The CPU Usage Constraints in COMET

Stefano Di Alesio, Shiva Nejati
Certus Software V&V Centre
Simula Research Laboratory
Oslo, Norway

September 30, 2011

1 Introduction

In this document we discuss the structure of the constraint program we used to define the schedulability analysis problem of our work [1]. This document contains only the first order logic formalizations for the single interval version of our COMET program, as the ones for the multiple interval can be easily obtained by universally quantifying formulas over the number of iterations. We will present the main part of the COMET code for both the single and the multiple interval versions. Results will be presented for both the non-parallel and the parallel versions of our tool.

Outline The article is organized as follows. Section 2 and 3 respectively present the input values and the variables used during our analysis. Section 4 presents an explanation of the constraints, their formalization in first order logic, and the corresponding COMET code. Section 5 presents the objective functions our complete search. Section 6 discusses the output of our tool.

2 Input values

In this section we describe the input values of our COMET program. Each input value belongs to a semantic set of input values and is presented with:

- A natural language description
- Its COMET implementation code for the single interval version
- Its COMET implementation code for the multiple interval version

General input data

- n denotes the number of threads.

```
int n = 3;
```

```
int n = 3;
```

- J denotes the set of threads.

```
range J = 0..n-1;
```

```
range J = 0..n-1;
```

- m denotes the number of activities.

```
int m = 7;
```

```
int m = 7;
```

- A denotes the set of activities.

```
range A = 0..m-1;
```

```
range A = 0..m-1;
```

- c denotes the number of cores.

```
int c = 1;
```

```
int c = 1;
```

- T denotes the observation time interval.

```
range T;
// [...]
T = 0..max(j in J)(max_ia[j]);
```

```
range T = 0..1000;
```

- i denotes the maximum number of iterations among each thread observed. i is defined only for the multiple interval version.

```
int i = 3;
```

- K denotes the set of iterations. K is defined only for the multiple interval version.

```
range K = 0..i-1;
```

- *cp* denotes the (complete) solver used for solving the constraint optimization problem.

```
Solver<CP> cp();
```

```
Solver<CP> cp();
```

Threads input data

- *min_ia* and *max_ia* respectively denote the minimum and maximum interarrival times for threads. -1 is set for unknown values.

```
int min_ia[J] = [100, -1, 100];
int max_ia[J] = [100, -1, 100];
```

```
int min_ia[J] = [100, -1, 100];
int max_ia[J] = [100, -1, 100];
```

- *min_r* and *max_r* respectively denote the minimum and maximum release times for threads. -1 is set for unknown values.

```
int min_r[J] = [0, 0, 0];
int max_r[J] = [0, 0, 0];
```

```
int min_r[J] = [0, 0, 0];
int max_r[J] = [0, 0, 0];
```

- *priority* denotes the priority level (defined as an integer value) for threads.

```
int priority[J] = [100, 100, 100];
```

```
int priority[J] = [100, 100, 100];
```

Activities input data

- *min_d* and *max_d* respectively denote the minimum and maximum durations for activities.

```
int min_d[A] = [5, 2, 10, 2, 5, 5, 1];
int max_d[A] = [10, 2, 20, 2, 5, 10 , 1];
```


```
int min_d[A] = [5, 2, 10, 2, 5, 5, 1];
int max_d[A] = [10, 2, 20, 2, 5, 10 , 1];
```


- *thread* denotes the thread of each activity.

```
int _thread[A] = [0, 0, 1, 1, 1, 2, 2];
```


```
int _thread[A] = [0, 0, 1, 1, 1, 2, 2];
```


- *delay* denotes the delay time of each activity.

```
int delay[A] = [0, 0, 0, 5, 20, 0, 0];
```


```
int delay[A] = [0, 0, 0, 5, 20, 0, 0];
```


- *temporal_precedence* denotes the temporal precedence matrix for activities. *temporal_precedence* is defined as follows:

$$temporal\_precedence(a_1, a_2) = \begin{cases} 1 & \text{if } a_1 \preceq_t a_2 \\ 0 & \text{otherwise} \end{cases}$$

```
int temporal_precedence[A, A] =
   [[0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0]];
```


```
int temporal_precedence[A, A] =
   [[0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0]];
```


- *data_dependency* denotes the data dependency matrix for activities. *data_dependency* is defined as follows:

$$data\_dependency(a_1, a_2) = \begin{cases} 1 & \text{if } a_1 \preceq_d a_2 \\ 0 & \text{otherwise} \end{cases}$$

```
int data_dependency[A, A] =
   [[0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0]];
```

```
int data_dependency[A, A] =
   [[0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0]];
```

- *loop* denotes the loop matrix for activities. *loop* is defined as follows:

$$loop(a_0, a_q) = \begin{cases} 1 & \text{if } a_0, \ldots, a_q \text{ represents a temporally ordered list of the activities of an infinite while loop} \\ 0 & \text{otherwise} \end{cases}$$

loop is also only defined for the multiple interval version.

```
int loop[A, A] =
   [[0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0]];
```

3 Variables

In this section we describe the variables of our COMET program. Each variable belongs to a semantic set of variables and is presented with:

- A natural language description
- Its COMET implementation code for the single interval version
- Its COMET implementation code for the multiple interval version

Thread variables

- p denotes the period of each thread.

```
var<CP>{int} p[J](cp, T);
```

```
var<CP>{int} p[J](cp, T);
```

- r denotes the release time of each thread.

```
var<CP>{int} r[J](cp, T);
```

```
var<CP>{int} r[J](cp, T);
```

Activity variables

- *eligible_for_execution* denotes the earliest start time for each activity, regardless of the number of cores.

```
var<CP>{int} eligible_for_execution[A](cp, T);
```

```
var<CP>{int} eligible_for_execution[A, K](cp, T);
```

- *start* and *end* respectively denote the start time and end time for each activity.

```
var<CP>{int} start[A](cp, T);
var<CP>{int} end[A](cp, T);

var<CP>{int} start[A, K](cp, T);
var<CP>{int} end[A, K](cp, T);
```

- *active* denotes the execution matrix for each activity. *active* is defined as follows:

$$active(a,t) = \begin{cases} 1 & \text{if the activity } a \text{ is running at the time point } t \\ 0 & \text{otherwise} \end{cases}$$

```
var<CP>{int} active[A, T](cp, 0..1);

var<CP>{int} active[A, T, K](cp, 0..1);
```

4 Constraints

In this section we describe the constraints of our optimization problem. Each constraint belongs to a semantic set of constraints and is presented with:

- A natural language description
- A first order logic formalization
- Its COMET implementation code for the single interval version
- Its COMET implementation code for the multiple interval version

Well-formedness (sanity rules)

- Each activity must finish before the period of its corresponding thread elapses.

$$\forall a \in A \cdot end(a) \leq p(thread(a))$$

```
forall(a in A)
   if(p[_thread[a]] != -1)
      cp.post(end[a] <= p[_thread[a]]);

forall(a in A, k in K)
   if(p[_thread[a]] != -1)
      cp.post(end[a, k] <= (k+1) * p[_thread[a]]);
```

- Each activity cannot start before its earliest start time.

$$\forall a \in A \cdot eligible\_for\_execution(a) \leq start(a)$$

```
forall(a in A)
   cp.post(eligible_for_execution[a] <= start[a]);

forall(a in A, k in K)
   cp.post(eligible_for_execution[a, k] <= start[a, k]);
```

- The number of time points at which an activity is running is bounded by its min/max duration.

$$\forall a \in A \cdot min\_d(a) \leq |d(a)| \leq max\_d(a)$$

```
forall(a in A)
   cp.post(min_duration[a] <= sum(t in T) active[a, t]);
forall(a in A)
   cp.post(sum(t in T) active[a, t] <= max_duration[a]);
```

```
forall(a in A, k in K)
   cp.post(min_duration[a] <= sum(t in T) active[a, t, k]);
forall(a in A, k in K)
   cp.post(sum(t in T) active[a, t, k] <= max_duration[a]);
```

- An activity starts running at its start time, ends just before its end time, and does not run before its start time or after its end time.

$$\begin{aligned}\forall a \in A \cdot \forall t\ 0 \leq t \leq T \cdot \\ & active(a, start(a)) = 1 \wedge \\ & active(a, end(a) - 1) = 1 \wedge \\ & (t < start(a) \vee t \geq end(a)) \Rightarrow active(a, t) = 0\end{aligned}$$

```
forall(a in A)
   cp.post(active[a, start[a]] == 1);
forall(a in A)
   cp.post(active[a, end[a]-1] == 1);
forall(a in A, t in T)
   cp.post(t < start[a] => active[a, t] == 0);
forall(a in A, t in T)
   cp.post(t > end[a]-1 => active[a, t] == 0);

forall(a in A, k in K)
   cp.post(active[a, start[a, k], k] == 1);
forall(a in A, k in K)
   cp.post(active[a, end[a, k]-1, k] == 1);
forall(a in A, t in T, k in K)
   cp.post(t < start[a, k] => active[a, t, k] == 0);
forall(a in A, t in T, k in K)
   cp.post(t > end[a, k]-1 => active[a, t, k] == 0);
```

- The release time of each thread is bounded by its min/max values.

$$\forall j \in J \cdot min\_r(j) \leq r(j) \leq max\_r(j)$$

```
forall(j in J)
   cp.post(min_r[j] <= r[j]);
forall(j in J)
   cp.post(r[j] <= max_r[j]);

forall(j in J)
   cp.post(min_r[j] <= r[j]);
forall(j in J)
   cp.post(r[j] <= max_r[j]);
```

Loop Threads

- Consider activities $a_0^k, \ldots, a_q^k$ representing the activities of iteration k of a thread. Then, for every iteration k, we must have: $start(a_0^{(k+1)}) \geq end(a_q^k) + delay(a_q^k)$. The single-interval version obviously does not include this constraint.

$$\begin{aligned}\forall a_1, a_2 \in A \cdot\ \forall k \in K \cdot \\ & loop(a_1, a_2) \Rightarrow start(a_1, k+1) \geq end(a_2, k) + delay(a_2)\end{aligned}$$

```
forall(a1 in A, a2 in A, k in K.getLow()..K.getUp()-1)
   cp.post((loop[a1, a2] == 1) =>
      start[a1, k+1] >= end[a2, k] + delay[a2]);
```

Temporal Precedence

- For each $a_1, a_2 \in A$ s.t. $a_1 \preceq_t a_2$, we have $start(a_2) - end(a_1) \geq delay(a_1)$.

$$\forall a_1, a_2 \in A \cdot a_1 \preceq_t a_2 \Rightarrow start(a_2) - end(a_1) \geq delay(a_1)$$

```
forall(a1 in A, a2 in A)
   cp.post((temporal_precedence[a1, a2] == 1) =>
      (start[a2] - end[a1] >= delay[a1]));

forall(a1 in A, a2 in A, k in K)
   cp.post((temporal_precedence[a1, a2] == 1) =>
      (start[a2, k] - end[a1, k] >= delay[a1]));
```

Synch/Asynch Communication

- For each $a_1, a_2 \in A$ s.t. $a_1 \preceq_d a_2$, if some conditions that we can statically verify [1] about synchronous communication are met, we have $start(a_2) \geq end(a_1)$.

$$\forall a_1, a_2 \in A \cdot a_1 \preceq_d a_2 \Rightarrow start(a_2) \geq end(a_1)$$

```
forall(a1 in A, a2 in A)
   cp.post((data_dependency[a1, a2] == 1) =>
      (start[a2] >= end[a1]));

forall(a1 in A, a2 in A, k in K)
   cp.post((data_dependency[a1, a2] == 1) =>
      (start[a2, k] >= end[a1, k]));
```

Multi-Core

- The number of running activities at every time point is less than or equal to the number of cores.

$$\forall t \cdot \ 0 \leq t \leq T \Rightarrow \sum_{a \in A} active(a, t) \leq c$$

```
forall(t in T)
   cp.post(sum(a in A) active[a, t] <= c);

forall(a in A, t in T)
   cp.post(active_total[a, t] == sum(k in K) active[a, t, k]);
forall(t in T)
   cp.post(sum(a in A) active_total[a, t] <= c);
```

Scheduling Policy

- Each activity can potentially be preempted.

$$\forall a \in A \cdot end(a) - start(a) \geq |d(a)|$$

```
forall(a in A)
   cp.post(end[a] - start[a] >= sum(t in T) active[a, t]);

forall(a in A, k in K)
   cp.post(end[a, k] - start[a, k] >= sum(t in T) active[a, t, k]);
```

- The earliest time an activity a can start ($eligible\_for\_execution(a)$ or $efe(a)$ for short) is after the arrival time of its corresponding thread and after the earliest termination time of all the activities *preceding* a. Here, *preceding* includes both temporal precedence ($\preceq_t$) and data dependency ($\preceq_d$) orderings.

$$\begin{aligned} \forall a \in A \cdot efe(a) = \max\{&r(thread(a)), \\ &\max_{a_1 \in A}\{efe(a_1) + |d(a_1)| + delay(a_1) : a_1 \preceq_t a\}, \\ &\max_{a_1 \in A}\{efe(a_1) + |d(a_1)| : a_1 \preceq_d a\}\} \end{aligned}$$

```
forall(a in A)
   cp.post(eligible_for_execution[a] == max(
      r[_thread[a]],
      max(
         max(a1 in A)((eligible_for_execution[a1] +
            (sum(t in T) active[a1, t]) + delay[a1]) * temporal_precedence[a1, a]),
         max(a1 in A)((eligible_for_execution[a1] +
            (sum(t in T) active[a1, t])) * data_dependency[a1, a]))));

forall(a in A, k in K)
{
   if(p[_thread[a]] > 0)
      cp.post(eligible_for_execution[a, k] == max(
         p[_thread[a]]*k + r[_thread[a]],
         max(
            max(a1 in A)((eligible_for_execution[a1, k] + (sum(t in T) active[a1, t, k])
               + delay[a1]) * temporal_precedence[a1, a]),
            max(a1 in A)((eligible_for_execution[a1, k] + (sum(t in T) active[a1, t, k]))
               * data_dependency[a1, a]))));
   else
      if(k==K.getLow())
         cp.post(eligible_for_execution[a, k] == max(
            r[_thread[a]],
            max(
               max(a1 in A)((eligible_for_execution[a1, k] + (sum(t in T) active[a1, t, k])
                  + delay[a1]) * temporal_precedence[a1, a]),
               max(a1 in A)((eligible_for_execution[a1, k] + (sum(t in T) active[a1, t, k]))
                  * data_dependency[a1, a]))));
      else
         cp.post(eligible_for_execution[a, k] == max(
            max(a1 in A : _thread[a1]==_thread[a])(eligible_for_execution[a1, k-1] +
                sum(t in T) active[a1, t, k-1] + delay[a1]),
            max(
               max(a1 in A)((eligible_for_execution[a1, k] + (sum(t in T) active[a1, t, k])
                  + delay[a1]) * temporal_precedence[a1, a]),
               max(a1 in A)((eligible_for_execution[a1, k] + (sum(t in T) active[a1, t, k]))
                  * data_dependency[a1, a]))));
}
```

- At any time, if there are two activities that can be scheduled for parallel execution but only one is running, the one that is not running has a lower priority.

$$
\begin{aligned}
\forall a_0, a_1 \in A \cdot \forall t \cdot 0 \le t \le T \\
& (active(a_0, t) = 0 \wedge active(a_1, t) = 1 \wedge \\
& \sum_{a_2 \in A} active(a_2, t) = c \wedge \\
& efe(a_0) \le t \wedge \\
& t \ge \max_{a_2 \in A} \{end(a_2) + delay(a_2) : a_2 \preceq_t a_0\} \wedge \\
& t \ge \max_{a_2 \in A} \{end(a_2) : a_2 \preceq_d a_0\} \wedge \\
& end(a_0) > t) \Rightarrow \\
& priority(a_0) \le priority(a_1)
\end{aligned}
$$

```
forall(t in T, a0 in A, a1 in A)
   cp.post(((active[a0, t] == 0) && (active[a1, t] == 1)
      && ((sum(a2 in A) active[a2, t]) == c)
      && (eligible_for_execution[a0] <= t)
      && (t >= max(max(a2 in A)((end[a2] + delay[a2]) *
         temporal_precedence[a2, a0]), max(a2 in A)(end[a2] * data_dependency[a2, a0]))))
```

```
        && (end[a0] > t))
        => (priority[_thread[a1]] >= priority[_thread[a0]]));

   forall(t in T, a0 in A, a1 in A, k0 in K, k1 in K)
      if(p[_thread[a1]] != -1)
         cp.post(((active[a0, t, k0] == 0) && (active[a1, t, k1] == 1)
            && ((sum(a2 in A) active_total[a2, t]) == c)
            && (eligible_for_execution[a0, k0] <= t)
            && (t >= max(max(a2 in A)((end[a2, k0] + delay[a2]) *
               temporal_precedence[a2, a0]),  max(a2 in A)(end[a2, k0] *
               data_dependency[a2, a0])))
            && (end[a0, k0] > t)
            && ((k1 + 1)*p[_thread[a1]] - 1) >= (t + sum(a2 in A,
               t1 in t..((k1 + 1)*p[_thread[a1]] - 1) : _thread[a2]==_thread[a1])
               active[a2, t1, k1]))
            => (priority[_thread[a1]] >= priority[_thread[a0]]));
      else
         cp.post(((active[a0, t, k0] == 0) && (active[a1, t, k1] == 1)
            && ((sum(a2 in A) active_total[a2, t]) == c)
            && (eligible_for_execution[a0, k0] <= t)
            && (t >= max(max(a2 in A)((end[a2, k0] + delay[a2])
               * temporal_precedence[a2, a0]), max(a2 in A)(end[a2, k0] *
               data_dependency[a2, a0])))
            && (end[a0, k0] > t))
            => (priority[_thread[a1]] >= priority[_thread[a0]]));
```

5 Objective Functions

In this section we describe the objective functions of our optimization problem. Each objective function is presented with:

- A natural language description
- A first order logic formalization
- Its COMET implementation code for the single interval version
- Its COMET implementation code for the multiple interval version

Average CPU Usage The average CPU usage function f_{usage} is defined as the fraction between the sum of all the time points where any activity is running and the total available time on the cores:

$$f_{usage} \stackrel{def}{=} \frac{\sum_{a \in A, t \in T} active(a,t)}{T * c}$$

Our objective is to capture high usage of CPU, i.e. to maximize f_{usage}. The denominator of the fraction is a constant value, so only the numerator is maximized.

```
maximize<cp>
   sum(a in A, t in T)(active[a, t])

maximize<cp>
   sum(a in A, t in T)(active_total[a, t])
```

Makespan The makespan function $f_{makespan}$ is defined as the time it takes for all the activities in a set of threads to terminate, counting from the arrival time of the first thread of the set:

$$f_{makespan} \stackrel{def}{=} \max_{a \in A}\{end(a)\} - \min_{j \in J}\{r(j)\}$$

Our objective is to capture high values of makespan, i.e. to maximize $f_{makespan}$.

```
maximize<cp>
   max(a in A)(end[a]) - min(j in J)(r[j])

maximize<cp>
   max(a in A, k in K)(end[a, k]) - min(j in J)(r[j])
```

6 Outputs

In this section we describe the output of our tool for both the non-parallel and the parallel single interval versions of our tool. The output record is structured as follows:

- A time print reporting the start time of the computation
- The *Input* block showing a summary of the input data used during the computation, showing:
  - The number of cores c
  - The set of threads J
  - The set of activities A
  - The observation time interval T
  - The minimum and maximum interarrival time for each thread min_ia and max_ia
  - The minimum and maximum release time for each thread min_r and max_r
  - The priority for each thread *priority*
  - The minimum and maximum duration for each activity min_d and max_d
  - The thread for each activity *job*
  - The delay for each activity *delay*
  - The temporal precedence matrix *temporal_precedence*
  - The data dependency matrix *data_dependency*
- The *Output* block showing a list of the feasible solutions found during the computation, ordered by their computation time and thus by increasing values of the objective function. Each solution features:
  - A time print reporting the time of the solution computation
  - The period of each thread p
  - The release time of each thread r
  - The start time for each activity *start*
  - The end time for each activity *end*
  - The earliest execution time for each activity $eligible\_for\_execution$
  - The delay for each activity *delay*
  - The active matrix *active*, where only the time points t s.t. $active(a, t) = 1$ are shown and where each row reports the duration of the corresponding activity
  - The value the objective function has in the solution
- A time print reporting the end time of the computation

CPU Usage, Non-Parallel Version

```
Start: 16:58:43 W. Europe Daylight Time
Input:
c: 1
J: 0..2
A: 0..6
T: 0..100
min_ia[100,50,100]
max_ia[100,100,100]
min_r[0,0,0]
max_r[0,0,0]
priority[100,100,100]
```

```
min_d[5,2,10,2,5,5,1]
max_d[10,2,20,2,5,10,1]
job[0,0,1,1,1,2,2]
delay[(101)[0..100],(101)[0..100],(101)[0..100],(101)[0..100],
   (101)[0..100],(101)[0..100],(101)[0..100]]
TP: anonymous[0,1,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,1,0,0,0]
TP: anonymous[0,0,0,0,1,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,1]
TP: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,1,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,1]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
Output:
18:12:8 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[30,37,10,35,44,0,49]
end[35,39,30,37,49,10,50]
eligible_for_execution[0,5,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34] [5]
ACT: anonymous[37,38] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[35,36] [2]
ACT: anonymous[44,45,46,47,48] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[49] [1]
---------------------------------
objective tightened to: 45
18:12:50 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[30,38,10,36,45,0,50]
end[36,40,30,38,50,10,51]
eligible_for_execution[0,6,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35] [6]
ACT: anonymous[38,39] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[36,37] [2]
ACT: anonymous[45,46,47,48,49] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[50] [1]
---------------------------------
objective tightened to: 46
18:13:31 W. Europe Daylight Time
p[100,(50)[51..100],100]
r[0,0,0]
start[30,39,10,37,46,0,51]
end[37,41,30,39,51,10,52]
eligible_for_execution[0,7,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35,36] [7]
```

```
ACT: anonymous[39,40] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[37,38] [2]
ACT: anonymous[46,47,48,49,50] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[51] [1]
---------------------------------
objective tightened to: 47
18:14:11 W. Europe Daylight Time
p[100,(49)[52..100],100]
r[0,0,0]
start[30,40,10,38,47,0,52]
end[38,42,30,40,52,10,53]
eligible_for_execution[0,8,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35,36,37] [8]
ACT: anonymous[40,41] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[38,39] [2]
ACT: anonymous[47,48,49,50,51] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[52] [1]
---------------------------------
objective tightened to: 48
18:14:50 W. Europe Daylight Time
p[100,(48)[53..100],100]
r[0,0,0]
start[30,41,10,39,48,0,53]
end[39,43,30,41,53,10,54]
eligible_for_execution[0,9,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35,36,37,38] [9]
ACT: anonymous[41,42] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[39,40] [2]
ACT: anonymous[48,49,50,51,52] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[53] [1]
---------------------------------
objective tightened to: 49
18:15:29 W. Europe Daylight Time
p[100,(47)[54..100],100]
r[0,0,0]
start[30,42,10,40,49,0,54]
end[40,44,30,42,54,10,55]
eligible_for_execution[0,10,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35,36,37,38,39] [10]
ACT: anonymous[42,43] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[40,41] [2]
ACT: anonymous[49,50,51,52,53] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[54] [1]
---------------------------------
objective tightened to: 50
End:  7:32:53 W. Europe Daylight Time
```

Makespan, Non-Parallel Version

```
Start: 16:10:21 W. Europe Daylight Time
```

```
Input:
c: 1
J: 0..2
A: 0..6
T: 0..100
min_p[100,50,100]
max_p[100,100,100]
min_r[0,0,0]
max_r[0,0,0]
priority[100,100,100]
min_d[5,2,10,2,5,5,1]
max_d[10,2,20,2,5,10,1]
_thread[0,0,1,1,1,2,2]
min_delay[0,0,0,3,15,0,0]
max_delay[0,0,0,7,25,0,0]
TP: anonymous[0,1,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,1,0,0,0]
TP: anonymous[0,0,0,0,1,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,1]
TP: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,1,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,1]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
Output:
17:21:57 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[30,37,10,35,44,0,49]
end[35,39,30,37,49,10,50]
eligible_for_execution[0,5,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34] [5]
ACT: anonymous[37,38] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[35,36] [2]
ACT: anonymous[44,45,46,47,48] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[49] [1]
---------------------------------
objective tightened to: 50
17:22:5 W. Europe Daylight Time
p[100,(50)[51..100],100]
r[0,0,0]
start[30,35,10,37,46,0,51]
end[35,37,30,39,51,10,52]
eligible_for_execution[0,5,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34] [5]
ACT: anonymous[35,36] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[37,38] [2]
ACT: anonymous[46,47,48,49,50] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[51] [1]
```

```
---------------------------------
objective tightened to: 52
17:22:46 W. Europe Daylight Time
p[100,(49)[52..100],100]
r[0,0,0]
start[30,36,10,38,47,0,52]
end[36,38,30,40,52,10,53]
eligible_for_execution[0,6,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35] [6]
ACT: anonymous[36,37] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[38,39] [2]
ACT: anonymous[47,48,49,50,51] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[52] [1]
---------------------------------
objective tightened to: 53
17:23:27 W. Europe Daylight Time
p[100,(48)[53..100],100]
r[0,0,0]
start[30,37,10,39,48,0,53]
end[37,39,30,41,53,10,54]
eligible_for_execution[0,7,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35,36] [7]
ACT: anonymous[37,38] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[39,40] [2]
ACT: anonymous[48,49,50,51,52] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[53] [1]
---------------------------------
objective tightened to: 54
17:24:6 W. Europe Daylight Time
p[100,(47)[54..100],100]
r[0,0,0]
start[30,38,10,40,49,0,54]
end[38,40,30,42,54,10,55]
eligible_for_execution[0,8,0,20,29,0,34]
delay[0,0,0,7,(11)[15..25],0,0]
ACT: anonymous[30,31,32,33,34,35,36,37] [8]
ACT: anonymous[38,39] [2]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29] [20]
ACT: anonymous[40,41] [2]
ACT: anonymous[49,50,51,52,53] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[54] [1]
---------------------------------
objective tightened to: 55
End:  6:23:33 W. Europe Daylight Time
```

CPU Usage, Parallel Version

```
Start: 12:17:12 W. Europe Daylight Time
Input:
c: 1
J: 0..2
A: 0..6
T: 0..100
min_p[100,50,100]
```

```
max_p[100,100,100]
min_r[0,0,0]
max_r[0,0,0]
priority[100,100,100]
min_d[5,2,10,2,5,5,1]
max_d[10,2,20,2,5,10,1]
_thread[0,0,1,1,1,2,2]
TP: anonymous[0,1,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,1,0,0,0]
TP: anonymous[0,0,0,0,1,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,1]
TP: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,1,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,1]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
Output:
12:19:9 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[6,33,11,31,38,0,43]
end[11,35,31,33,43,6,44]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[6,7,8,9,10] [5]
ACT: anonymous[33,34] [2]
ACT: anonymous[11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30] [20]
ACT: anonymous[31,32] [2]
ACT: anonymous[38,39,40,41,42] [5]
ACT: anonymous[0,1,2,3,4,5] [6]
ACT: anonymous[43] [1]
---------------------------------
objective tightened to: 41
12:19:14 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,36,14,34,41,0,46]
end[14,38,34,36,46,9,47]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13] [5]
ACT: anonymous[36,37] [2]
ACT: anonymous[14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33] [20]
ACT: anonymous[34,35] [2]
ACT: anonymous[41,42,43,44,45] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[46] [1]
---------------------------------
objective tightened to: 44
12:23:2 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,37,15,35,42,0,47]
end[15,39,35,37,47,9,48]
eligible_for_execution[0,6,0,20,27,0,32]
```

```
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14] [6]
ACT: anonymous[37,38] [2]
ACT: anonymous[15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34] [20]
ACT: anonymous[35,36] [2]
ACT: anonymous[42,43,44,45,46] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[47] [1]
---------------------------------
objective tightened to: 45
Output:
12:26:49 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,38,16,36,43,0,48]
end[16,40,36,38,48,9,49]
eligible_for_execution[0,7,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15] [7]
ACT: anonymous[38,39] [2]
ACT: anonymous[16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35] [20]
ACT: anonymous[36,37] [2]
ACT: anonymous[43,44,45,46,47] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[48] [1]
---------------------------------
objective tightened to: 46
12:30:31 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,39,17,37,44,0,49]
end[17,41,37,39,49,9,50]
eligible_for_execution[0,8,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15,16] [8]
ACT: anonymous[39,40] [2]
ACT: anonymous[17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36] [20]
ACT: anonymous[37,38] [2]
ACT: anonymous[44,45,46,47,48] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[49] [1]
---------------------------------
objective tightened to: 47
12:34:9 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,40,18,38,45,0,50]
end[18,42,38,40,50,9,51]
eligible_for_execution[0,9,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15,16,17] [9]
ACT: anonymous[40,41] [2]
ACT: anonymous[18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37] [20]
ACT: anonymous[38,39] [2]
ACT: anonymous[45,46,47,48,49] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[50] [1]
---------------------------------
objective tightened to: 48
12:37:43 W. Europe Daylight Time
```

```
p[100,(50)[51..100],100]
r[0,0,0]
start[9,41,19,39,46,0,51]
end[19,43,39,41,51,9,52]
eligible_for_execution[0,10,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15,16,17,18] [10]
ACT: anonymous[41,42] [2]
ACT: anonymous[19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38] [20]
ACT: anonymous[39,40] [2]
ACT: anonymous[46,47,48,49,50] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[51] [1]
--------------------------------
objective tightened to: 49
13:17:19 W. Europe Daylight Time
p[100,(49)[52..100],100]
r[0,0,0]
start[10,42,20,40,47,0,52]
end[20,44,40,42,52,10,53]
eligible_for_execution[0,10,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19] [10]
ACT: anonymous[42,43] [2]
ACT: anonymous[20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39] [20]
ACT: anonymous[40,41] [2]
ACT: anonymous[47,48,49,50,51] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[52] [1]
--------------------------------
objective tightened to: 50
End: 15:13:20 W. Europe Daylight Time
```

Makespan, Parallel Version

```
Start:  9:19:27 W. Europe Daylight Time
Input:
c: 1
J: 0..2
A: 0..6
T: 0..100
min_p[100,50,100]
max_p[100,100,100]
min_r[0,0,0]
max_r[0,0,0]
priority[100,100,100]
min_d[5,2,10,2,5,5,1]
max_d[10,2,20,2,5,10,1]
_thread[0,0,1,1,1,2,2]
TP: anonymous[0,1,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,1,0,0,0]
TP: anonymous[0,0,0,0,1,0,0]
TP: anonymous[0,0,0,0,0,0,0]
TP: anonymous[0,0,0,0,0,0,1]
TP: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,1,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,1]
```

```
DD: anonymous[0,0,0,0,0,0,0]
DD: anonymous[0,0,0,0,0,0,0]
Output:
9:21:29 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[7,34,12,32,39,0,44]
end[12,36,32,34,44,7,45]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[7,8,9,10,11] [5]
ACT: anonymous[34,35] [2]
ACT: anonymous[12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31] [20]
ACT: anonymous[32,33] [2]
ACT: anonymous[39,40,41,42,43] [5]
ACT: anonymous[0,1,2,3,4,5,6] [7]
ACT: anonymous[44] [1]
---------------------------------
objective tightened to: 45
9:21:31 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[8,35,13,33,40,0,45]
end[13,37,33,35,45,8,46]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[8,9,10,11,12] [5]
ACT: anonymous[35,36] [2]
ACT: anonymous[13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32] [20]
ACT: anonymous[33,34] [2]
ACT: anonymous[40,41,42,43,44] [5]
ACT: anonymous[0,1,2,3,4,5,6,7] [8]
ACT: anonymous[45] [1]
---------------------------------
objective tightened to: 46
9:21:39 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,36,14,34,41,0,46]
end[14,38,34,36,46,9,47]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13] [5]
ACT: anonymous[36,37] [2]
ACT: anonymous[14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33] [20]
ACT: anonymous[34,35] [2]
ACT: anonymous[41,42,43,44,45] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[46] [1]
---------------------------------
objective tightened to: 47
9:21:41 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[8,33,13,35,42,0,47]
end[13,35,33,37,47,8,48]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[8,9,10,11,12] [5]
ACT: anonymous[33,34] [2]
```

```
ACT: anonymous[13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32] [20]
ACT: anonymous[35,36] [2]
ACT: anonymous[42,43,44,45,46] [5]
ACT: anonymous[0,1,2,3,4,5,6,7] [8]
ACT: anonymous[47] [1]
--------------------------------
objective tightened to: 48
9:21:49 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,34,14,36,43,0,48]
end[14,36,34,38,48,9,49]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13] [5]
ACT: anonymous[34,35] [2]
ACT: anonymous[14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33] [20]
ACT: anonymous[36,37] [2]
ACT: anonymous[43,44,45,46,47] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[48] [1]
--------------------------------
objective tightened to: 49
9:23:50 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[10,35,15,37,44,0,49]
end[15,37,35,39,49,10,50]
eligible_for_execution[0,5,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[10,11,12,13,14] [5]
ACT: anonymous[35,36] [2]
ACT: anonymous[15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34] [20]
ACT: anonymous[37,38] [2]
ACT: anonymous[44,45,46,47,48] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[49] [1]
--------------------------------
objective tightened to: 50
9:29:21 W. Europe Daylight Time
p[100,(51)[50..100],100]
r[0,0,0]
start[9,36,16,38,45,0,50]
end[16,38,36,40,50,9,51]
eligible_for_execution[0,7,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15] [7]
ACT: anonymous[36,37] [2]
ACT: anonymous[16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35] [20]
ACT: anonymous[38,39] [2]
ACT: anonymous[45,46,47,48,49] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[50] [1]
--------------------------------
objective tightened to: 51
9:33:6 W. Europe Daylight Time
p[100,(50)[51..100],100]
r[0,0,0]
start[9,37,17,39,46,0,51]
end[17,39,37,41,51,9,52]
```

```
eligible_for_execution[0,8,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15,16] [8]
ACT: anonymous[37,38] [2]
ACT: anonymous[17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36] [20]
ACT: anonymous[39,40] [2]
ACT: anonymous[46,47,48,49,50] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[51] [1]
---------------------------------
objective tightened to: 52
9:36:41 W. Europe Daylight Time
p[100,(49)[52..100],100]
r[0,0,0]
start[9,38,18,40,47,0,52]
end[18,40,38,42,52,9,53]
eligible_for_execution[0,9,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15,16,17] [9]
ACT: anonymous[38,39] [2]
ACT: anonymous[18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37] [20]
ACT: anonymous[40,41] [2]
ACT: anonymous[47,48,49,50,51] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[52] [1]
---------------------------------
objective tightened to: 53
9:40:13 W. Europe Daylight Time
p[100,(48)[53..100],100]
r[0,0,0]
start[9,39,19,41,48,0,53]
end[19,41,39,43,53,9,54]
eligible_for_execution[0,10,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[9,10,11,12,13,14,15,16,17,18] [10]
ACT: anonymous[39,40] [2]
ACT: anonymous[19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38] [20]
ACT: anonymous[41,42] [2]
ACT: anonymous[48,49,50,51,52] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8] [9]
ACT: anonymous[53] [1]
---------------------------------
objective tightened to: 54
9:58:19 W. Europe Daylight Time
p[100,(47)[54..100],100]
r[0,0,0]
start[10,40,20,42,49,0,54]
end[20,42,40,44,54,10,55]
eligible_for_execution[0,10,0,20,27,0,32]
delay[0,0,0,5,20,0,0]
ACT: anonymous[10,11,12,13,14,15,16,17,18,19] [10]
ACT: anonymous[40,41] [2]
ACT: anonymous[20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39] [20]
ACT: anonymous[42,43] [2]
ACT: anonymous[49,50,51,52,53] [5]
ACT: anonymous[0,1,2,3,4,5,6,7,8,9] [10]
ACT: anonymous[54] [1]
---------------------------------
objective tightened to: 55
End: 12:14:8 W. Europe Daylight Time
```

References

[1] Shiva Nejati, Stefano Di Alesio, Mehrdad Sabetzadeh, Lionel Briand. *"Modeling and Analysis of CPU Usage in Safety-Critical Embedded Systems to Support Stress Testing"*, 2011