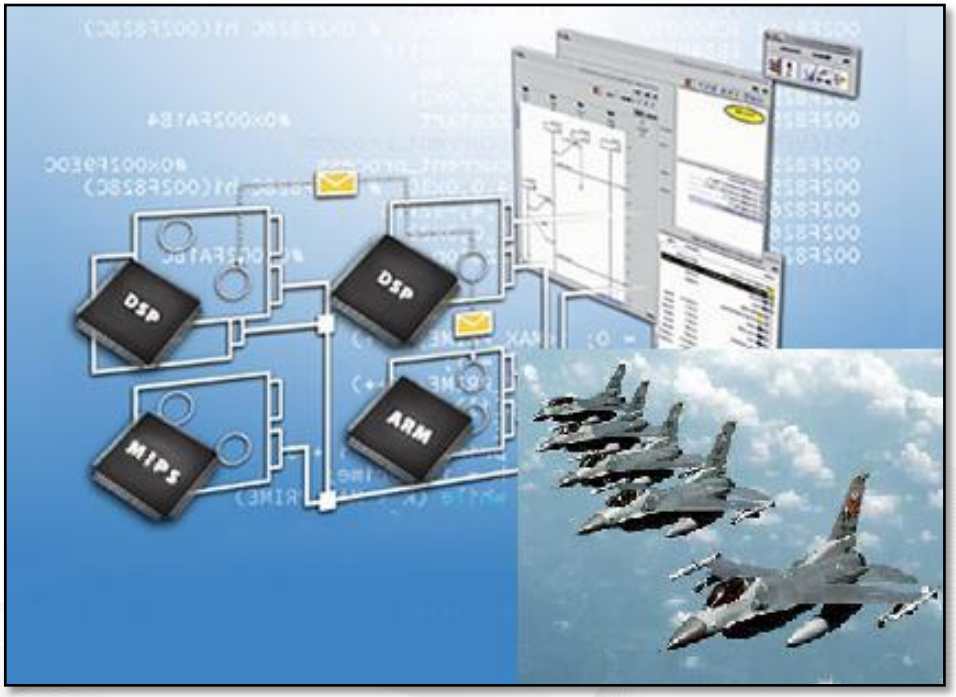
# Testing Deadline Misses for Real-Time Systems Using Constraint Optimization Techniques

Stefano Di Alesio[1,2]
Arnaud Gotlieb[1]
Shiva Nejati[1]
Lionel Briand[1,2]

[1]Simula Research Laboratory
[2]University of Luxembourg

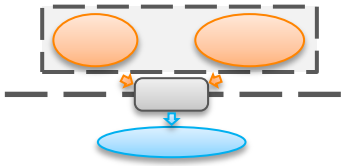



emcelettronica.com

21/04/2012

# We present a technique to use Constraint Optimization to test deadline misses for RTES

## Performance Requirements (PRs) vs. Real Time Embedded Systems (RTES)



## Using Constraint Programming for Verification and Validation of RTES

| | $j_0$ | $j_1$ | $j_2$ |
|---|---|---|---|
| $exec(j)$ | 2 | 2 | 2 |
| $p(j)$ | 100 | 101 | 102 |
| $dl(j)$ | 3 | 2 | 3 |
| $max\_ia(j$ | 3 | 2 | 3 |
| $min\_dr(j$<br>$max\_dr(j$ | 3 | 2 | 3 |

## Evaluation, Experience and Current Work

# RTES are typically safety-critical, and thus bound to meet strict Performance Requirements

control-link.net

# Performance Requirements are the most difficult requirements to verify

**They depend on the environment the software interacts with (hw devices)**


**They depend on the computing platform on which the software runs**



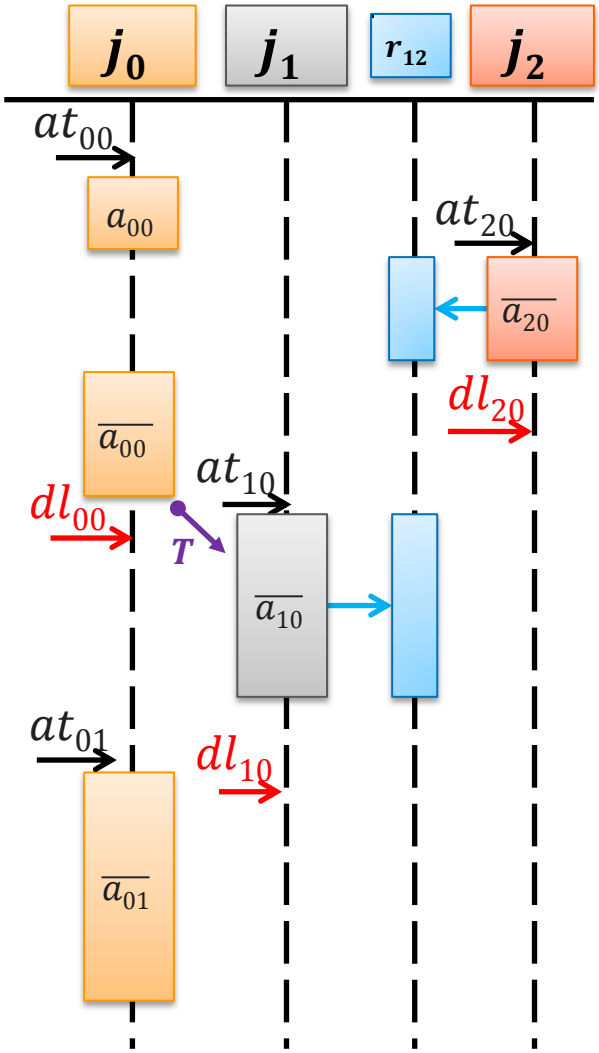
**They constraint the entire system's behavior and thus can't be checked locally**

# RTES have concurrent interdependent tasks which have to finish before their deadlines

Each task has a deadline (i.e., latest finishing time) w.r.t. its arrival time

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

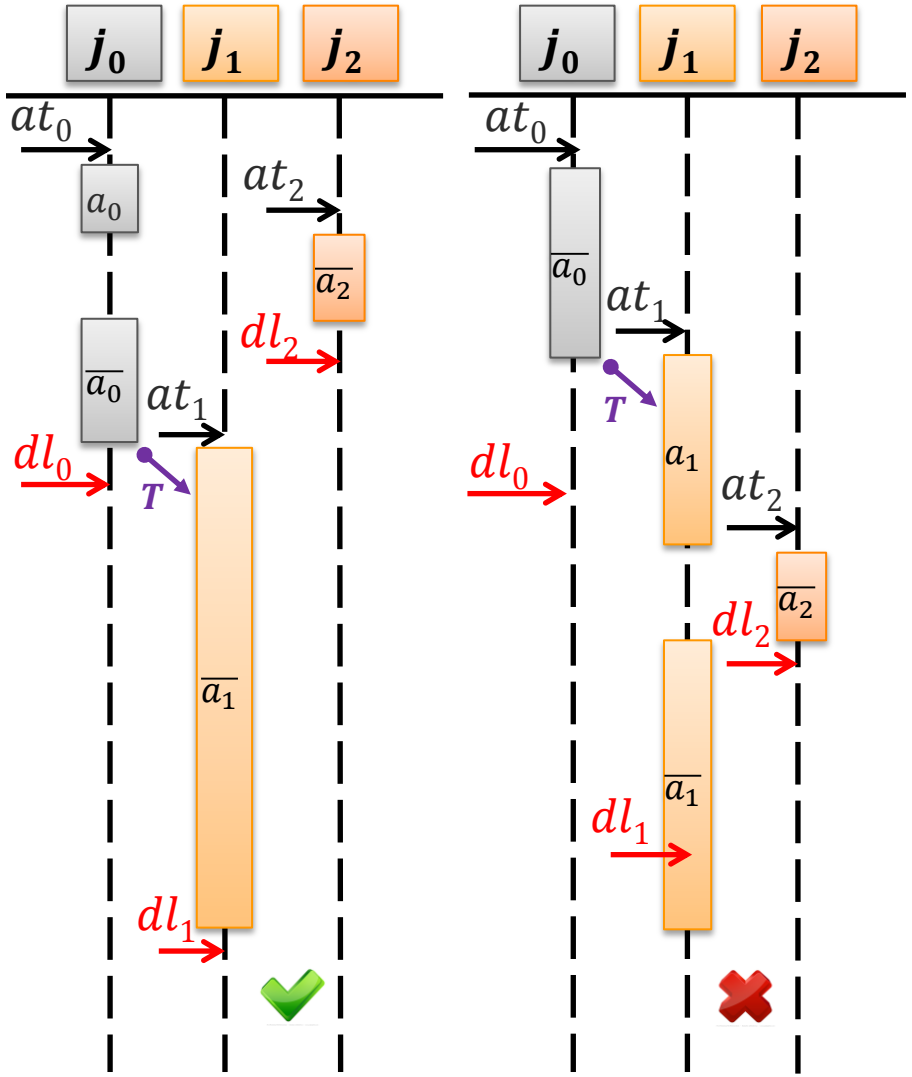
Each task can trigger other tasks, and can share computational resources with other tasks



# Particular sequences of arrival times of tasks determine deadline miss scenarios

$j_0, j_1, j_2$ 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!



# We are looking for sequences of arrival times maximizing the likelihood of deadline misses

**Arrival times for tasks in a RTES depend on the environment**

**Arrival times can be tuned during software testing**

$a_1 = 1$
$a_2 = 3$
$a_3 = 3$
$a_4 = 7$

$a_1 = 1$
$a_2 = 3$
$a_3 = 4$
$a_4 = 7$

**Real Time Embedded System**

**Real Time Embedded System**

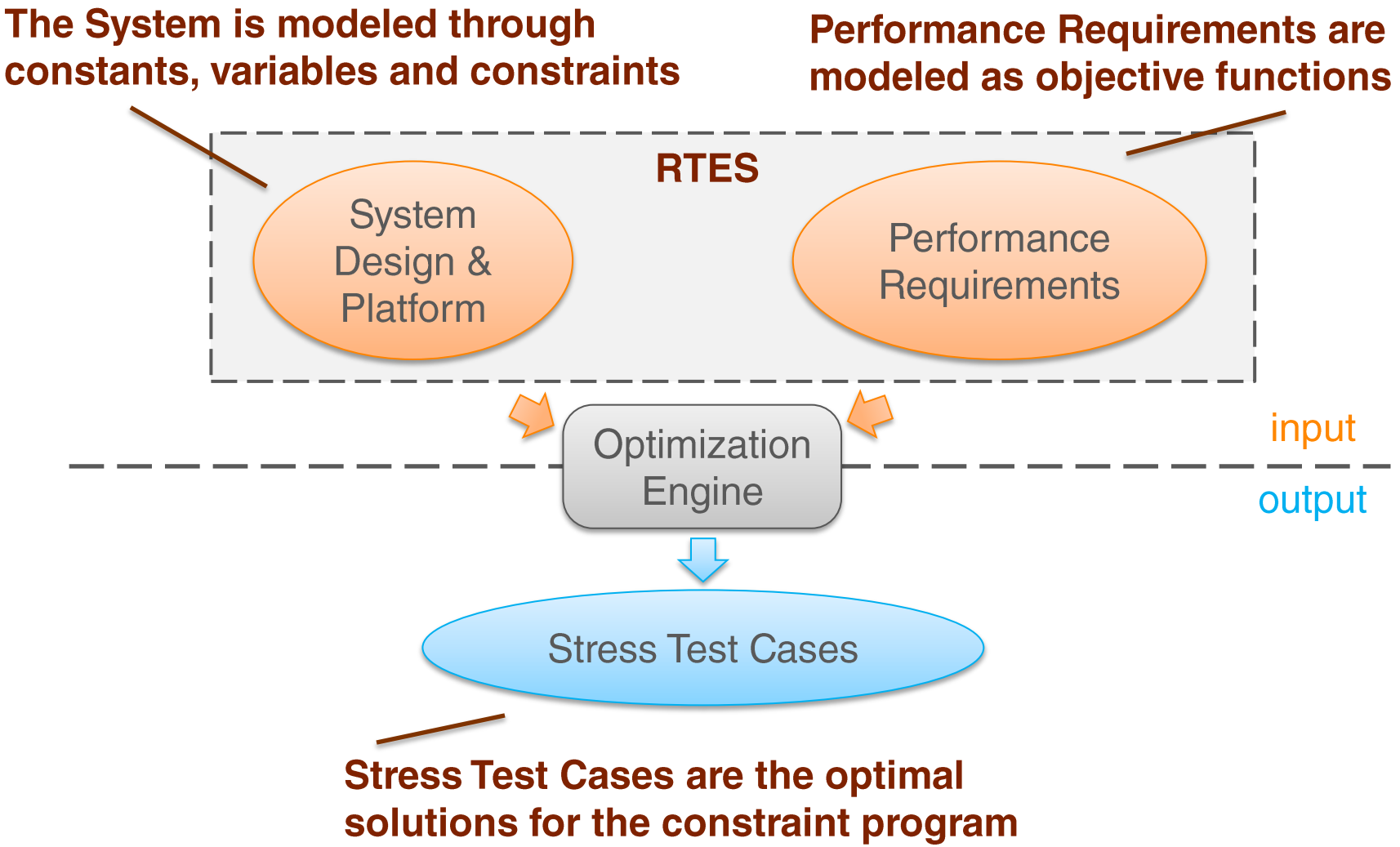
**A sequence of arrival times identified by our approach as likely to lead to a deadline miss defines a Stress Test Case**

# This problem has been well studied, but each existing approach has its weaknesses

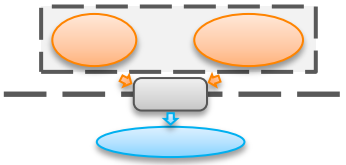
| | Schedulability Theory | Model Checking | Genetic Algorithms |
|---|---|---|---|
| **Basis** | Mathematical Theory | System Modeling | System Modeling |
| **Background** | WCET, Queuing Theory, etc. | Fixed-point Computation | Meta-Heuristic Search |
| **Key Features** | Theorems [1] | Graph-based / Symbolic [2] | Non-Complete Search [3] |
| **Weaknesses** | Assumptions, Multi-Core | Complex Modeling | Non-Exhaustive Search |


[1] J. W. S. Liu, "Real-Time Systems". Prentice Hall, 2000
[2] M. Mikucionis, K. Larsen, B. Nielsen, J. Illum, A. Skou, S.Palm, J.Pedersen, and P. Hougaaard, "Schedulability analysis using UPPAAL: Herschel-Planck case study", in ISoLA, 2010
[3] L. Briand, Y. Labiche, and M. Shousha, "Using genetic algorithms for early schedulability analysis and stress testing in real-time systems", Genetic Programming and Evolvable Machines, vol. 7 no. 2, pp. 145-170, 2006

# We model the RTES Design, Platform and PRs through a Integer Linear Program solved by CPLEX



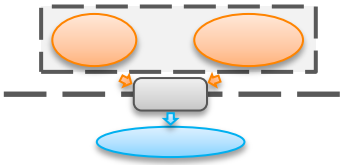
# Our approach tries to mitigate some weaknesses found in related works



| | Schedulability Theory | Model Checking | Genetic Algorithms | Our Approach |
|---|---|---|---|---|
| **Basis** | Mathematical Theory | System Modeling | System Modeling | System Modeling |
| **Background** | WCET, Queuing Theory, etc. | Fixed-point Computation | Meta-Heuristic Search | Mathematical Optimization |
| **Key Features** | Theorems [1] | Graph-based / Symbolic [2] | Non-Complete Search [3] | Complete Search |
| **Weaknesses** | Assumptions, Multi-Core | Complex Modeling | Non-Exhaustive Search | Performance / Scalability (?) |


[1] J. W. S. Liu, "Real-Time Systems". Prentice Hall, 2000
[2] M. Mikucionis, K. Larsen, B. Nielsen, J. Illum, A. Skou, S.Palm, J.Pedersen, and P. Hougaaard, "Schedulability analysis using UPPAAL: Herschel-Planck case study", in ISoLA, 2010
[3] L. Briand, Y. Labiche, and M. Shousha, "Using genetic algorithms for early schedulability analysis and stress testing in real-time systems", Genetic Programming and Evolvable Machines, vol. 7 no. 2, pp. 145-170, 2006

# Tasks and Platform design properties are modeled through constants

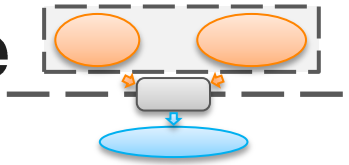


**Assumption 1: The scheduler checks tasks for preemptions at regular intervals ($tq$)**

**Assumption 2: The context switching time between tasks is negligible w.r.t. $tq$**

```
// T: Observation interval (range of
time quanta)
int tq = ...;
range T = 0..tq-1;

// c: Number of Processor Cores
int c = ...;

// n: Number of tasks
int n = ...;
range J = 0..n-1;

tuple TaskExecution {
int task;
int execution;
}

int priority[J] = ...;
int task_deadline[J] = ...;
int max_interarrival_time[J] = ...;
int min_duration[J] = ...;
int max_duration[J] = ...;
int triggers[J, J] = ...;
int dependent[J, J] = ...;
```

# Tasks and Platform real time properties are modeled through variables



$efe(a) \stackrel{\text{def}}{=}$ earliest time when $a$ could start if an unlimited number of cores was available

```
dvar int arrival_time[a in A] in T;
dvar int duration[a in A] in
     min_duration[a.task]..max_duration[a
     .task];
dvar int eligible_for_execution[a in A]
     in est[a]..lst[a];
dvar int start[a in A] in est[a]..lst[a];
dvar int end[a in A] in eet[a]..let[a];
dvar int task_execution_deadline[a in A]
     in T;
dvar int deadline_miss[a in A] in -
     tq..tq;
dvar int active[a in A, t in T] in 0..1;
```

$$active(a,t) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } a \text{ is executing at time } t \\ 0 & \text{otherwise [1]} \end{cases}$$

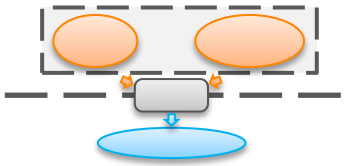
Time quanta →

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $a_0$ | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $a_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $a_2$ | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| $a_3$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

Task executions

[1] C.L. Pape and P. Baptiste, "Resource Constraints for preemptive job-shop Scheduling", Constraints, vol. 3, no. 4, pp. 263-287, 2098

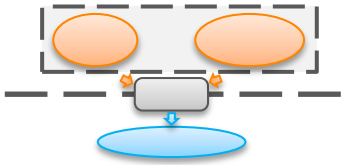
# The Platform scheduler behavior is modeled through 5 sets of constraints



**1. Well-Formedness: relations directly following from variables definitions**

```
/* I. Well-formedness constraints */
forall(a in A) {

     wf3: eligible_for_execution[a] <=
               start[a];
     wf4: start[a] <= end[a];

     if(prevc(A, a).task == a.task)
          wf6: eligible_for_execution[a] ==
                    maxl(arrival_time[a],
                         end[prevc(A, a)]);
     else
          wf7: eligible_for_execution[a] ==
                    arrival_time[a];

     wf8: duration[a] == sum(t in T)
               active[a, t];

     forall(t in T) {
          wf9: t == start[a] => active[a, t]
               == 1;
          wf10: t == end[a] - 1 => active[a,
                t] == 1;
          wf11: t <= start[a] - 1 =>
                     active[a, t] == 0;
          wf12: t >= end[a] => active[a, t]
                     == 0;
          }
```

# The Platform Scheduler behavior is modeled through 5 sets of constraints



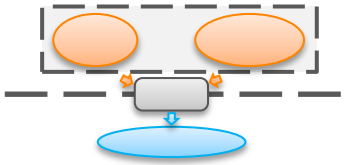
**2. Temporal Ordering: executions, triggering and resource usage relations**

```
/* II. Temporal Ordering constraints */
forall(a in A) {

     forall(a1 in A : a1.task == a.task &&
          a1.execution == a.execution-1)
          to1: start[a] >= end[a1];

     forall(a1 in A : triggers[a.task,
          a1.task] == 1)
          to2: end[a] == arrival_time[a1];

     forall(a1 in A : dependent[a.task,
          a1.task] == 1) {
          to3: start[a] != start[a1];
          to4: start[a] <= start[a1]-1 =>
               start[a1] >= end[a];
          }
     }
```

**3. Multicore: computing capacity of the platform**

```
/* III. Multi-core Constraint */
forall(t in T)
     mc: sum(a in A) active[a, t] <= c;
```

# The Platform Scheduler behavior is modeled through 5 sets of constraints



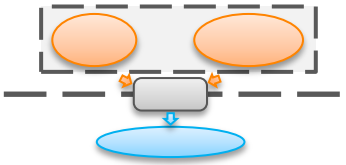
**4. Preemptive Scheduling: priority-driven preemptive scheduling behavior**

```
/* IV. Preemptive Scheduling Constraints */
forall(t in T, a0 in A, a1 in A)
     ps2: (active[a0, t] == 0 &&
           active[a1, t] == 1 &&
           sum(a2 in A) active[a2, t] == c &&
           eligible_for_execution[a0] <= t &&
           end[a0] >= t+1)
               =>
          (priority[a1.task] >=
           priority[a0.task]);
```

**5. Good CPU Usage: scheduler's CPU Usage optimizations**

```
/* V. Good CPU Usage Constraints */
forall(a in A, t in T)
     gcu1: (sum(a1 in A) active[a1, t]<=c-1)
               =>
           (active[a, t] == 1 ||
           eligible_for_execution[a]>= t+1 ||
           end[a] <= t);

forall(a0 in A, a1 in A, t in T : t < tq-1)
     gcu2: (active[a0, t] == 1 &&
            active[a0, t+1] == 0)
               =>
           (end[a0] == t+1 ||
           (active[a1, t+1] == 1 =>
               priority[a1.task] >=
               priority[a0.task]+1));
```

# The Performance Requirement is modeled as an objective function to maximize



**The objective function is a counter for deadline misses**

```
wf13: deadline_miss[a] == end[a] -
      task_execution_deadline[a];

maximize
      sum(a in A)(maxl(0, minl(1, deadline_miss[a])));
```

**Main limitation: each deadline miss is given the same weight in the sum**

$$f \stackrel{\text{def}}{=} \sum_i max(0, min(1, e(a_i) - dl(a_i)))$$

**Potential alternative [1]: non-linear objective function to weight deadlines**

$$\tilde{f} \stackrel{\text{def}}{=} \sum_i 2^{e(a_i) - dl(a_i)}$$

[1] L. Briand, Y. Labiche, and M. Shousha, "Using genetic algorithms for early schedulability analysis and stress testing in real-time systems", Genetic Programming and Evolvable Machines, vol. 7 no. 2, pp. 145-170, 2006

# Correctness was evaluated analyzing the results computed starting from a set of toy examples

| | $j_0$ | $j_1$ | $j_2$ |
|---|---|---|---|
| $exec(j)$ | 2 | 2 | 2 |
| $p(j)$ | 100 | 101 | 102 |
| $dl(j)$ | 3 | 2 | 3 |
| $\max\_ia(j)$ | 3 | 2 | 3 |
| $\min\_dr(j)$<br>$\max\_dr(j)$ | 3 | 2 | 3 |

**In this case, we found a solution where both executions of task $j_0$ miss their deadline**

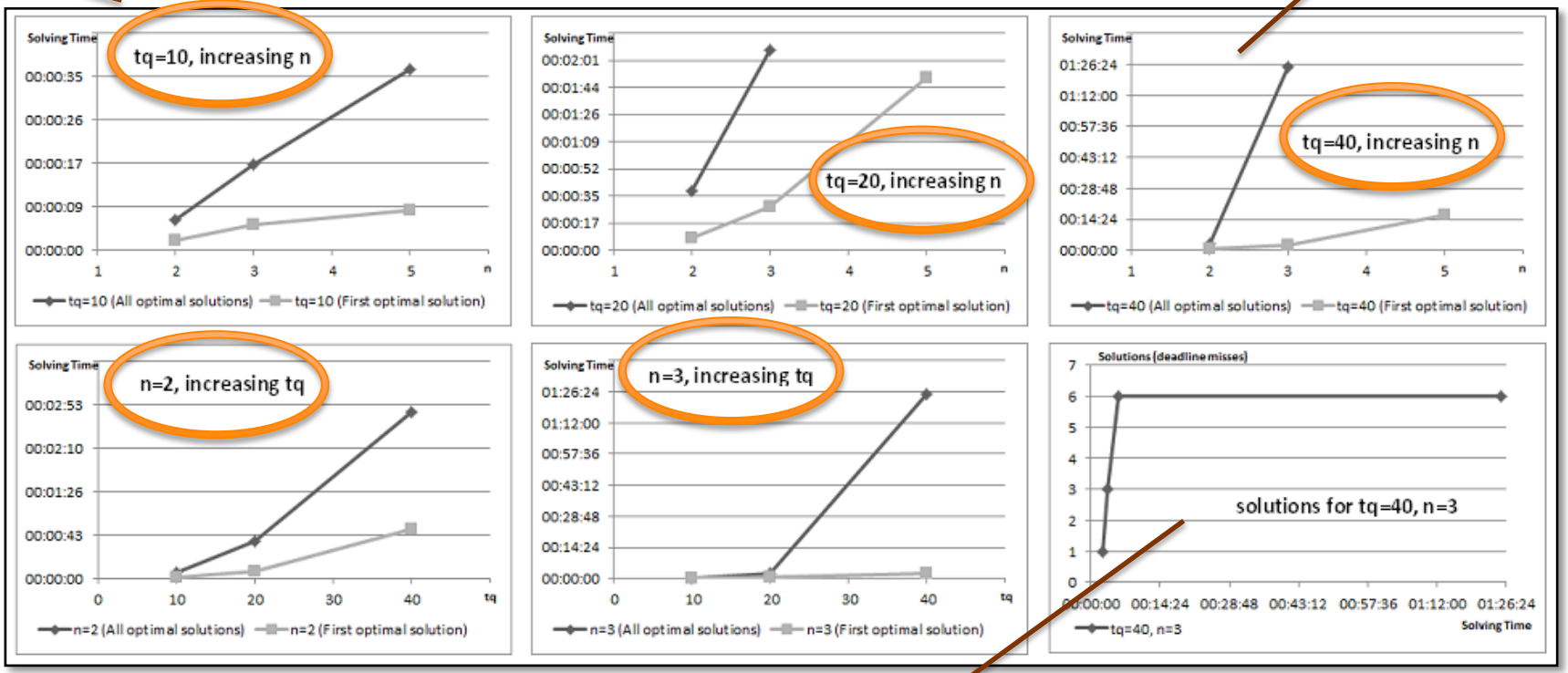
```
arrival_time (at): [0 3 2 4 0 3]
duration (dr): [3 3 2 2 3 3]
eligible_for_execution (efe): [0 7 2 4 0 3]
start (s): [0 7 2 4 0 3]
end (e): [7 10 4 6 3 6]
task_execution_deadline (edl): [3 6 4 6 3 6]
deadline_miss: [4 4 0 0 0 0]
active: [[1 1 0 0 0 0 1 0 0 0]
         [0 0 0 0 0 0 0 1 1 1]
         [0 0 1 1 0 0 0 0 0 0]
         [0 0 0 0 1 1 0 0 0 0]
         [1 1 1 0 0 0 0 0 0 0]
         [0 0 0 1 1 1 0 0 0 0]]
```

# Performance was evaluated by measuring solving times with increasing input size

| | $j_0$ | $j_1$ | $j_2$ |
|---|---|---|---|
| $exec(j)$ | 2 | 2 | 2 |
| $p(j)$ | 100 | 101 | 102 |
| $dl(j)$ | 3 | 2 | 3 |
| $max\_ia(j$ | 3 | 2 | 3 |
| $min\_dr(j$<br>$max\_dr(j$ | 3 | 2 | 3 |

**We evaluated Performance by increasing $n$ and $tq$**

**It took a significant amount of time to find all optimal solutions**




**Most optimal solutions were found shortly after the search started, even if the search took a much more time to terminate**

# Our current work relies on improving the approach scalability with respect to $n$ and $tq$

| | $j_0$ | $j_1$ | $j_2$ |
|---|---|---|---|
| $exec(j)$ | 2 | 2 | 2 |
| $p(j)$ | 100 | 101 | 102 |
| $dl(j)$ | 3 | 2 | 3 |
| $max\_ia(j$ | 3 | 2 | 3 |
| $min\_dr(j$<br>$max\_dr(j$ | 3 | 2 | 3 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $a_0$ | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $a_1$ | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| $a_2$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**Problem: it's hard to compute the $active$ matrix ($2^{n*exec(j_n)*tq}$ possible values)**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $a_0$ | 1 | | | | | | 1 | |
| $a_1$ | | 1 | 1 | 1 | | 1 | | |
| $a_2$ | | | | | 1 | | | |

**Idea: we don't really need the whole matrix, but just to know where the 1's are!**

| | running in |
|---|---|
| $a_0$ | $[0,1), [6,7)$ |
| $a_1$ | $[1,4), [5,6)$ |
| $a_2$ | $[4,5)$ |

# In summary, Constraint Optimization is a promising approach to derive Stress Test Cases for RTES

**System Platform, Tasks and PRs are modeled in a Constraint Program**

**Solving the CP finds tunable values more likely to stress test the system**

**Significant advantages over other approaches encourage future work**

istockphoto.com

**Questions?**