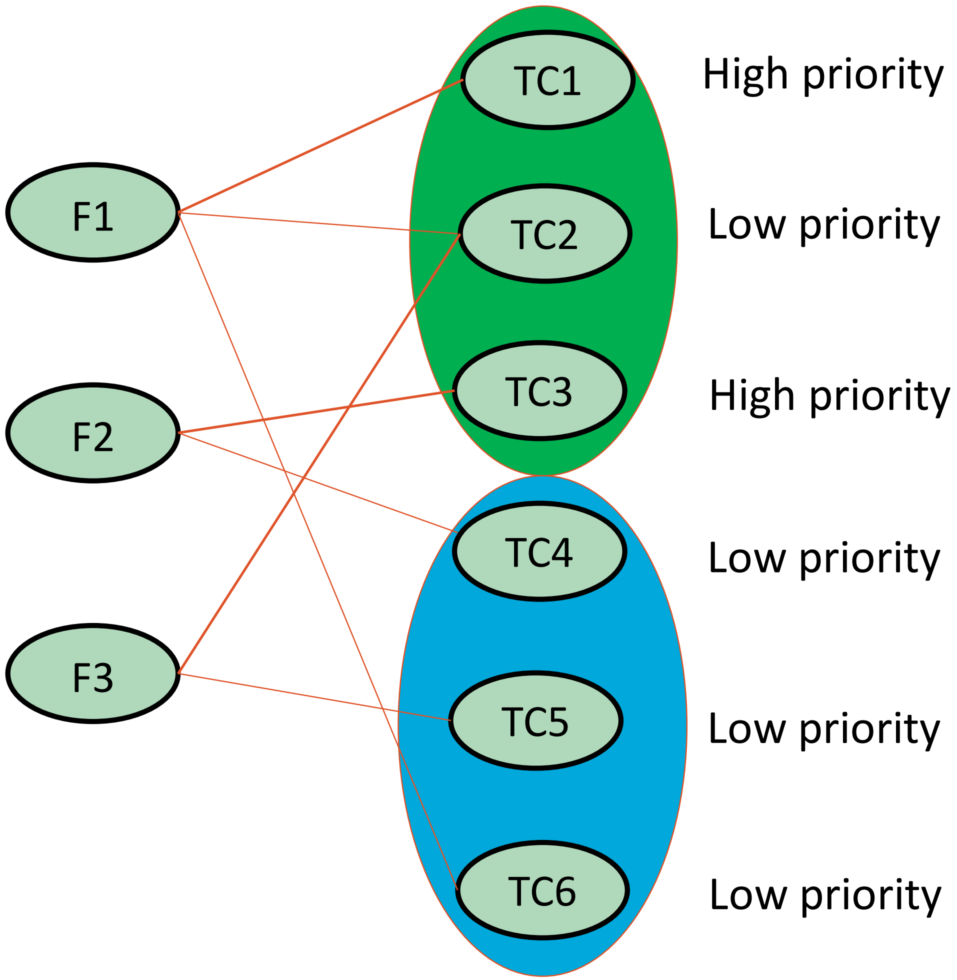
Feature coverage
is always a prerequiste



Execution time!

Other criteria to minimize

Feature coverage
is always a prerequiste



Fault revealing capabilities!

simula

Proposed approaches

1. Actual multi-objectives optimization with search-based algorithms (Pareto Front) *[Wang et al. JSS'15]*

   Aggregated cost function using weights for each objective

   Approximate solutions
   No constraint model!

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

   Exact solutions
   Constrained optimization model!

simula

Optimal Test Suite Reduction with Costs

[Gotlieb et al. ICSOFT-EA'16]

$F_1,..,F_n$: Features
$t_1,..,t_m$: Test cases
$c_1,..,c_m$: Unit cost for each test case

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

Minimize TotalCost

s.t

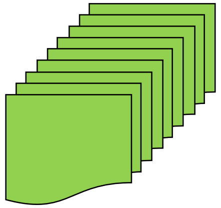
gcc($[F_1, …, F_n], [t_1, …, t_m], [O_1, …, O_m]$)
for i=1 to m do $B_i = (O_i > 0)$
scalar_product($[B_1, …, B_m], [c_1, …, c_m]$, TotalCost)

where **scalar_product** encodes $B_1*c_1 + .. + B_m*c_m$ = TotalCost

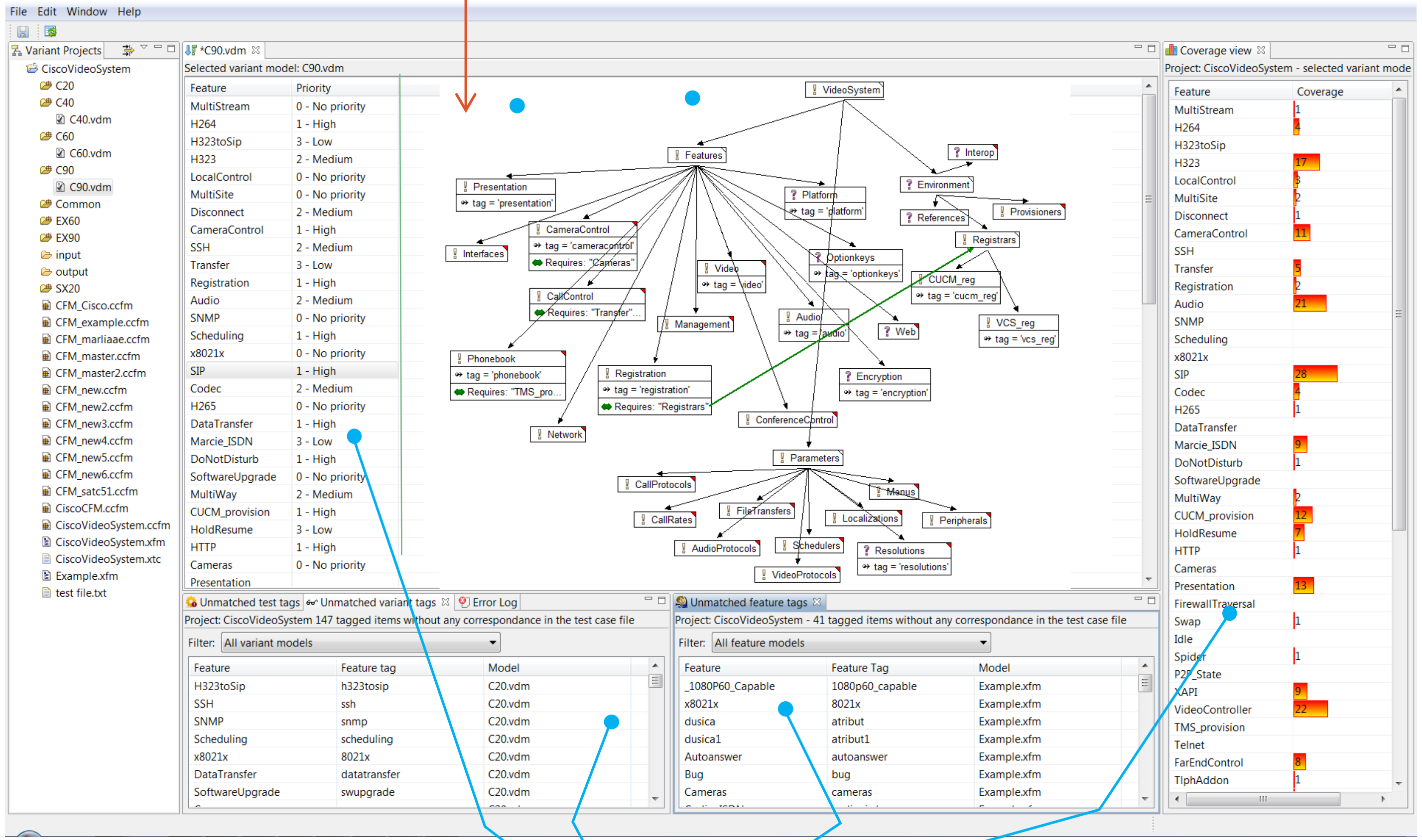
simula

TITAN [Marijan, Gotlieb ICST'17]

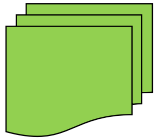
Variability model to describe a product line



Unoptimized test suite



Optimized (reduced) test suite

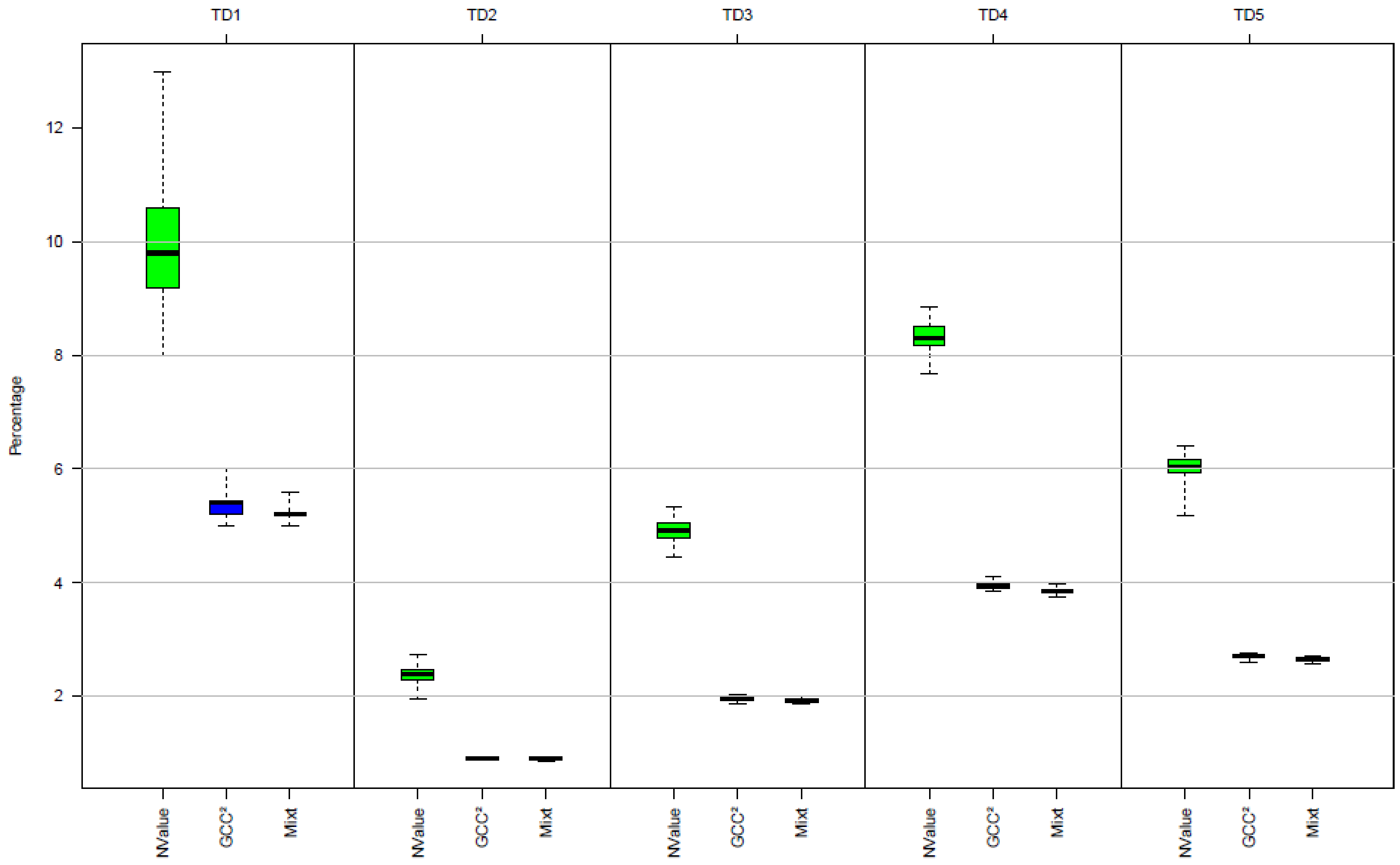


Diagnostic views, feature coverage

simula

Model comparison on random instances (uniform costs)

(Reduced Test Suite percentage in 30sec of search)

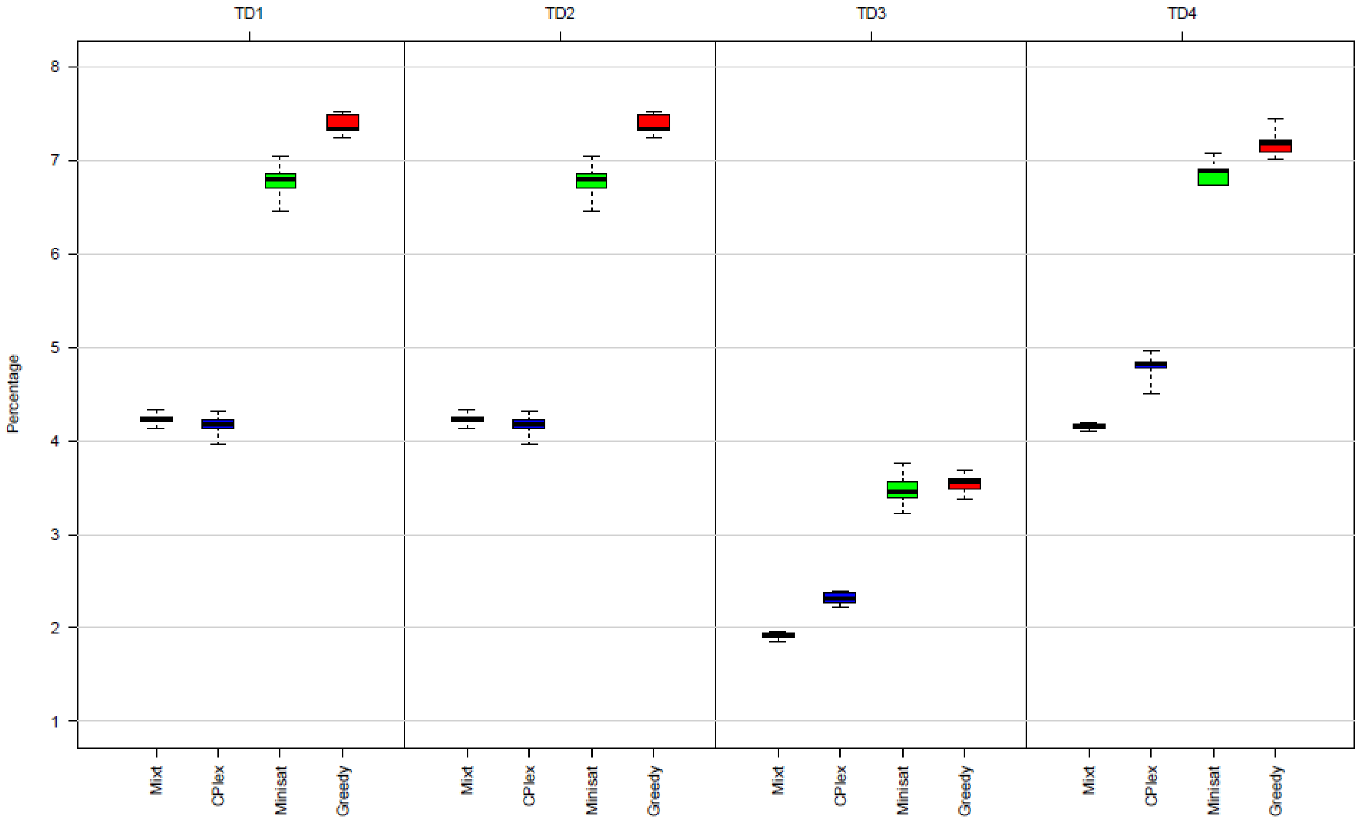


	TD1	TD2	TD3	TD4	TD5
Requirements	250	500	1000	1000	1000
Test cases	500	5000	5000	5000	7000
Density	20	20	20	8	8

simula

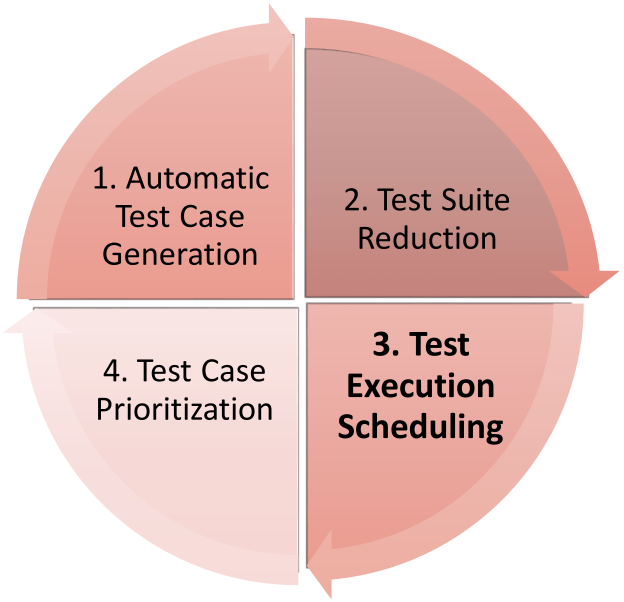
Comparison with CPLEX, MiniSAT, Greedy (uniform costs)

(Reduced Test Suite percentage in 60 sec)

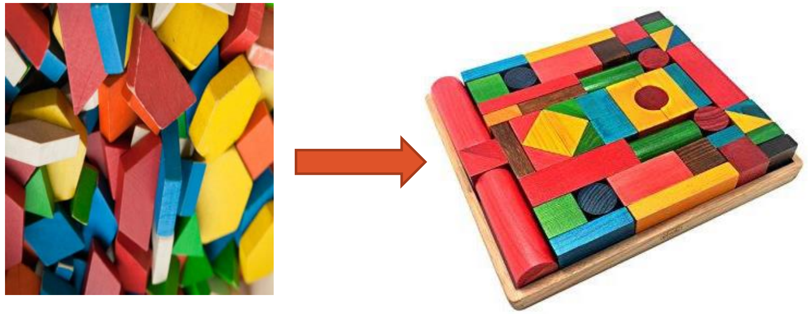


But, less encouraging results when non-uniform costs are used! (CPLEX always better than TITAN)

	TD1	TD2	TD3	TD4
Requirements	1000	1000	1000	2000
Test cases	5000	5000	5000	5000
Density	7	7	20	20



3. Test Execution Scheduling



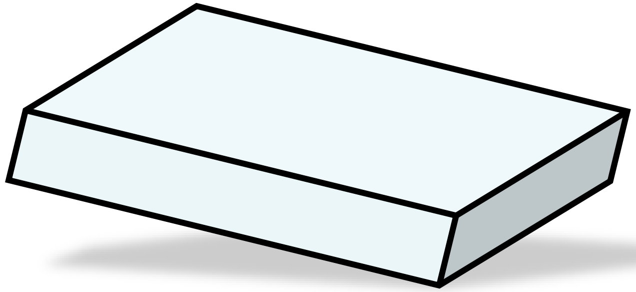
Constraint-based Scheduling

Test Execution Scheduling

Test Cases
with distinct
characteristics

Schedule

Assignment of Test Cases
to Agents which
1. Satisfies capacity constraints
2. Optimize [illegible] me cost function



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

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

simula

Constraint Models for Test Scheduling

ABB

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

10..30 code changes per Day

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

1	Deployed at ABB / « good enough »
2	Evaluated / Needs Improvements
3	Evaluation in progress / Not yet deployed

T2, T5, T34

T3, T6, T45, T78

T7, T23

T4, T56, T67

T45, T55






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 → 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 card(m) Test Cases can be executed at any moment ;

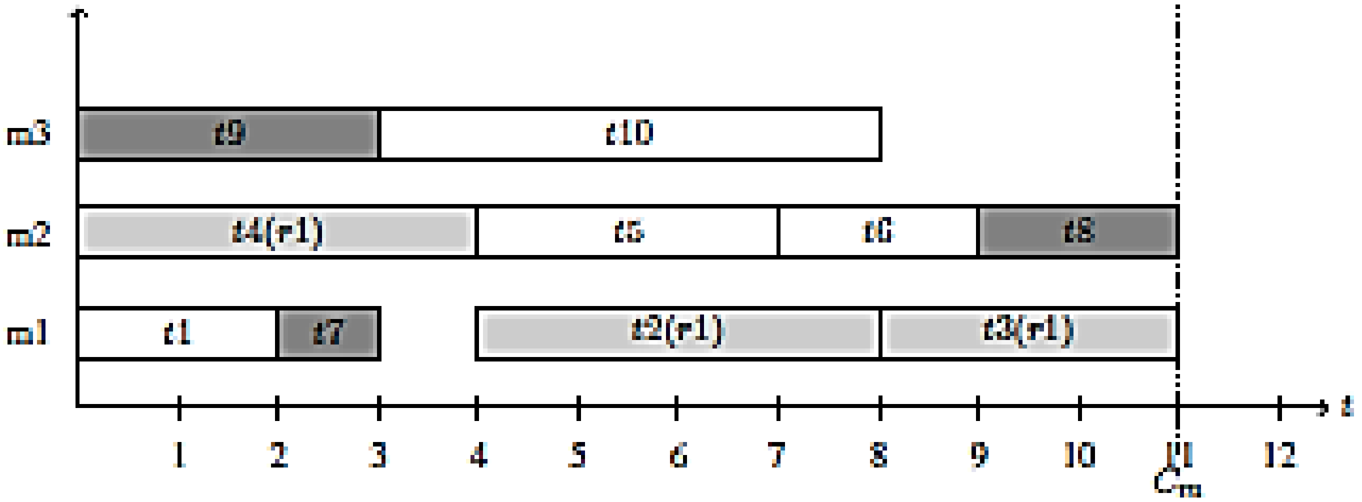
Function to optimize:

- *Timespan*: the overall duration of the schedule (in order to minimize the round-trip time)

NP-hard problem!

A realistic example

Test	Duration	Executable on	Use of global resource
$t1$	2	$m1, m2, m3$	-
$t2$	4	$m1, m2, m3$	$r1$
$t3$	3	$m1, m2, m3$	$r1$
$t4$	4	$m1, m2, m3$	$r1$
$t5$	3	$m1, m2, m3$	-
$t6$	2	$m1, m2, m3$	-
$t7$	1	$m1$	-
$t8$	2	$m2$	-
$t9$	3	$m3$	-
$t10$	5	$m1, m3$	-



simula

The cumulative global constraint [Aggoun & Beldiceanu AAAI'93]

cumulative(t, d, r, m)

Where

$t = (t_1, \ldots, t_N)$ is a vector of tasks, each t_i in $EST_i .. LST_i$

$d = (d_1, \ldots, d_N)$ is a vector of task duration

$r = (r_1, \ldots, r_N)$ is a vector of resource consumption rates

m is a scalar

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

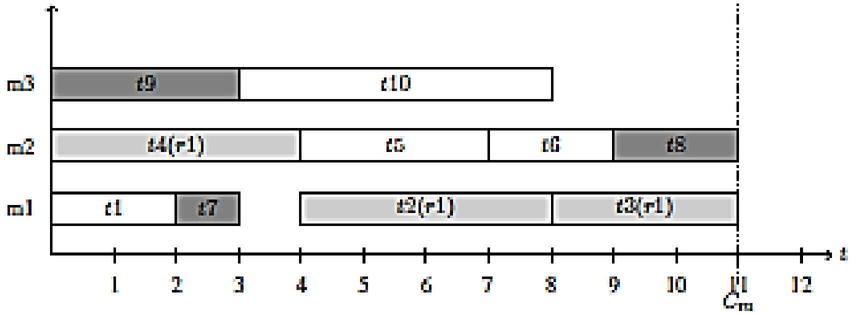
simula

Time-Aware Test Execution Scheduling

[Mossige et al. CP 2017]

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

Test	Duration	Executable on	Use of global resource
$t1$	2	$m1, m2, m3$	-
$t2$	4	$m1, m2, m3$	$r1$
$t3$	3	$m1, m2, m3$	$r1$
$t4$	4	$m1, m2, m3$	$r1$
$t5$	3	$m1, m2, m3$	-
$t6$	2	$m1, m2, m3$	-
$t7$	1	$m1$	-
$t8$	2	$m2$	-
$t9$	3	$m3$	-
$t10$	5	$m1, m3$	-



<u>An optimal solution:</u>

$S_1 = 0$, $S_2 = 4$, $S_3 = 8$, $S_4 = 0$, $S_5 = 4$, $S_6 = 7$, $S_7 = 2$, $S_8 = 9$, $S_{10} = 3$,
$M_1 = 1$, $M_2 = 1$, $M_3 = 1$, $M_4 = 2$, $M_5 = 2$, $M_6 = 2$, $M_7 = 1$, $M_8 = 2$, $M_9 = 3$, $M_{10} = 3$
MaxTime = 11

simula

Experimental results

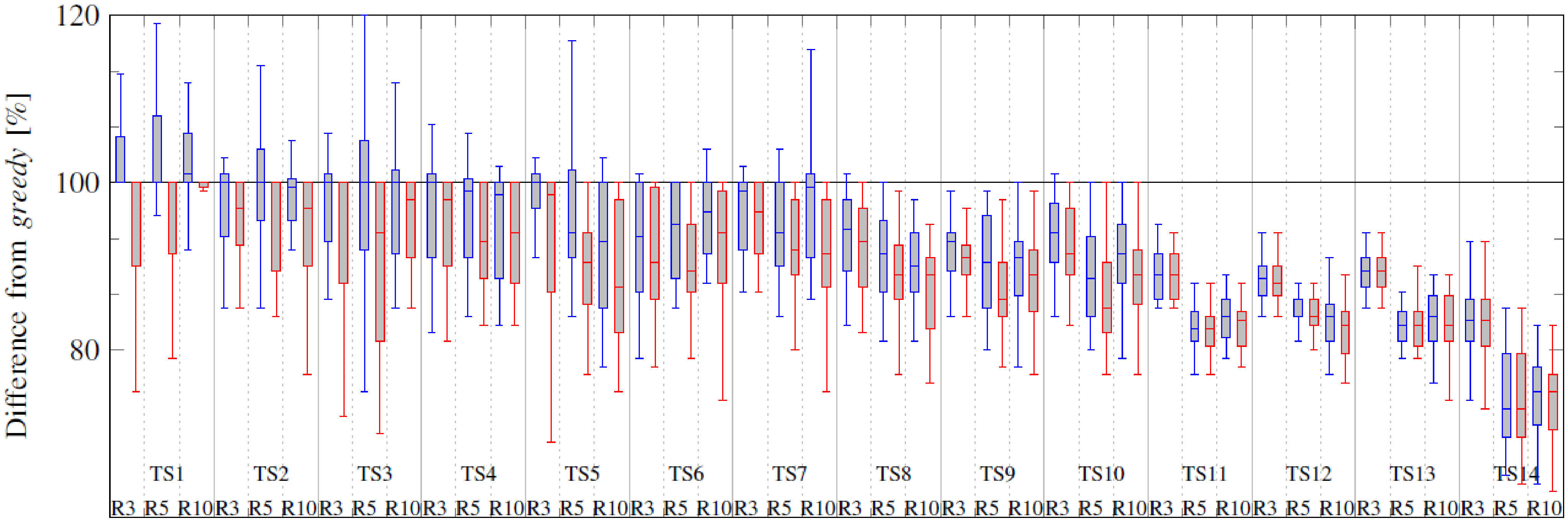
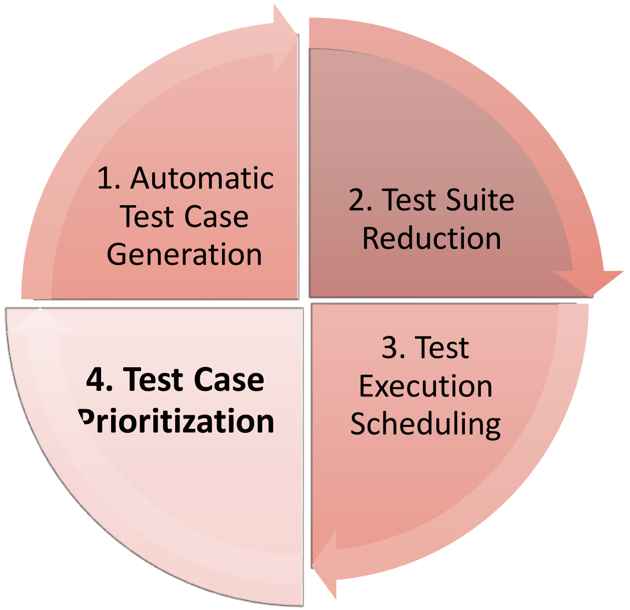


Fig. 5. The differences in schedule execution times produced by the different methods for test suites TS1–TS14, with *greedy* as the baseline of 100%. The blue is the difference between C_f^* and *greedy* and the red shows the difference between C_l^* and *greedy*.

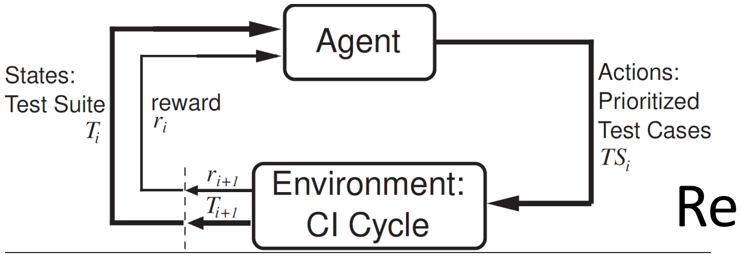
# machines	# of tests	20	30	40	50	100	500
	100	-	-	-	-	-	TS11
	50	-	-	-	-	TS8	TS12
	20	-	TS2	TS4	TS6	TS9	TS13
	10	TS1	TS3	TS5	TS7	TS10	TS14

But, how to handle priorities and execution history ?

simula



4. Test Case Prioritization



Reinforcement Learning

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

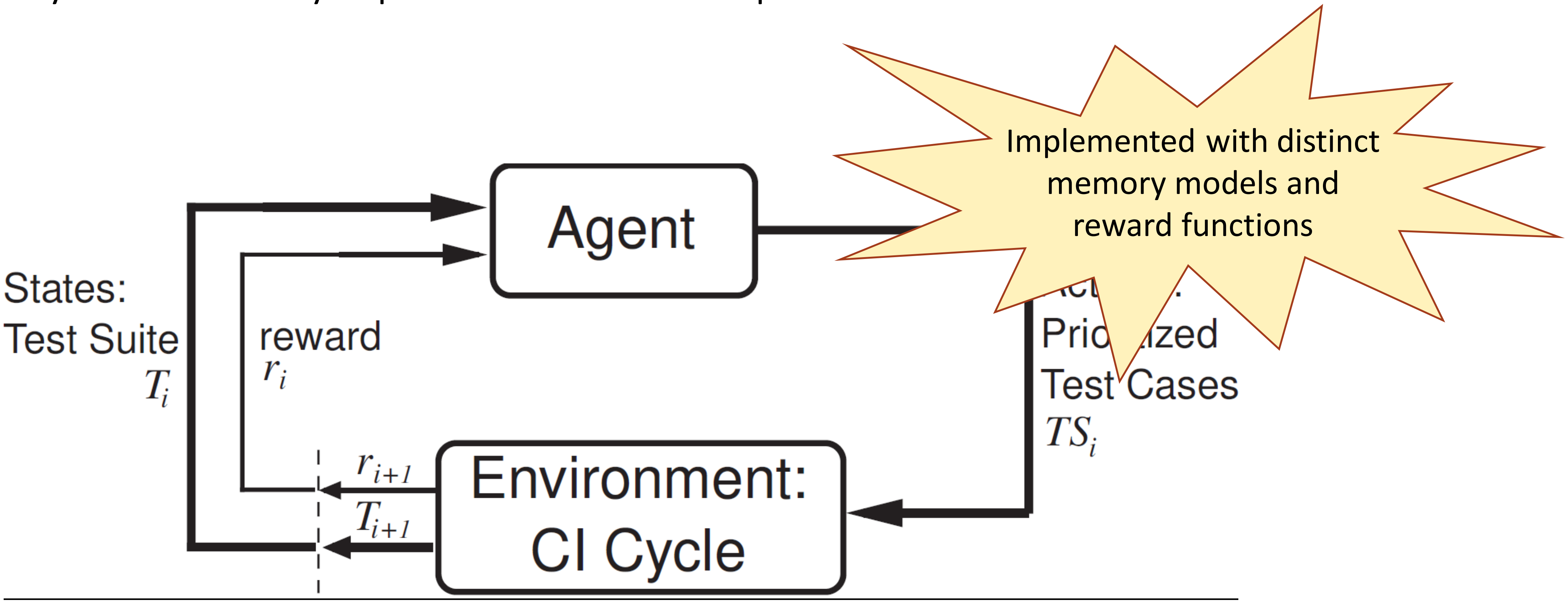
- Adapt testing to focus on the more error-prone parts of the tested system
- Adapt testing to the execution environment (available robots and devices, limited testing time and resources, experiences from previous cycles in continuous integration)

Robot learning different testing techniques

simula

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



simula

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

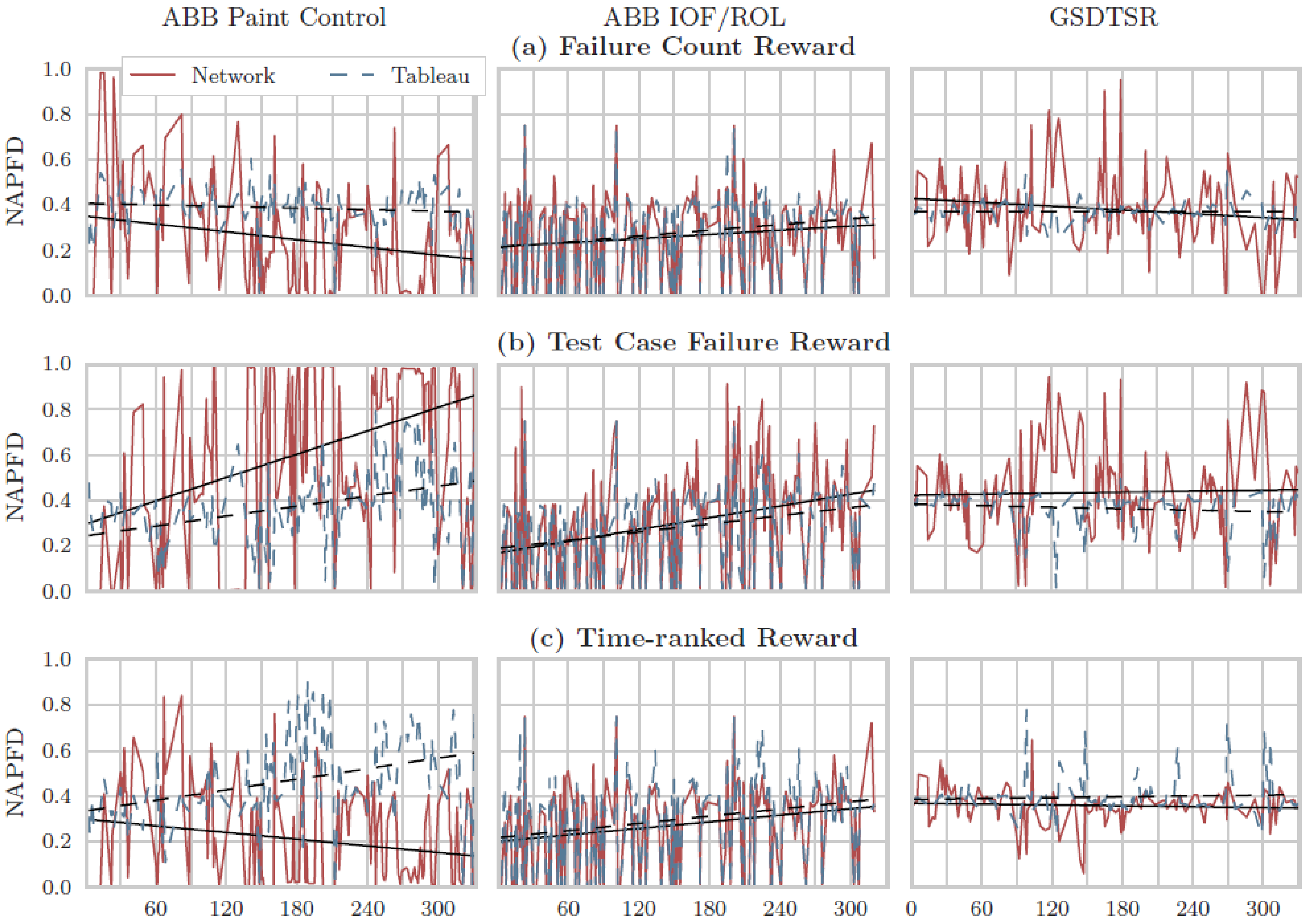
$$reward_i^{fail}(t) = |\mathcal{TS}_i^{fail}| \qquad (\forall\, t \in \mathcal{T}_i)$$

Reward Function 2. Test Case Failure Reward

$$reward_i^{tcfail}(t) = \begin{cases} 1 - t.verdict_i & \text{if } t \in \mathcal{TS}_i \\ 0 & \text{otherwise} \end{cases}$$

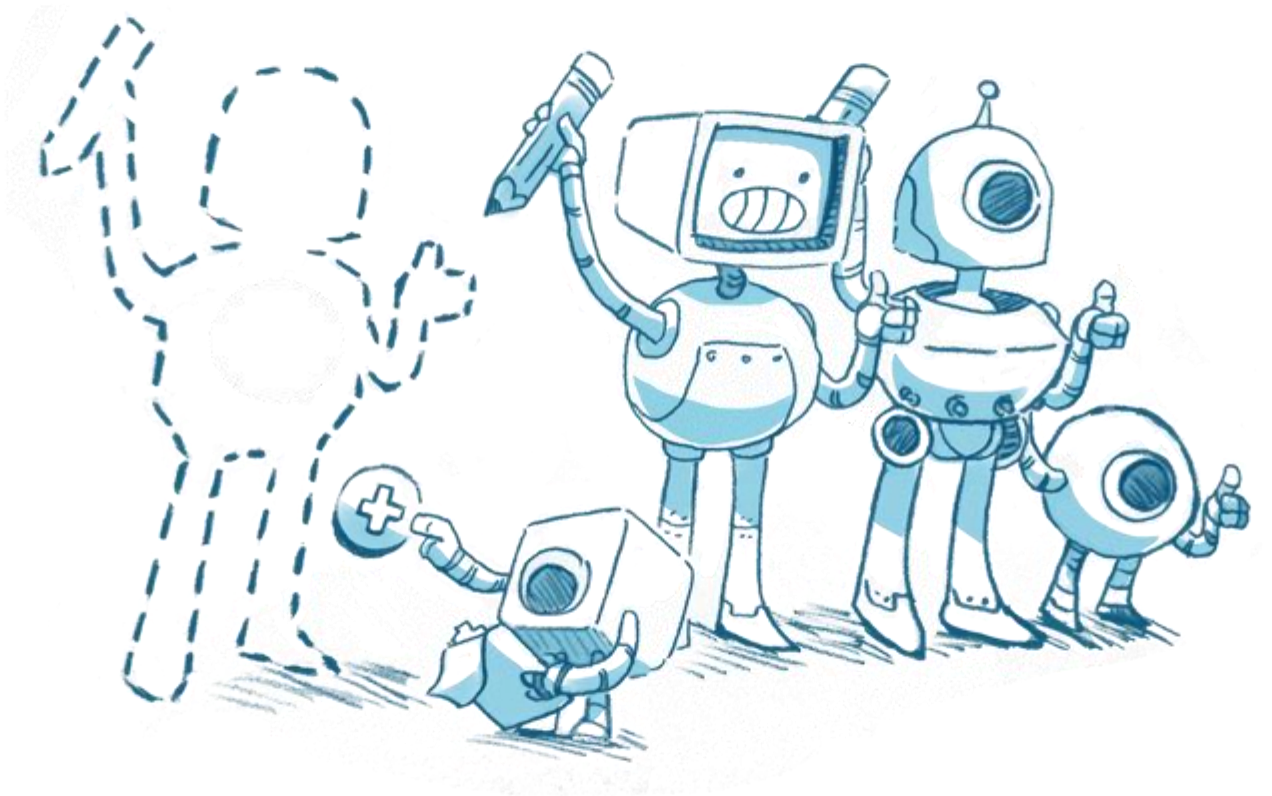
Reward Function 3. Time-ranked Reward

$$reward_i^{time}(t) = |\mathcal{TS}_i^{fail}| - t.verdict_i \times \sum_{\substack{t_k \in \mathcal{TS}_i^{fail} \wedge \\ rank(t) < rank(t_k)}} 1$$



simula

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

1. [Spieker et al. 2017] H. Spieker, A. Gotlieb, D. Marijan and M. Mossige
   Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration
   In Proc. of 26th Int. Symp. on Soft. Testing and Analysis (ISSTA-17), Santa Barbara, USA, July 2017.
2. [Gotlieb Marijan 2017] A. Gotlieb and D. Marijan
   Using Global Constraints to Automate Regression Testing AI Magazine 38, Spring, 2017.
3. [Marijan et al. 2017] D. Marijan, A. Gotlieb, M. Liaaen, S. Sen and C. Ieva
   TITAN: Test Suite Optimization for Highly Configurable Software
   In Int. Conf. on Soft. Testing, Verification and Validation (ICST-17), Tools Track, Tokyo, Japan, 2017.
4. [Mossige et al. 2017] M.Mossige, A. Gotlieb, H. Spieker, H. Meling, M. Carlsson
   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**
6. [Mossige et al., 2015] M. Mossige, A. Gotlieb, and H. Meling.
   Testing robot controllers using constraint programming and continuous integration.
   Information and Software Technology, 57:169-185, Jan. 2015.
7. [Wang et al., 2015] S. Wang, S. Ali, and A. Gotlieb.
   Cost-effective test suite minimization in product lines using search techniques.
   Journal of Systems and Software 103: 370-391, 2015.
8. [Gotlieb et al., 2014] A. Gotlieb and D. Marijan.
   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.
9. [Mossige et al., 2014] M. Mossige, A. Gotlieb, and H. Meling.
   Using CP in automatic test generation for ABB robotics' paint control system.
   In Principles and Practice of Constraint Programming (CP-14) – **Awarded Best Application Paper**, Lyon, Fr., Sep. 2014.