# Global Constraints in Software Testing Applications

Arnaud Gotlieb
Certus Centre
Simula Research Laboratory
Norway







# The Certus Centre

Software Validation and Verification

Hosted by SIMULA

Established and awarded SFI in Oct. 2011

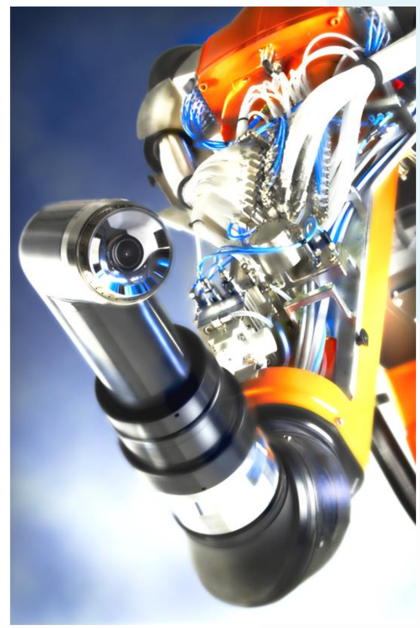
duration: 8 years

[www.certus-sfi.no](http://www.certus-sfi.no)

Cisco Systems Norway

Norwegian Custom and excise

ABB Robotics Stavanger



Kongsberg Maritime

esito

[ simula . research laboratory ]
- by thinking constantly about it

Certus

sfi Centre for Research-based Innovation

# Agenda

I. Software Testing

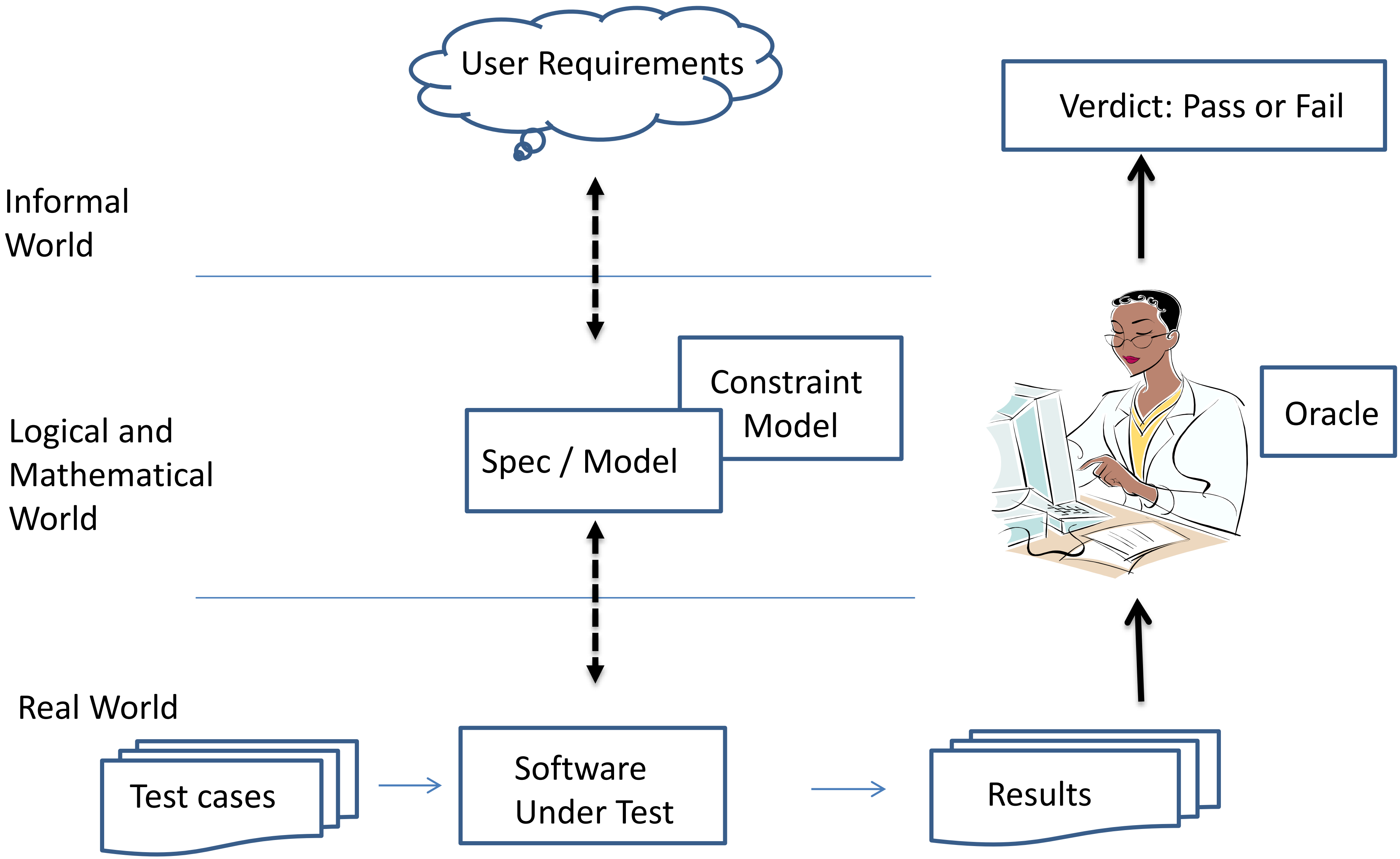
II. Optimal Test Suite Reduction

III. Multi-objectives Test Suite Reduction

IV. Industrial Application

V. Conclusions and Perspectives

# Software Testing



# Software Testing

Software test preparation is a **cognitively complex task**:

- Requires to understand both model and code to create interesting test cases ;
- Program's input space is usually very large (sometimes unbounded) ;
- Complex software (e.g., implementing ODEs or PDEs) yields to complex bugs ;
- Test oracles are hard to define (non-testable programs) ;

**Not easily amenable to automation**:

- Automatic test data generation is undecideable in the general case!
- Exploring the input space yields to combinatorial explosion ;
- Fully automated oracles are usually not available ;

# How software testing differs from other program verification techniques?

☞ **Static analysis** finds simple faults (division-by-zero, overflows, …) at compile-time, while **software testing** finds functional faults at run-time (P returns 3 while 2 was expected)

☞ **Program proving** aims at formally proving mathematical invariants, while **software testing** evaluates the program in its execution environment

☞ **Model-checking** explores paths of a model of the software under test for checking temporal properties or finding counter-examples, while **software testing** is based on program executions

# Some Hot Research Topics in Software Testing

- Automatic test case generation

  Find test cases to exercise specific behaviors, to execute specific code locations, to cover some test objectives (e.g., all-statements, all-k-paths)

- Test suite reduction, test suite prioritization, test execution scheduling

- Robustness and performence testing

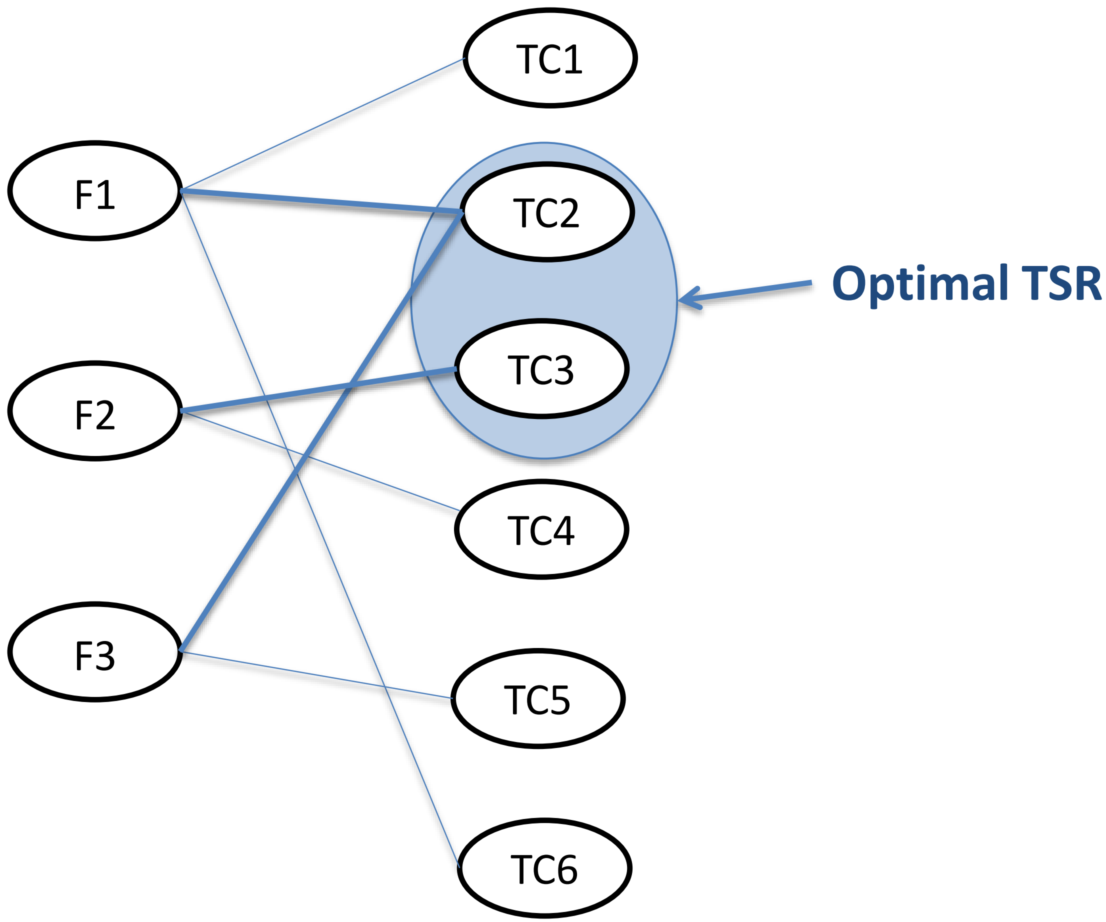
- Testing complex code (e.g., floating-point and iterative computations)

**Our thesis:** Global constraints can efficiently tackle these problems!

(*High-level primitives with specialised filtering algorithms*)

# Optimal Test Suite Reduction

# Optimal TSR: the core problem




**Optimal TSR**: find a minimal subset of TC such that each F is covered at least once (Practical importance but NP-hard problem!) – An instance of *Minimum Set Cover*

# The nvalue global constraint

$$nvalue(n, v)$$

Where:

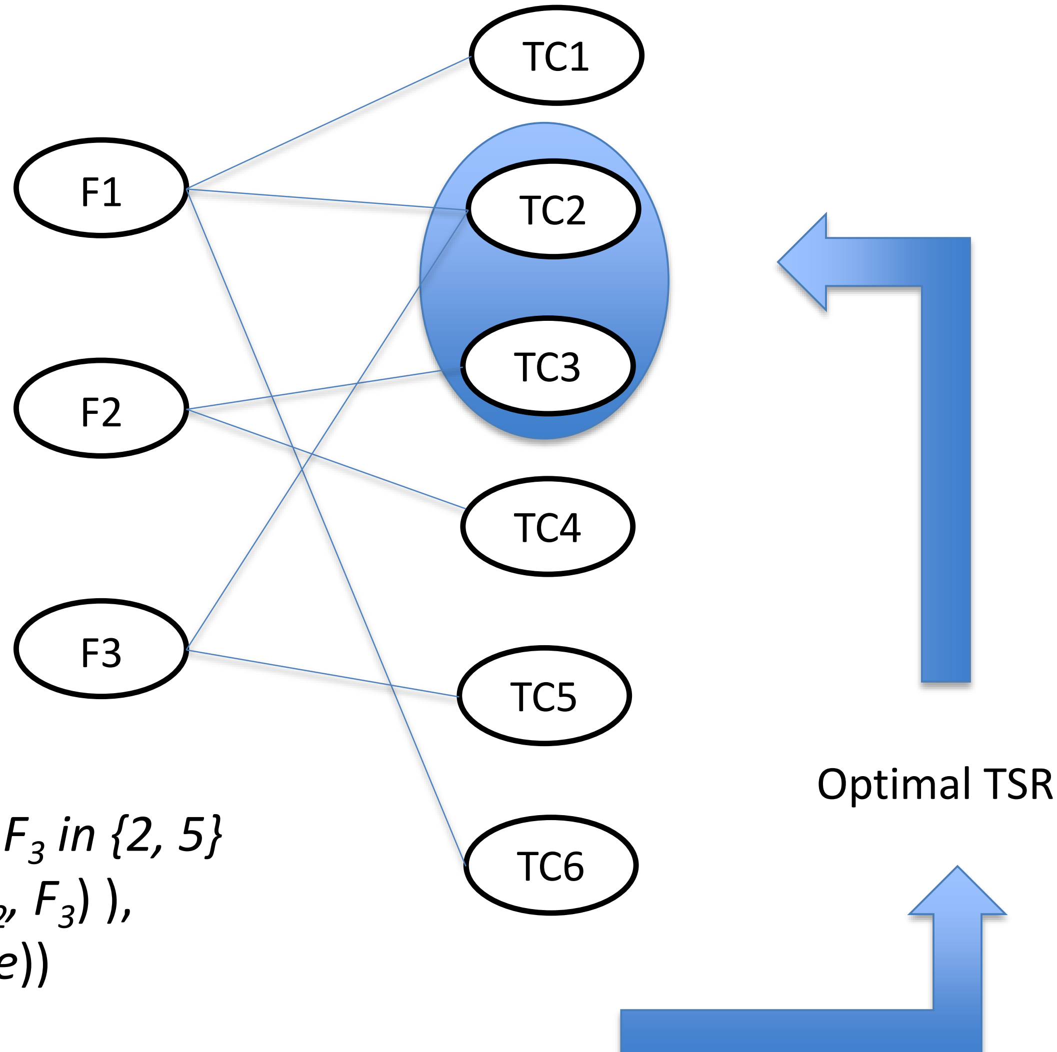
$n$ is an FD_variable

$v = (v_1, \dots, v_k)$ is a vector of FD_variables

$nvalue(n, v)$ holds iff $n = card(\{v_i\}_{i \text{ in } 1..k})$

Introduced in [Pachet and Roy’99], first filtering algorithm in [Beldiceanu’01]
Solution existence for nvalue is NP-hard [Bessiere et al. ‘04]

# Optimal TSR: CP model with nvalue (1)




*$F_1$ in {1, 2, 6}, $F_2$ in {3, 4}, $F_3$ in {2, 5}*
nvalue( *MaxNvalue*, *($F_1$, $F_2$, $F_3$)* ),
label(minimize(*MaxNvalue*))

/* branch-and-bound search among feasible solutions */

# The global_cardinality constraint

$$gcc(\ t, d, v)$$

Where

$t = (t_1, \ldots, \ t_N)$ is a vector of N variables, each $t_j$ in $Min_j .. Max_j$

$d = (d_1, \ldots., d_k)$ is a vector of k values

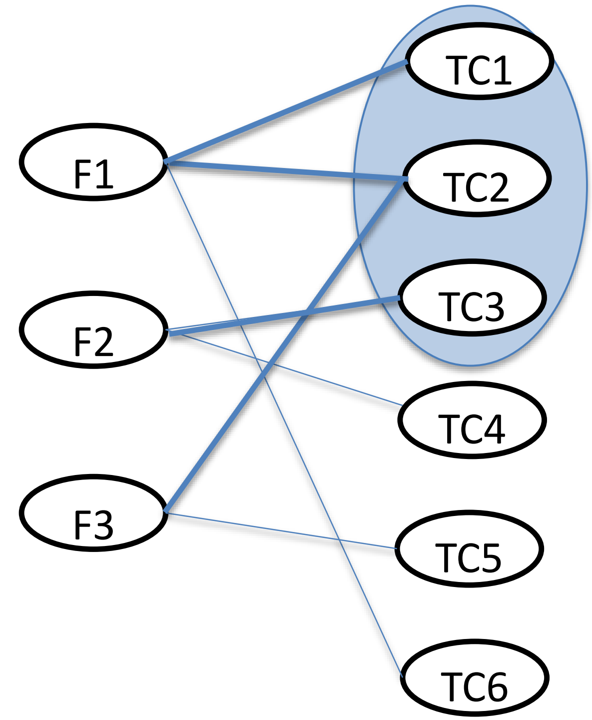
$v = (v_1, \ldots, \ v_k)$ is a vector of k variables, each $v_i$ in $Min_i..Max_i$

$gcc(t, d, v)$ holds iff

$$\forall i \ in\ 1..k,$$
$$v_i = card(\ \{t_j = di\}_{j\ in\ 1..N})$$

Filtering algorithms for $gcc$ are based on max flow computations in a network flow [Regin AAAI'96]

# 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 a solution of TSR
$TC_1$ covers exactly $V_1$ requirements in $(F_1, F_2, F_3)$
$TC_2$ " $V_2$ "
$TC_3$ " $V_3$ "
...

Where $F_1$, $F_2$, $F_3$, $V_1$, $V_2$, $V_3$, ... denote finite-domain variables

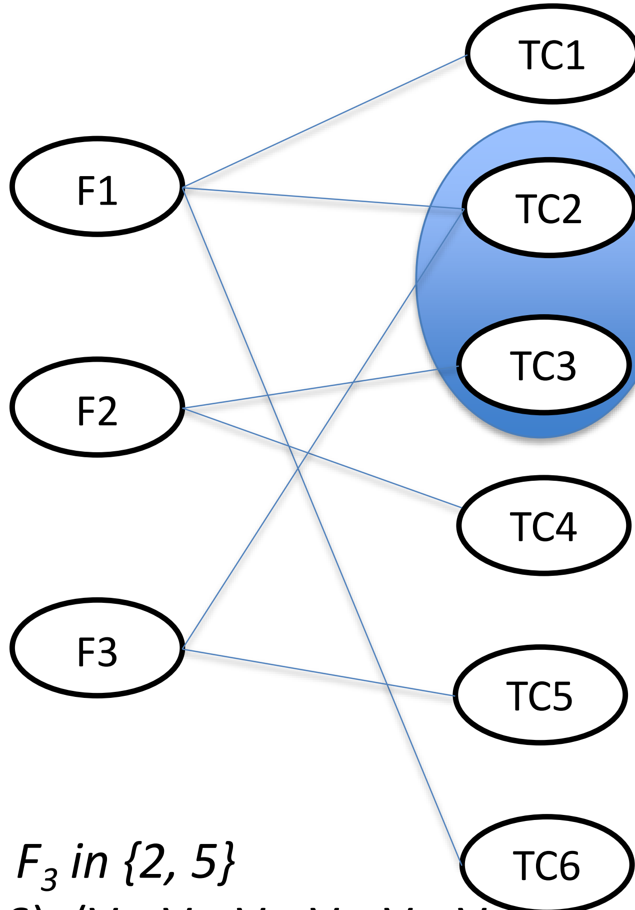
$F_1$ in {1, 2, 6}, $F_2$ in {3, 4}, $F_3$ in {2, 5}
$V_1$ in {0, 1}, $V_2$ in {0, 2}, $V_3$ in {0, 1}, $V_4$ in {0, 1}, $V_5$ in {0, 1}, $V_6$ in {0, 1}

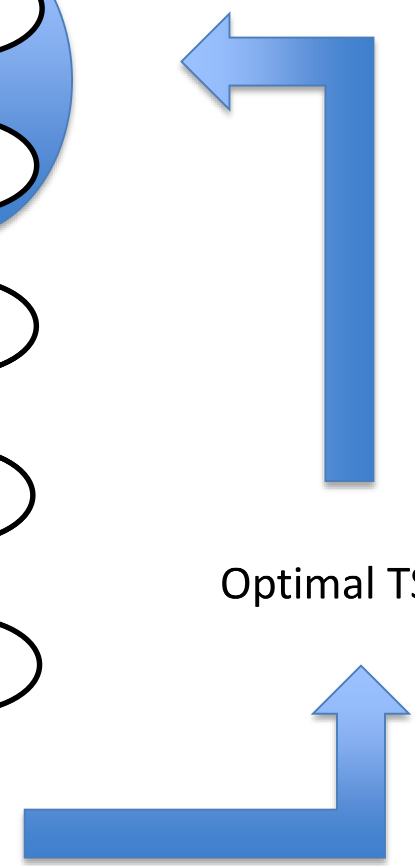
Here, for example, **$V_1 = 1$, $V_2 = 2$, $V_3 = 1$, $V_4 = 0$, $V_5 = 0$, $V_6 = 0$** is a feasible solution

But, not an optimal one!

# Optimal TSR: CP model with two gcc (2)

[Gotlieb et al., 2014]



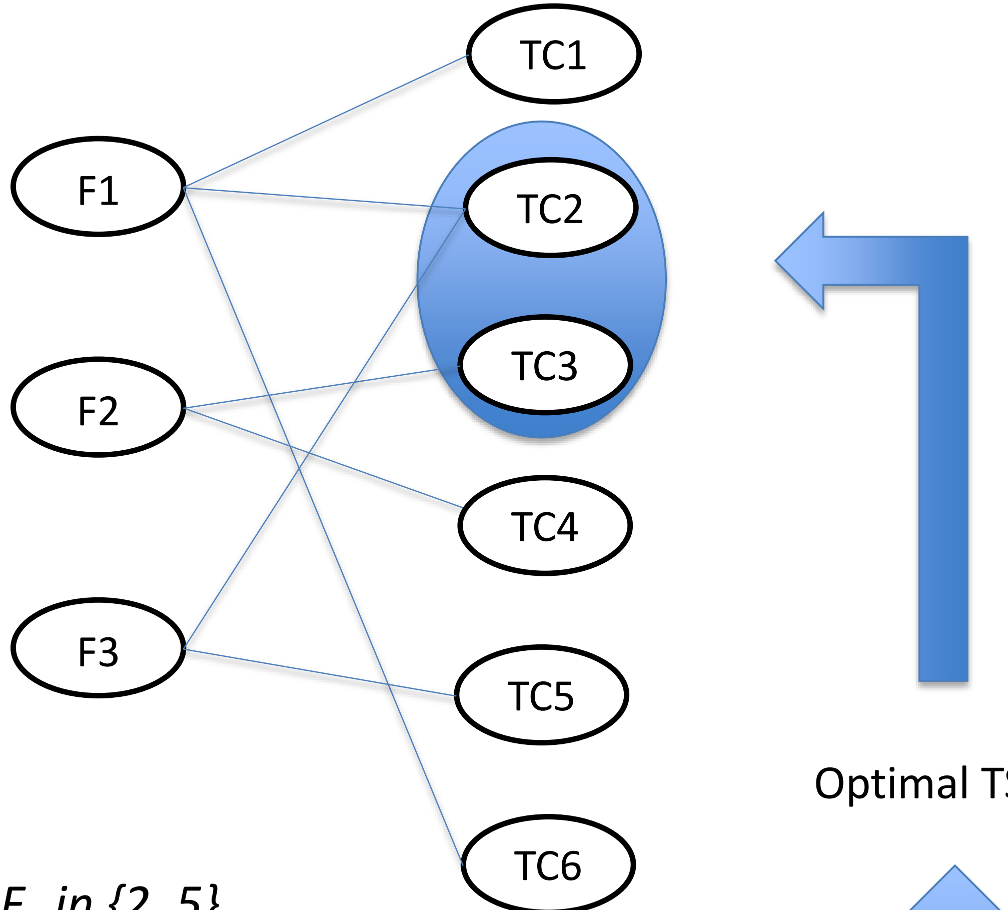




$F_1$ *in {1, 2, 6}*, $F_2$ *in {3, 4}*, $F_3$ *in {2, 5}*
gcc( ($F_1$, $F_2$, $F_3$), (1,2,3,4,5,6), ($V_1$, $V_2$, $V_3$, $V_4$, $V_5$, $V_6$) ),
gcc(($V_1$, $V_2$, $V_3$, $V_4$, $V_5$, $V_6$), (0-_), (*MaxOReq*-_ )),
label(maximize(*MaxOReq*))

/* search heuristics by enumerating the Vi first */

# 3. Optimal TSR: CP model Mixt (3)

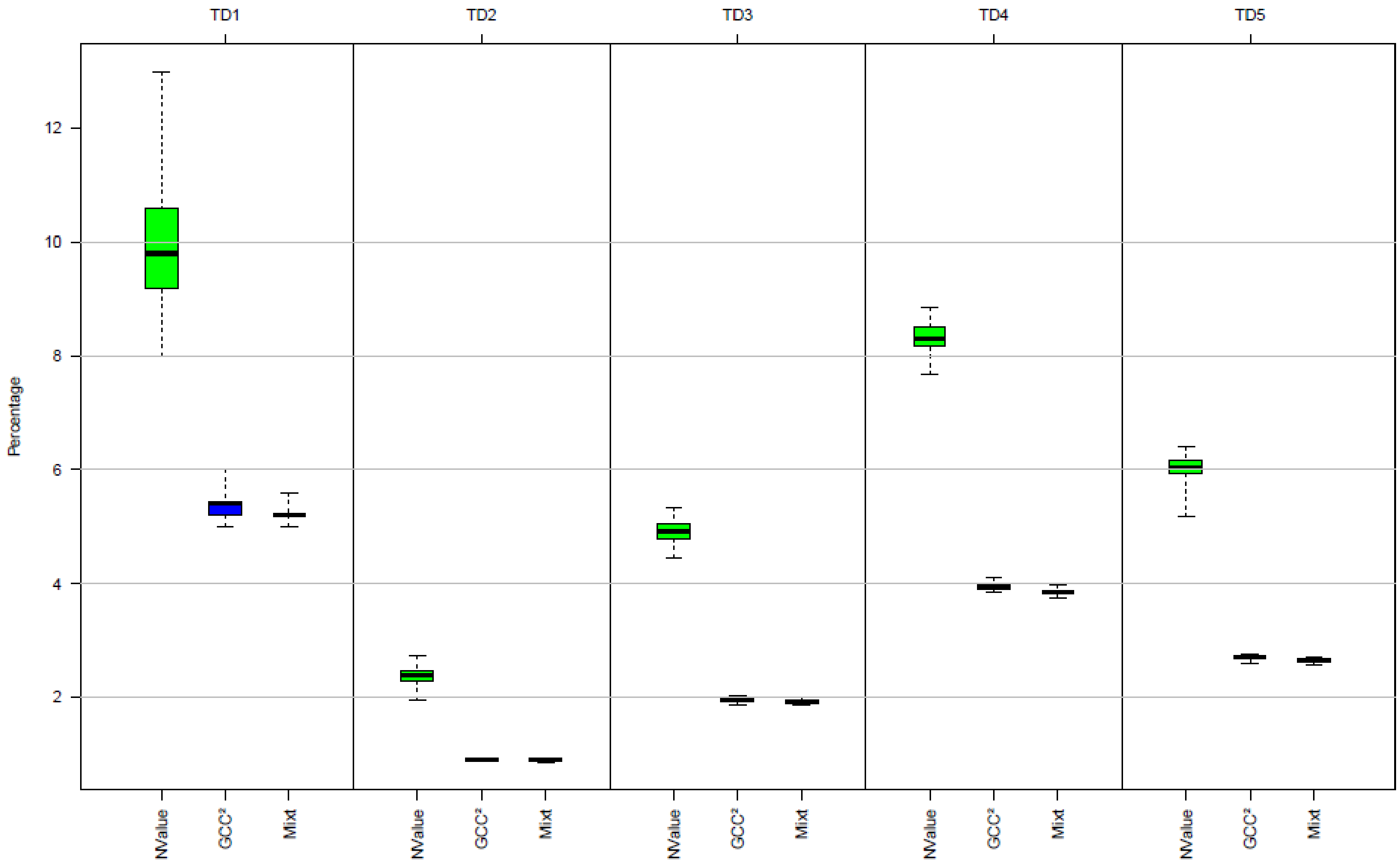
[joint work with A. Pétillon and M. Carlsson]




*$F_1$ 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, ($F_1$, $F_2$, $F_3$),
label(minimize(MaxNvalue))

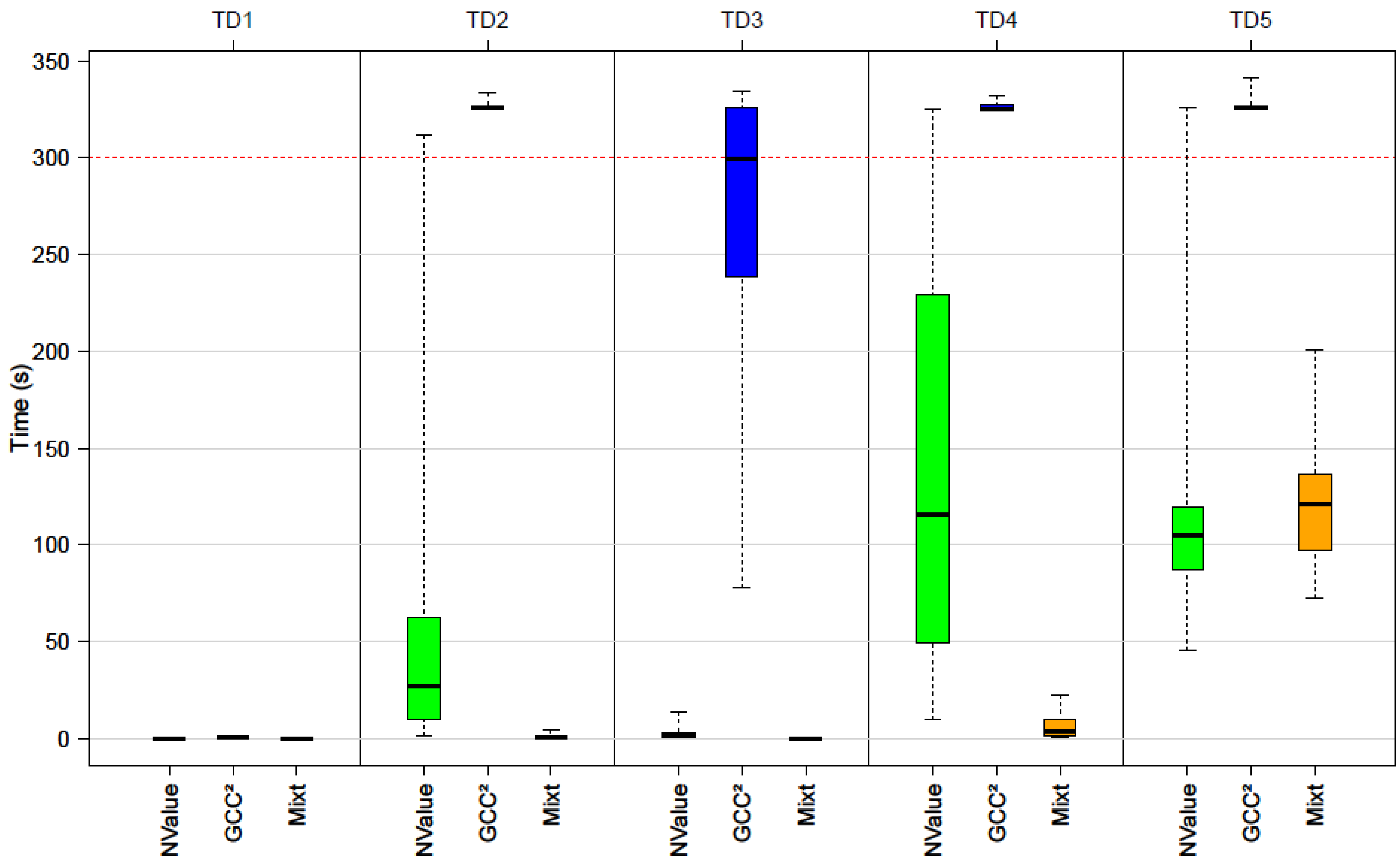
/* + presolve + labelling heuristics based on max */

# Model comparison on random instances (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 |

# Model comparison on random instances (CPU time to find a global optimum)




| | TD1 | TD2 | TD3 | TD4 | TD5 |
|---|---|---|---|---|---|
| Requirements | 20 | 90 | 60 | 60 | 30 |
| Test cases | 70 | 100 | 100 | 200 | 500 |
| Density | 8 | 20 | 20 | 20 | 8 |

# Optimal TSR: existing approaches

- Exact method: ILP formulation [Hsu Orso ICSE 2009] – MINTS/CPLEX, MINTS/MiniSAT

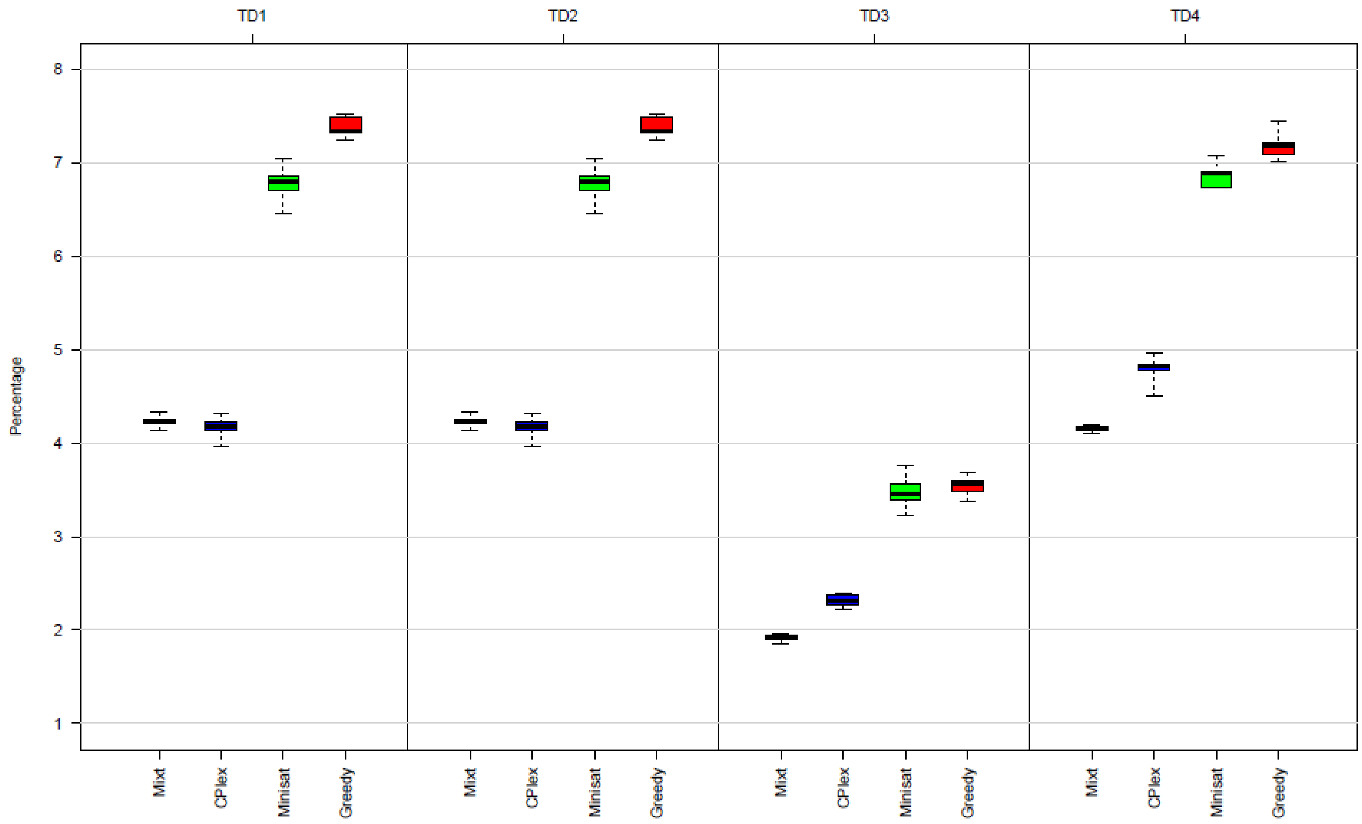
Minimize $\sum_{i=1..6} xi$ (minimize the number of test cases)

subject to $\left\{\begin{matrix} x1 + x2 + x6 \geq 1 \\ x3 + x4 \geq 1 \\ x2 + x5 \geq 1 \end{matrix}\right\}$ (cover every req. at least once)

- Approximation algorithms (greedy) –

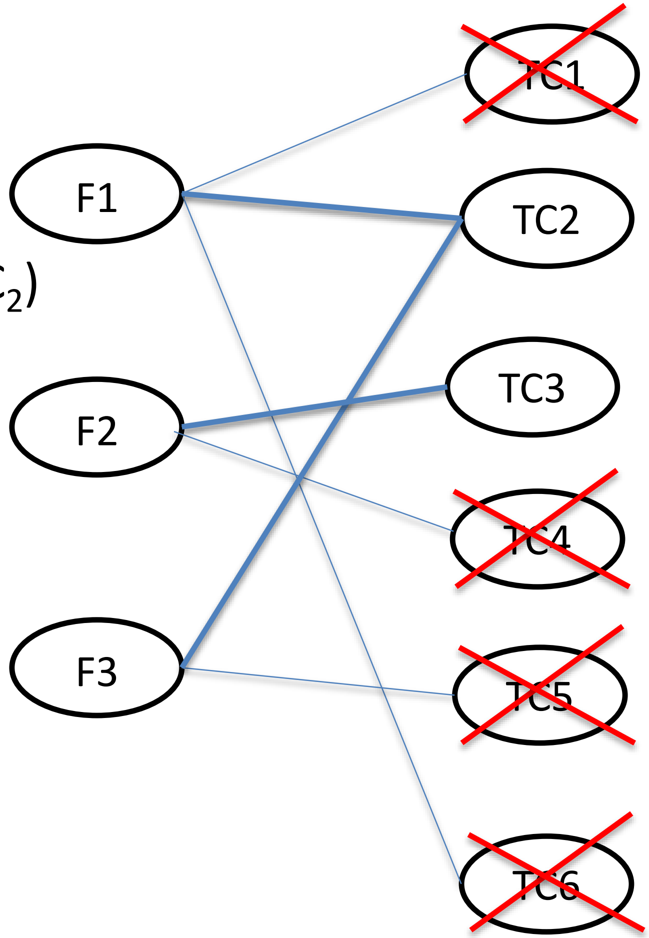
```
R = Set of reqs, Current = Ø
while( Current ≠ R)
    Select a test case that covers the most uncovered reqs ;
    Add covered reqs to Current ;
return Current
```

# Comparison with other approaches (Reduced Test Suite percentage in 60 sec)




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

# Introducing model presolve



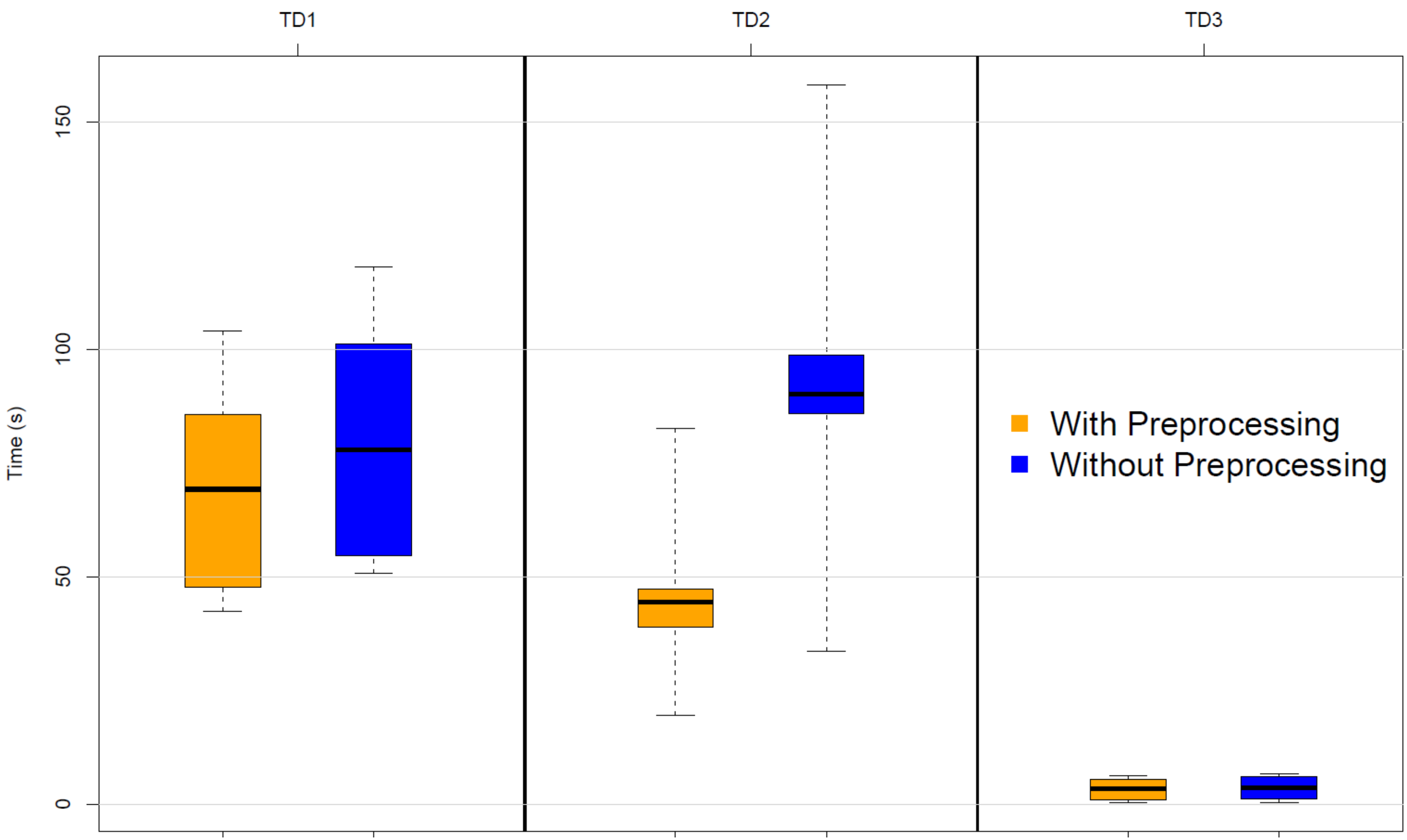

$F_1$ in {1, 2, 6} → $F_1 = 2$ as $cov(TC_1) = cov(TC_6) \subset cov(TC_2)$
withdraw $TC_1$ and $TC_6$

$F_3$ is covered → withdraw $TC_5$

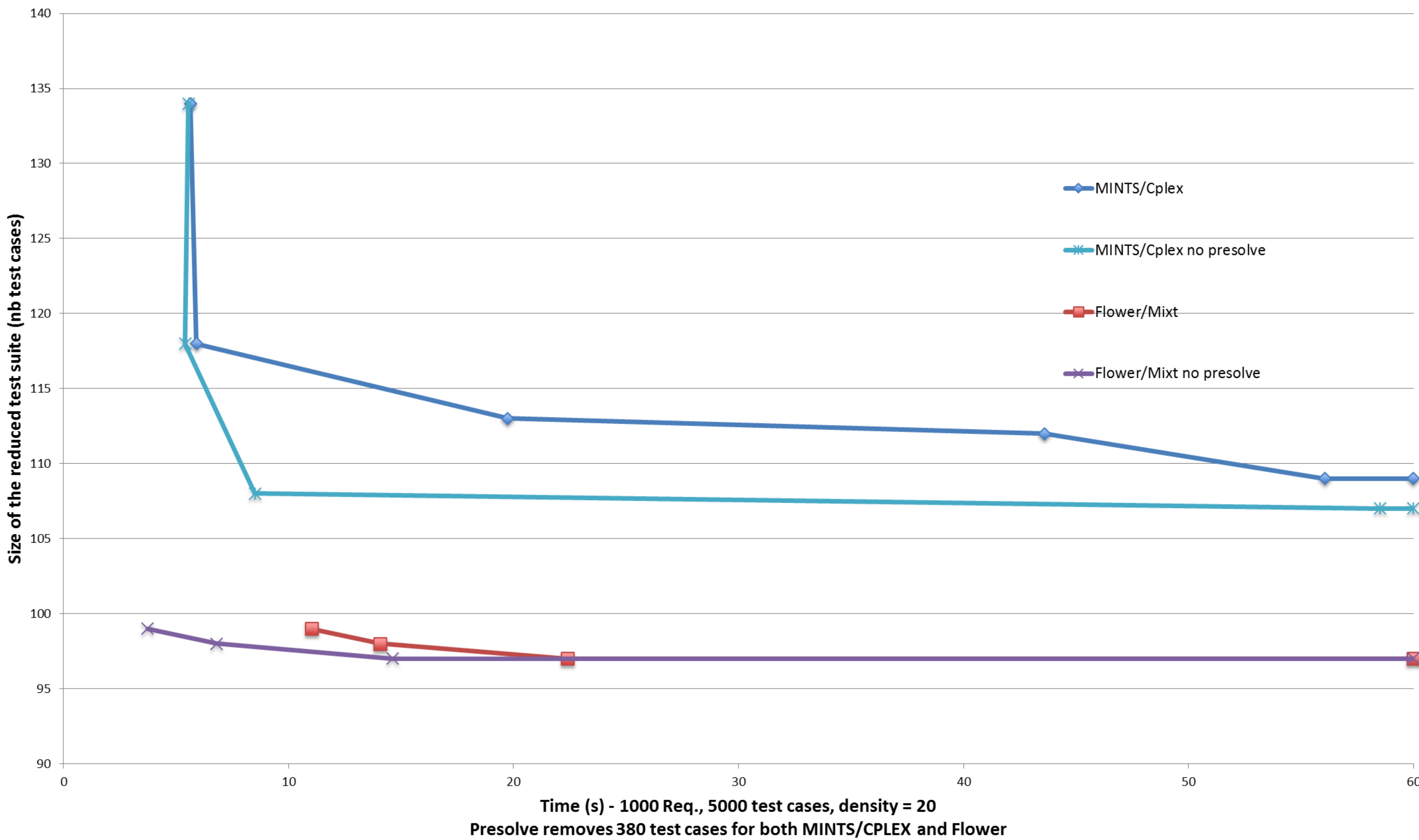
$F_2$ in {3,4} → e.g., $F_2 = 3$, withdraw $TC_4$

We proposed an iterative algorithm to apply these preprocessing rules to simplify the problem

# Presolve: Experimental results (1)



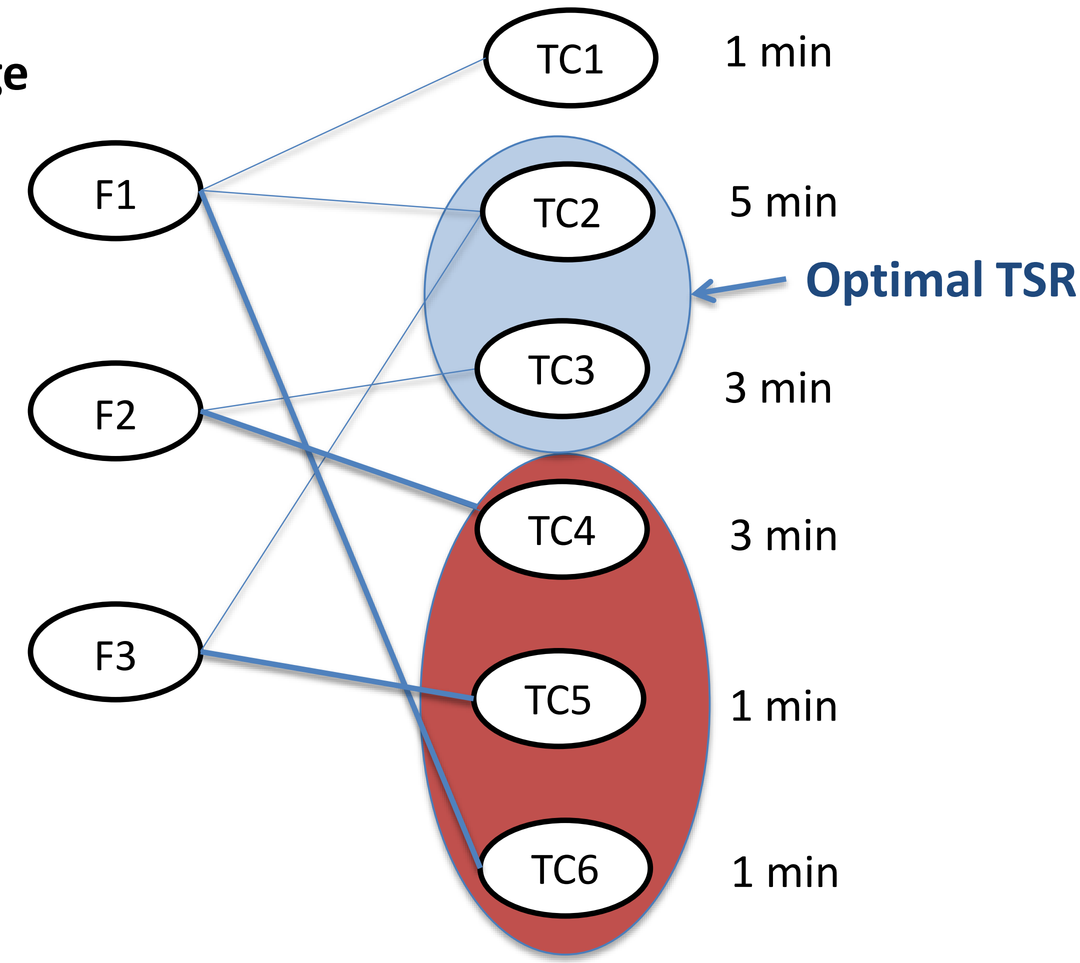
# Presolve: Experimental results (2)



# Multi-objectives Test Suite Reduction

# Optimal TSR: the core problem

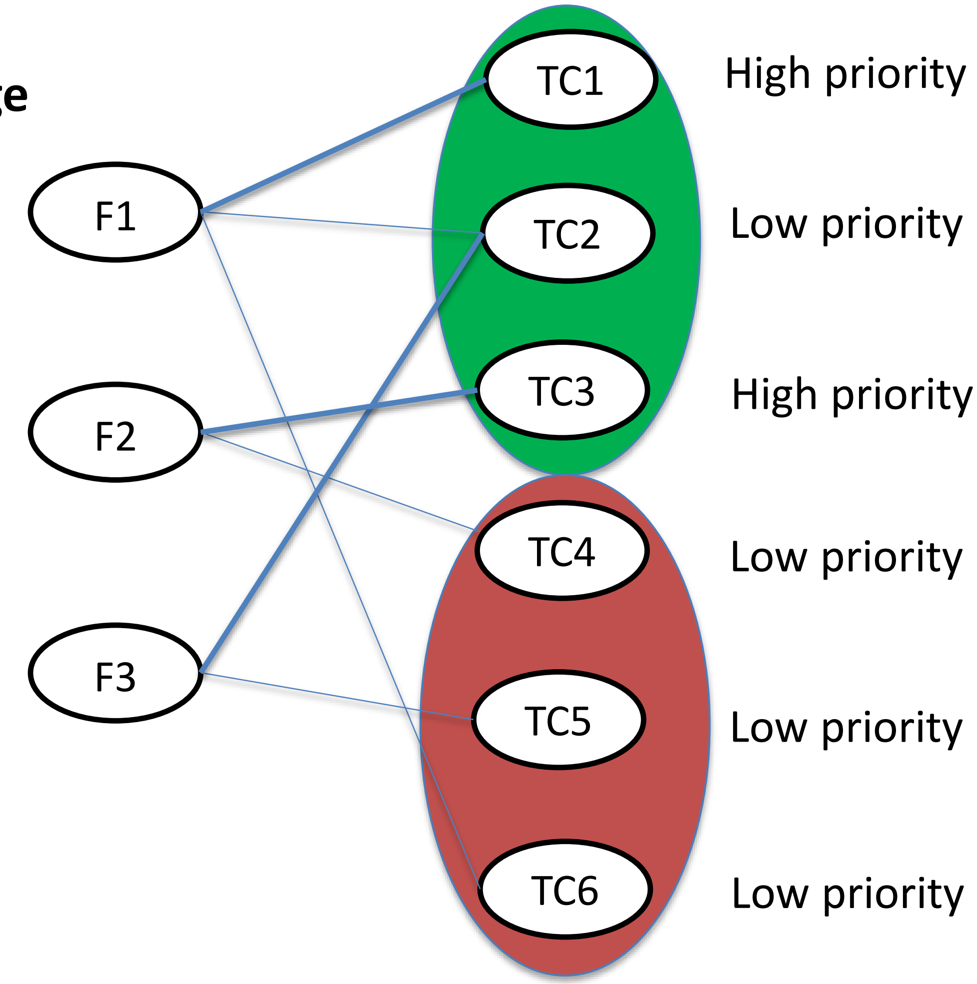
**Requirements coverage** is always a prerequiste but other criteria than the size of the test suite are also sought:




Execution time!

# Optimal TSR: the core problem

**Requirements coverage** is always a prerequiste but other criteria than the size of the test suite are also sought:




Fault revealing capabilities!

# Proposed approaches

1. Actual multi-objectives optimization with search-based algorithms (Pareto Front) [Wang et al., 2013, 2014]

   Aggregated cost function using RW-algo, URW-algo, and many others
   Based on computed values

   **No constraint model!**

2. Cost-based single-objective constrained optimization

   Based on a CP model with global constraints

   **Constrained optimization model!**

# Flower/C: An extension of Flower with costs

$R_1,..,R_n$: Requirements
$t_1,..,t_m$: Test cases - Each test case $t_i$ is associated a unitary cost $c_i$
$O_1,..,O_m$: Occurrences variables

Minimize TotalCost

s.t

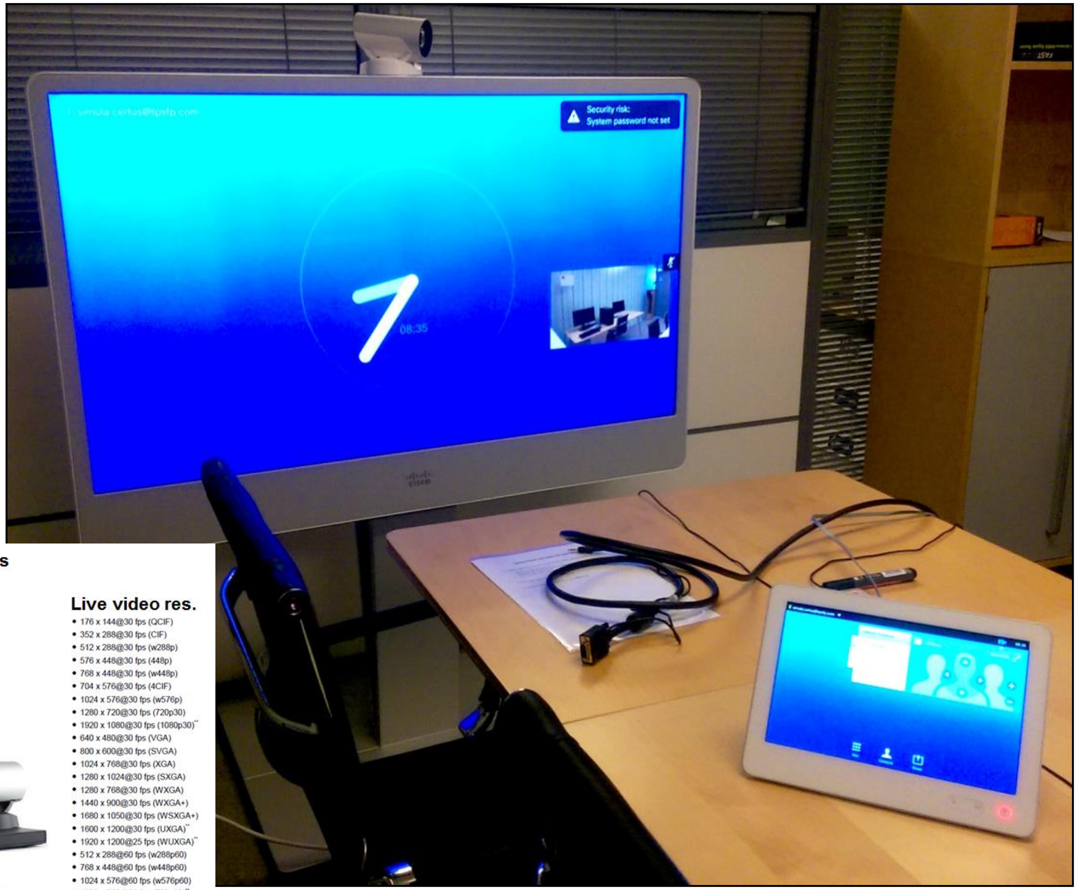
gcc(($R_1$, ..., $R_n$), ($t_1$, ..., $t_m$), ($O_1$, ..., $O_m$))
for i=1 to m do $B_i = (O_i > 0)$
scalar_product(($B_1$, ..., $B_m$), ($c_1$, ..., $c_m$), TotalCost)

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

On-going experimental evaluation!

# Industrial Application

# The CISCO's Video Conferencing Systems Product Line



### Multisite features
- 4-way 1080p30 High Definition SIP/H.323 MultiSite
- Full individual audio and video transcoding
- Individual layouts in MultiSite CP (takes out self view)
- H.323/SIP/VoIP in the same conference
- Support for Presentation (H.239/BFCP) from any parti
- Best Impression (Automatic CP Layouts)
- H.264, Encryption, Dual Stream from any site
- IP Downspeeding
- Dial in/Dial out
- Additional telephone call (no license required)
- Conference rates up to 10 Mbps

### Audio features
- CD-Quality 20KHz Mono and Stereo
- Eight separate acoustic echo cancellers
- 8-port Audio mixer
- Automatic Gain Control (AGC)
- Automatic Noise Reduction
- Active lip synchronization

### Audio standards
- G.711, G.722, G.722.1, 64 kbps & 128 kbps MPEG4 AAC-LD, AAC-LD Stereo

### IP network features
- DNS lookup for service configuration
- Differentiated Services (QoS)
- IP adaptive bandwidth management (in
- Auto gatekeeper discovery
- Dynamic playout and lip-sync buffering
- H.245 DTMF tones in H.323
- Date and Time support via NTP
- Packet Loss based Downspeeding
- URI Dialing
- TCP/IP
- DHCP
- 802.1x Network authentication
- ClearPath

### Live video res.
- 176 x 144@30 fps (QCIF)
- 352 x 288@30 fps (CIF)
- 512 x 288@30 fps (w288p)
- 576 x 448@30 fps (448p)
- 768 x 448@30 fps (w448p)
- 704 x 576@30 fps (4CIF)
- 1024 x 576@30 fps (w576p)
- 1280 x 720@30 fps (720p30)
- 1920 x 1080@30 fps (1080p30)**
- 640 x 480@30 fps (VGA)
- 800 x 600@30 fps (SVGA)
- 1024 x 768@30 fps (XGA)
- 1280 x 1024@30 fps (SXGA)
- 1280 x 768@30 fps (WXGA)
- 1440 x 900@30 fps (WXGA+)
- 1680 x 1050@30 fps (WSXGA+)
- 1600 x 1200@30 fps (UXGA)**
- 1920 x 1200@25 fps (WUXGA)**
- 512 x 288@60 fps (w288p60)
- 768 x 448@60 fps (w448p60)
- 1024 x 576@60 fps (w576p60)
- 1280 x 720@60 fps (720p60)**
- 720p30 from 768kbps
- 720p60 from 1152kbps
- 1080p30 from 1472kpbs

### Video features
- Native 16:9 Widescreen
- Advanced Screen Layouts
- Intelligent Video Management
- Local Auto Layout
- 9 embedded individual video compositors

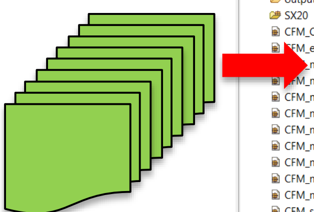
### Security features
- Management via HTTPS and SSH
- IP Administration Password
- Menu Administration Password
- Disable IP services
- Network Settings protection

### Bandwidth
- H.323/SIP up to 6 Mbps point-to-point
- Up to 10 Mbps total MultiSite bandwidth

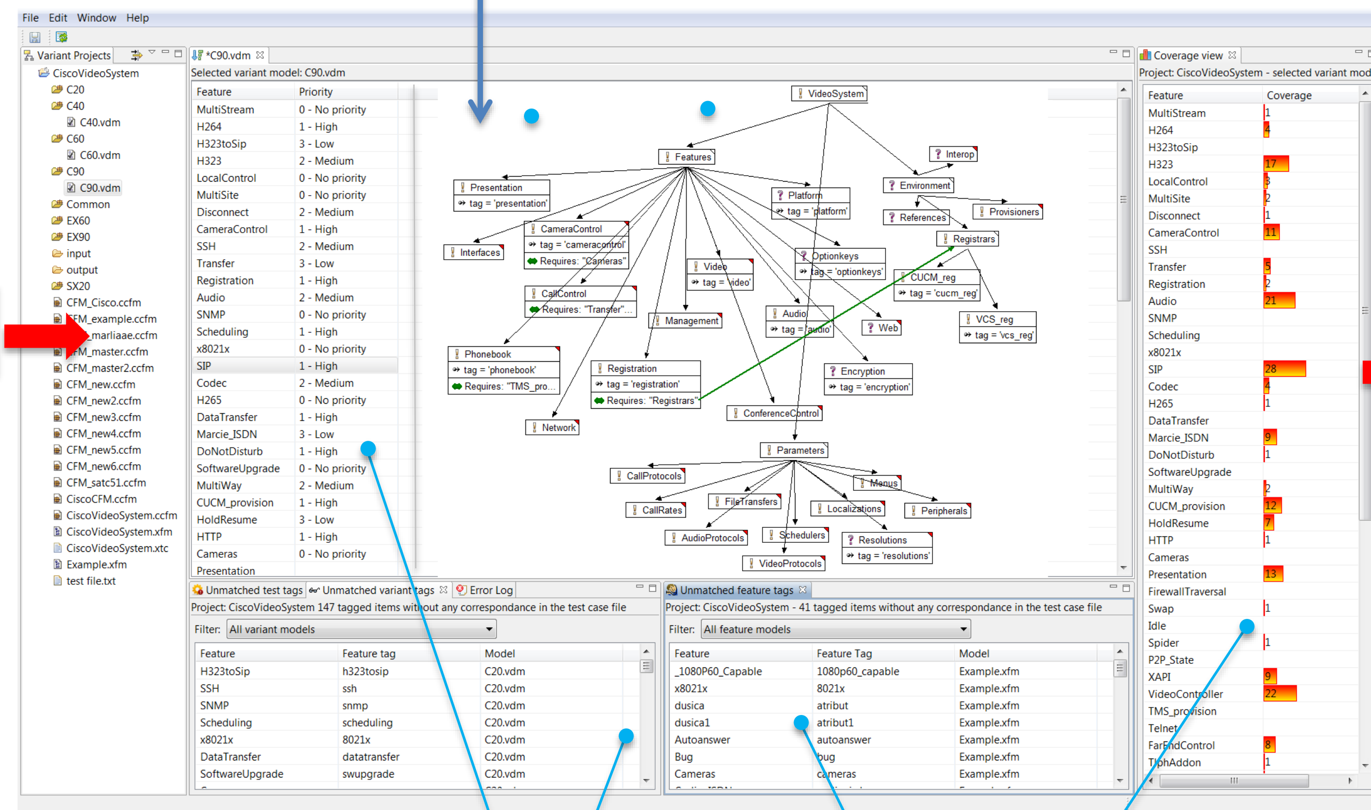
### Protocols
- H.323
- SIP

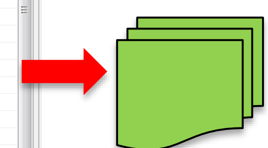
# TITAN


Variability model to describe a software product line



Unoptimized test suite



Optimized (reduced/ prioritized) test suite

Diagnostic views, feature coverage

# Conclusions

- Global constraints can efficiently and effectively tackle difficult software testing problems – experimental results and initial industrial case studies
- So far, only a few subset of existing global constraints have been explored for that purpose (e.g., nvalue, gcc, element, all_different,…)
- Some software testing problems require the creation of dedicated global constraints to facilitate disjunctive reasoning, case-based reasoning or probabilistic reasoning

→ there is room for *Research & Innovation (H2020)* in that area!

# Perspectives

- More industrial case studies for demonstrating the potential of global constraints for software testing applications
  - ABB Robotics [Mossige et al., 2014, 2015]
  - THALES

TITAN in the commercial preparation phase

- Test Case Execution Scheduling with **CUMULATIVE**

# References
## (cited in the slides)


[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.

[Wang et al., 2014] S. Wang, S. Ali, and A. Gotlieb.
*Cost-effective test suite minimization in product lines using search techniques*.
Journal of Systems and Software, 2014. In Press - avail. on line 27 Aug. 2014.

[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.

[Mossige et al., 2014] Morten Mossige, Arnaud Gotlieb, and Hein Meling.
Using CP in automatic test generation for ABB robotics' paint control system.
In *Principles and Practice of Constraint Programming (CP'14) –*
*Application track, Awarded best application paper*, Lyon, France, Sep. 2014.

[Wang et al., 2013] S. Wang, S. Ali, and A. Gotlieb.
Minimizing test suites in software product lines using weight-based genetic algorithms.
In *Genetic and Evolutionary Computation Conference (GECCO'13)*, Amsterdam, Jul. 2013.