Approximation of the value in a weighted timed game

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Joint work with Samy Jaziri and Nicolas Markey
An example: The task graph scheduling problem

Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

$P_1$ (fast):

<table>
<thead>
<tr>
<th>time</th>
<th>2 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td></td>
</tr>
<tr>
<td>$\times$</td>
<td>3 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
</tr>
<tr>
<td>in use</td>
</tr>
</tbody>
</table>

$P_2$ (slow):

<table>
<thead>
<tr>
<th>time</th>
<th>5 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td></td>
</tr>
<tr>
<td>$\times$</td>
<td>7 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
</tr>
<tr>
<td>in use</td>
</tr>
</tbody>
</table>

\[2/24\]
An example: The task graph scheduling problem

Compute \( D \times (C \times (A + B)) + (A + B) + (C \times D) \) using two processors:

\[
P_1 \text{ (fast)}: \quad \begin{array}{|c|c|}
\hline
& \text{time} \\
+ & 2 \text{ picoseconds} \\
\times & 3 \text{ picoseconds} \\
\hline
\end{array}
\quad \begin{array}{|c|c|}
\hline
& \text{energy} \\
\text{idle} & 10 \text{ Watt} \\
in \text{ use} & 90 \text{ Watts} \\
\hline
\end{array}
\]

\[
P_2 \text{ (slow)}: \quad \begin{array}{|c|c|}
\hline
& \text{time} \\
+ & 5 \text{ picoseconds} \\
\times & 7 \text{ picoseconds} \\
\hline
\end{array}
\quad \begin{array}{|c|c|}
\hline
& \text{energy} \\
\text{idle} & 20 \text{ Watts} \\
in \text{ use} & 30 \text{ Watts} \\
\hline
\end{array}
\]
An example: The task graph scheduling problem

Compute \( D \times (C \times (A+B)) + (A+B) + (C \times D) \) using two processors:

**\( P_1 \) (fast):**

<table>
<thead>
<tr>
<th></th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>2 ps</td>
</tr>
<tr>
<td>( \times )</td>
<td>3 ps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>10 Watt</td>
</tr>
<tr>
<td>in use</td>
<td>90 Watts</td>
</tr>
</tbody>
</table>

**\( P_2 \) (slow):**

<table>
<thead>
<tr>
<th></th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>5 ps</td>
</tr>
<tr>
<td>( \times )</td>
<td>7 ps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>20 Watts</td>
</tr>
<tr>
<td>in use</td>
<td>30 Watts</td>
</tr>
</tbody>
</table>

**Sch1**

- \( T_1 \): 2 ps
- \( T_2 \): 3 ps
- \( T_3 \): 5 ps
- \( T_4 \): 7 ps
- \( T_5 \): 10 ps
- \( T_6 \): 19 ps

**Sch2**

- \( T_1 \): 2 ps
- \( T_2 \): 3 ps
- \( T_3 \): 5 ps
- \( T_4 \): 7 ps
- \( T_5 \): 10 ps
- \( T_6 \): 13 ps

1. 37 nanojoules

1. 32 nanojoules
An example: The task graph scheduling problem

Compute $D \times (C \times (A+B)) + (A+B) + (C \times D)$ using two processors:

**P₁ (fast):**

<table>
<thead>
<tr>
<th>Time</th>
<th>2 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>10 Watt</td>
</tr>
<tr>
<td>In use</td>
<td>90 Watts</td>
</tr>
</tbody>
</table>

**P₂ (slow):**

<table>
<thead>
<tr>
<th>Time</th>
<th>5 picoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 picoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>20 Watts</td>
</tr>
<tr>
<td>In use</td>
<td>30 Watts</td>
</tr>
</tbody>
</table>

The task graph scheduling problem can be visualized with a diagram showing the execution order and time taken for each task. The diagram includes nodes for tasks $T₁$, $T₂$, $T₃$, $T₄$, $T₅$, and $T₆$, with arrows indicating the dependencies and the order in which the tasks are executed. The table shows the execution time for each node, with a possible idle time of 5 picoseconds, and the energy consumption for idle and in use states for processors $P₁$ and $P₂$. The diagram also highlights the scheduling scenarios (Sch₁, Sch₂, Sch₃) with respective execution times and energy consumption for tasks $T₁$, $T₂$, $T₃$, $T₄$, $T₅$, and $T₆$.
The model of timed automata
The model of timed automata

\[
\begin{align*}
\text{safe} & \quad \longrightarrow \quad \text{problem, } x:=0 \\
\text{problem} & \quad \longrightarrow \quad \text{alarm, } y:=0 \\
\text{alarm} & \quad \longrightarrow \quad \text{repair, } x \leq 15 \\
\text{repair} & \quad \longrightarrow \quad \text{failsafe, } \text{delayed, } y:=0 \\
\text{failsafe} & \quad \longrightarrow \quad \text{repair, } 2 \leq y \wedge x \leq 56 \\
\end{align*}
\]

\[
\begin{array}{cccccccccc}
\text{safe} & \quad \xrightarrow{23} & \quad \text{safe} & \quad \xrightarrow{\text{problem}} & \quad \text{alarm} & \quad \xrightarrow{15.6} & \quad \text{alarm} & \quad \xrightarrow{\text{delayed}} & \quad \text{failsafe} \\
x & \quad 0 & \quad 23 & \quad 0 & \quad 15.6 & \quad 15.6 & \quad \ldots \\
y & \quad 0 & \quad 23 & \quad 23 & \quad 38.6 & \quad 0 & \quad \ldots \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
\text{failsafe} & \quad \xrightarrow{2.3} & \quad \text{failsafe} & \quad \xrightarrow{\text{repair}} & \quad \text{repairing} & \quad \xrightarrow{22.1} & \quad \text{repairing} & \quad \xrightarrow{\text{done}} & \quad \text{safe} \\
\ldots & \quad 15.6 & \quad 17.9 & \quad 17.9 & \quad 40 & \quad 40 \\
0 & \quad 2.3 & \quad 0 & \quad 22.1 & \quad 22.1 & \quad \ldots \\
\end{array}
\]
Modelling the task graph scheduling problem
Modelling the task graph scheduling problem

- Processors

\[ P_1: \]
\[
\begin{align*}
\text{idle} & \quad x:=0 \quad \text{add}_1 \\
(x \leq 2) & \quad x=2 \quad \text{done}_1 \\
& \quad \text{idle} \\
& \quad x:=0 \quad \text{mult}_1 \\
& \quad x=3 \quad \text{done}_1 \\
& \quad (x \leq 3) \\
& \quad \text{idle} \\
& \quad x:=0 \\
& \quad \text{mult}_1
\end{align*}
\]

\[ P_2: \]
\[
\begin{align*}
\text{idle} & \quad x:=0 \quad \text{add}_2 \\
(y \leq 5) & \quad y=5 \quad \text{done}_2 \\
& \quad \text{idle} \\
& \quad x:=0 \quad \text{mult}_2 \\
& \quad y=7 \quad \text{done}_2 \\
& \quad (y \leq 7) \\
& \quad \text{idle} \\
& \quad x:=0 \\
& \quad \text{mult}_2
\end{align*}
\]

Modelling energy

\[ P_1: \]
\[
\begin{align*}
& + \quad \text{idle} \\
& \quad x:=0 \quad \text{add}_1 \\
& \quad x=2 \quad \text{done}_1 \\
& \quad \text{idle} \\
& \quad x:=0 \quad \text{mult}_1 \\
& \quad x=3 \quad \text{done}_1 \\
& \quad \text{idle} \\
& \quad x:=0 \quad \text{mult}_1 \\
& \quad (x \leq 2) \times (x \leq 3)
\end{align*}
\]

\[ P_2: \]
\[
\begin{align*}
& + \quad \text{idle} \\
& \quad y:=0 \quad \text{add}_2 \\
& \quad y=5 \quad \text{done}_2 \\
& \quad \text{idle} \\
& \quad y:=0 \quad \text{mult}_2 \\
& \quad y=7 \quad \text{done}_2 \\
& \quad \text{idle} \\
& \quad y:=0 \quad \text{mult}_2 \\
& \quad (y \leq 5) \times (y \leq 7)
\end{align*}
\]

Modelling uncertainty
Modelling the task graph scheduling problem

**Processes**

- **P₁**: 
  - \( x = 2 \) \( \rightarrow \) \( x = 3 \)
  - \( x = 0 \)
  - \( (x \leq 2) \)

- **P₂**: 
  - \( y = 5 \) \( \rightarrow \) \( y = 7 \)
  - \( x = 0 \)
  - \( (y \leq 5) \)

**Tasks**

- **T₄**: 
  - \( t_1 \land t_2 \)
  - \( t_4 \leftarrow 1 \)

- **T₅**: 
  - \( t_3 \)
  - \( t_5 \leftarrow 1 \)

A schedule is a path in the product automaton
Modelling the task graph scheduling problem

- **Processors**
  - $P_1$: $x=2$ (idle) $\times (x \leq 2)$
  - $x:=0$ $\text{add}_1$
  - $x:=0$ $\text{done}_1$
  - $x:=0$ $\times (x \leq 3)$
  - $x:=0$ $\text{mult}_1$

- **Tasks**
  - $T_4$: $t_1 \land t_2$ $\rightarrow t_4:=1$
  - $t_3$ $\rightarrow t_5:=1$

- **Processors**
  - $P_2$: $y=5$ (idle) $\times (y \leq 5)$
  - $y:=0$ $\text{add}_2$
  - $y:=0$ $\text{done}_2$
  - $x:=0$ $\times (y \leq 7)$
  - $x:=0$ $\text{mult}_2$

- **Modelling energy**
  - $P_1$: $x=2$ $\times 90$ $\times (x \leq 2)$
  - $x:=0$ $\text{add}_1$
  - $x:=0$ $\text{done}_1$
  - $x:=0$ $\times 90$ $\times (x \leq 3)$
  - $x:=0$ $\text{mult}_1$

  - $P_2$: $y=5$ $\times 30$ $\times (y \leq 5)$
  - $y:=0$ $\text{add}_2$
  - $y:=0$ $\text{done}_2$
  - $x:=0$ $\times 30$ $\times (y \leq 7)$
  - $x:=0$ $\text{mult}_2$

- **A good schedule is a path in the product automaton with a low cost**
Modelling the task graph scheduling problem

- **Processors**
  - \(P_1\):
    - \(x = 2\) \(\rightarrow\) \(x = 3\)
    - \(y = 5\)
  - \(P_2\):
    - \(y = 5\)

- **Modelling energy**
  - \(P_1\):
    - \(x = 2\)
    - \(y = 5\)
  - \(P_2\):
    - \(y = 5\)

- **Tasks**
  - \(T_4\):
    - \(t_1 \land t_2\)
  - \(T_5\):
    - \(t_3\)

- **Modelling uncertainty**
  - \(P_1\):
    - \(x \geq 1\)
  - \(P_2\):
    - \(y \geq 3\)
Modelling the task graph scheduling problem

- **Processors**
  - \( P_1 \):
    - \( x = 2 \):
      - \( \text{add}_1 \)
      - \( \text{done}_1 \)
    - \( x = 3 \):
      - \( \text{mult}_1 \)
    - \( x = 0 \):
      - \( \text{idle} \)
    - \( x \leq 2 \):
      - \( +10 \)
    - \( x \leq 3 \):
      - \( +90 \)
  - \( P_2 \):
    - \( y = 5 \):
      - \( \text{add}_2 \)
      - \( \text{done}_2 \)
    - \( y = 7 \):
      - \( \text{mult}_2 \)
    - \( y = 0 \):
      - \( \text{idle} \)
    - \( y \leq 5 \):
      - \( +30 \)
    - \( y \leq 7 \):
      - \( +20 \)

- **Tasks**
  - \( T_4 \):
    - \( t_1 \land t_2 \):
      - \( \text{add}_i \)
      - \( \text{done}_i \)
    - \( t_4 = 1 \):
  - \( T_5 \):
    - \( t_3 \):
      - \( \text{add}_i \)
      - \( \text{done}_i \)
    - \( t_5 = 1 \):

- **Modelling energy**
  - \( P_1 \):
    - \( x = 2 \):
      - \( \text{add}_1 \)
      - \( \text{done}_1 \)
    - \( x = 3 \):
      - \( \text{mult}_1 \)
    - \( x = 0 \):
      - \( \text{idle} \)
    - \( x \leq 2 \):
      - \( +90 \)
    - \( x \leq 3 \):
      - \( +10 \)
  - \( P_2 \):
    - \( y = 5 \):
      - \( \text{add}_2 \)
      - \( \text{done}_2 \)
    - \( y = 7 \):
      - \( \text{mult}_2 \)
    - \( y = 0 \):
      - \( \text{idle} \)
    - \( y \leq 5 \):
      - \( +30 \)
    - \( y \leq 7 \):
      - \( +20 \)

- **Modelling uncertainty**
  - \( P_1 \):
    - \( x \geq 1 \):
      - \( \text{add}_1 \)
      - \( \text{done}_1 \)
    - \( x = 0 \):
      - \( \text{idle} \)
    - \( x \geq 1 \):
      - \( +90 \)
    - \( x \leq 2 \):
      - \( +10 \)
    - \( x \leq 3 \):
      - \( +90 \)
  - \( P_2 \):
    - \( y \geq 3 \):
      - \( \text{add}_2 \)
      - \( \text{done}_2 \)
    - \( y = 0 \):
      - \( \text{idle} \)
    - \( y \geq 2 \):
      - \( +30 \)
    - \( x \geq 1 \):
      - \( \text{add}_1 \)
      - \( \text{done}_1 \)
    - \( x = 0 \):
      - \( \text{idle} \)
    - \( y \geq 2 \):
      - \( +20 \)
    - \( y \leq 3 \):
      - \( +30 \)
    - \( x \leq 2 \):
      - \( +20 \)
    - \( x \leq 3 \):
      - \( +30 \)
Weighted/priced timed automata [ALP01,BFH+01]

That can be generalized!
Weighted/priced timed automata

\[ \ell_0 + 5 \rightarrow \ell_1 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \xrightarrow{(y=0)} \ell_2 + 10 \rightarrow \ell_3 + 1 \rightarrow \text{smiley face} \]

\[ \ell_2 \xrightarrow{x=2, c} \ell_3 + 7 \]

That can be generalized!
Weighted/priced timed automata

\[
\begin{align*}
\ell_0 & \xrightarrow{x \leq 2, c, y := 0} \ell_1 \\
\ell_1 & \xrightarrow{u} \ell_2 \xrightarrow{x = 2, c} \ell_3 \xrightarrow{c} 7 \\
& \xrightarrow{+1} \ell_3 \xrightarrow{c} \ell_2 \xrightarrow{+10} \\
& \xrightarrow{+7} \bullet \\
\end{align*}
\]

\[
\begin{array}{c|ccc|ccc|ccc|c}
 & \ell_0 & \ell_0 & \ell_1 & \ell_3 & \ell_3 & \ell_2 & \ell_2 & \ell_3 & \ell_3 & \ell_0 \\
\hline
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 1.3 & 1.3 & 2 & 0 & 0 \\
y & 0 & 1.3 & 0 & 0 & 0.7 & 0 & 0.7 & 0 & 0 & 0.7 \\
\end{array}
\]

cost :

That can be generalized!
Weighted/priced timed automata

\[ \ell_0 + 5 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \xrightarrow{(y = 0)} \ell_2 \xrightarrow{+10} \ell_3 \xrightarrow{x = 2, c +1} \ell_4 \xrightarrow{x = 2, c +7} \]

\[ \ell_0 \xrightarrow{1.3} \ell_0 \xrightarrow{c} \ell_1 \xrightarrow{u} \ell_3 \xrightarrow{0.7} \ell_3 \xrightarrow{c} \]

\[
\begin{array}{c|c|c|c|c|c|c|c}
 & \ell_0 & \ell_0 & \ell_1 & \ell_2 & \ell_3 & \ell_3 & \\
 x & 0 & 1.3 & 1.3 & 1.3 & 1.3 & 2 & \\
y & 0 & 1.3 & 0 & 0 & 0 & 0.7 & \\
\end{array}
\]

cost : 6.5

That can be generalized!
Weighted/priced timed automata

\[ \ell_0 \xrightarrow{+5} \ell_1 \xrightarrow{(y=0)} \ell_2 \xrightarrow{u} \ell_3 \xrightarrow{(x=2,c)} \ell_4 \]

Cost:
\[
\begin{array}{cccc}
\ell_0 & \ell_1 & \ell_2 & \ell_3 \\
\begin{array}{cc}
x & 0 \\
y & 0 \\
\end{array} & \begin{array}{cc}
1.3 & 2 \\
1.3 & 0.7 \\
\end{array} & \begin{array}{cc}
1.3 & 0 \\
0 & 0 \\
\end{array} & \begin{array}{cc}
1.3 & 0 \\
0 & 0 \\
\end{array}
\end{array}
\]

\[
\text{cost: } 6.5 + 0 = 6.5
\]

That can be generalized!
Weighted/priced timed automata

- \[ \ell_0 \xrightarrow{\, \, x \leq 2, c, y:=0 \,} \ell_1 \]
- \[ \ell_1 \xrightarrow{u} \ell_2 \]
- \[ \ell_2 \xrightarrow{\, \, x=2, c \,} \ell_3 \]
- \[ \ell_3 \xrightarrow{\, \, x=2, c \,} \]

Transition costs:

<table>
<thead>
<tr>
<th></th>
<th>\ell_0</th>
<th>\ell_0</th>
<th>\ell_1</th>
<th>\ell_2</th>
<th>\ell_3</th>
<th>\ell_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>0</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>( y )</td>
<td>0</td>
<td>1.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Cost:

\[ \text{cost : } 6.5 + 0 + 0 = 6.5 \]
Weighted/priced timed automata

\[
\begin{align*}
\ell_0 & \xrightarrow{1.3} \ell_0 & \ell_0 & \xrightarrow{c} \ell_1 & \ell_1 & \xrightarrow{u} \ell_3 & \ell_3 & \xrightarrow{0.7} \ell_3 & \ell_3 & \xrightarrow{c} \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 0.7 & 0.7 & & & \\
y & 0 & 1.3 & 0 & 0 & 0 & 0.7 & & & & \\
\text{cost :} & 6.5 & + & 0 & + & 0 & + & 0.7 & & &
\end{align*}
\]

That can be generalized!
Weighted/priced timed automata

\[ \ell_0 \xrightarrow{1.3} \ell_0 \xrightarrow{c} \ell_1 \xrightarrow{u} \ell_3 \xrightarrow{0.7} \ell_3 \xrightarrow{c} \]

\[
\begin{array}{cccccccc}
\ell_0 & \ell_0 & \ell_1 & \ell_3 & \ell_3 & \ell_3 & \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & \\
y & 0 & 1.3 & 0 & 0 & 0.7 & \\
\end{array}
\]

\[
\text{cost : } 6.5 + 0 + 0 + 0.7 + 7
\]

That can be generalized!
Weighted/priced timed automata

That can be generalized!

Cost: $6.5 + 0 + 0 + 0.7 + 7 = 14.2$
Weighted/priced timed automata

\[ \ell_0 \xrightarrow{+5} \ell_1 \xrightarrow{(y=0)} \ell_2 \xrightarrow{+10} \ell_3 \xrightarrow{+1} \] with edges labeled by $x \leq 2, c, y := 0$ for transitioning from $\ell_0$ to $\ell_1$.

**Question:** what is the optimal cost for reaching \( \smiley \)?

\[ \inf_{0 \leq t \leq 2} \min (5t + 10(2-t) + 1, 5t + (2-t) + 7) = 9 \]

That can be generalized!
Weighted/priced timed automata

Question: what is the optimal cost for reaching 😊?

$$5t + 10(2 - t) + 1$$
Weighted/priced timed automata

Question: what is the optimal cost for reaching 😊?

\[ 5t + 10(2 - t) + 1, \quad 5t + (2 - t) + 7 \]
Weighted/priced timed automata

Question: what is the optimal cost for reaching 😊?

\[
\min ( 5t + 10(2 - t) + 1 , 5t + (2 - t) + 7 )
\]
Weighted/priced timed automata

Question: what is the optimal cost for reaching 😊?

\[
\inf_{0 \leq t \leq 2} \min \left( 5t + 10(2 - t) + 1, \ 5t + (2 - t) + 7 \right) = 9
\]
Weighted/priced timed automata

\[ \ell_0 \xrightarrow{x \leq 2, c, y:=0} \ell_1 \xrightarrow{y=0} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{x=2, c} \ell_3 \xrightarrow{c+1} \ell_3 \xrightarrow{x=2, c} \ell_3 \xrightarrow{+1} \ell_3 \xrightarrow{+10} \text{smiley face} \]

**Question:** what is the optimal cost for reaching \( \text{smiley face} \)?

\[
\inf_{0 \leq t \leq 2} \min \left( 5t + 10(2 - t) + 1, \ 5t + (2 - t) + 7 \right) = 9
\]

\( \sim \) **strategy:** leave immediately \( \ell_0 \), go to \( \ell_3 \), and wait there 2 t.u.
Weighted/priced timed automata

Question: what is the optimal cost for reaching \( \bigcirc \)?

\[
\inf_{0 \leq t \leq 2} \min \left( 5t + 10(2 - t) + 1, \ 5t + (2 - t) + 7 \right) = 9
\]

\( \leadsto \) strategy: leave immediately \( \ell_0 \), go to \( \ell_3 \), and wait there 2 t.u.

That can be generalized!

A simple timed game

\[ \ell_0 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \xrightarrow{(y = 0)} \ell_2 \xrightarrow{x = 2, c} \ell_3 \xrightarrow{x = 2, c} \text{happy face} \]

Question: what is the optimal cost we can ensure while reaching \( \ell_3 \)?

\[
\inf_{0 \leq t \leq 2} \max (5t + 10(2-t) + 1, 5t + (2-t) + 7) = 14 + \frac{1}{3};
\]

Strategy: wait in \( \ell_0 \), and when \( t = \frac{4}{3} \), go to \( \ell_1 \).
A simple weighted timed game

\[
\ell_0 + 5 \\
\xleq 2, c, y : = 0 \\
\ell_1 + 10 \\
(y = 0) \\
\ell_2 \\
\ell_3 + 1 \\
(x = 2, c) \\
\ell_2 + 1 \\
\]
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

\[5t + 10(2 - t) + 1\]
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching ☑️?

\[ 5t + 10(2 - t) + 1, \ 5t + (2 - t) + 7 \]
A simple weighted timed game

Question: what is the optimal cost we can ensure while reaching 😊?

\[
\max ( 5t + 10(2 - t) + 1 , 5t + (2 - t) + 7 )
\]
A simple weighted timed game

\[ \inf_{0 \leq t \leq 2} \max \left( 5t + 10(2 - t) + 1, 5t + (2 - t) + 7 \right) = 14 + \frac{1}{3} \]

**Question:** what is the optimal cost we can ensure while reaching 😊?
A simple weighted timed game

**Question:** what is the optimal cost we can ensure while reaching \( \square \)?

\[
\inf_{0 \leq t \leq 2} \max \left( 5t + 10(2 - t) + 1, \ 5t + (2 - t) + 7 \right) = 14 + \frac{1}{3}
\]

\( \sim \) **strategy:** wait in \( \ell_0 \), and when \( t = \frac{4}{3} \), go to \( \ell_1 \)
Optimal reachability in weighted timed games (1)

This topic has been fairly hot these last fifteen years...

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11, HIM13, BGK+14]
Optimal reachability in weighted timed games (1)

This topic has been fairly hot these last fifteen years...

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11, HIM13, BGK+14]

[LMM02]
Tree-like weighted timed games can be solved in 2EXPTIME.
This topic has been fairly hot these last fifteen years...

[LMM02, ABM04, BCFL04, BBR05, BBM06, BLMR06, Rut11, HIM13, BGK+14]

[LMM02]
Tree-like weighted timed games can be solved in 2EXPTIME.

[ABM04, BCFL04]
Depth-\(k\) weighted timed games can be solved in EXPTIME. There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.
In weighted timed games, the optimal cost cannot be computed, as soon as games have three clocks or more.
In weighted timed games, the optimal cost cannot be computed, as soon as games have three clocks or more.

Turn-based optimal timed games are decidable in EXPTIME (resp. PTIME) when automata have a single clock (resp. with two rates). They are PTIME-hard.
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$. 
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

The cost is increased by $x_0$.

The cost is increased by $1 - x_0$. 

Add$^+$($x$)

Add$^-$($x$)
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$. 

In $x_0$, cost = $2x_0 + (1 - y_0) + 2$ if $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$ if $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$ if $y_0 = 2x_0$, in both branches, cost = 3; player 2 can enforce cost 3 + $|y_0 - 2x_0|$.

Player 1 has a winning strategy with cost $\leq 3$ iff $y_0 = 2x_0$. 
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

In $\mathcal{S}$, cost = $2x_0 + (1 - y_0) + 2$
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

\[
\begin{align*}
\text{In } \mathcal{G}_1, \text{ cost } &= 2x_0 + (1 - y_0) + 2 \\
\text{In } \mathcal{G}_2, \text{ cost } &= 2(1 - x_0) + y_0 + 1
\end{align*}
\]
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- In $\smiley$, cost $= 2x_0 + (1 - y_0) + 2$
- In $\frowny$, cost $= 2(1 - x_0) + y_0 + 1$
- if $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- In the green branch, cost = $2x_0 + (1 - y_0) + 2$
- In the pink branch, cost = $2(1 - x_0) + y_0 + 1$

- If $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
- If $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

\[ \text{In }\begin{array}{l}
\text{methods } \text{green smiley, cost } = 2x_0 + (1 - y_0) + 2
\end{array}\]
\[ \text{In }\begin{array}{l}
\text{methods } \text{pink smiley, cost } = 2(1 - x_0) + y_0 + 1
\end{array}\]

- if $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
  - if $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$
  - if $y_0 = 2x_0$, in both branches, cost $= 3$
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

**In**, cost $= 2x_0 + (1 - y_0) + 2$

**In**, cost $= 2(1 - x_0) + y_0 + 1$

- If $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
- If $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$
- If $y_0 = 2x_0$, in both branches, cost $= 3$

\[ \text{∽ player 2 can enforce cost } 3 + |y_0 - 2x_0| \]
Computing the optimal cost: why is that hard?

Given two clocks $x$ and $y$, we can check whether $y = 2x$.

- In $\smiley$, cost = $2x_0 + (1 - y_0) + 2$
- In $sad$, cost = $2(1 - x_0) + y_0 + 1$

- if $y_0 < 2x_0$, player 2 chooses the first branch: cost $> 3$
- if $y_0 > 2x_0$, player 2 chooses the second branch: cost $> 3$
- if $y_0 = 2x_0$, in both branches, cost = 3
  $\Rightarrow$ player 2 can enforce cost $3 + |y_0 - 2x_0|$

- Player 1 has a winning strategy with cost $\leq 3$ iff $y_0 = 2x_0$
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:

- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

\[
x = \frac{1}{2^{c_1}} \quad \text{and} \quad y = \frac{1}{3^{c_2}}
\]
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:
  \[ x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2} \]

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:

- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

$$ x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2} $$

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.

Globally, $(x \leq 1, y \leq 1, u \leq 1)$

$x=1, x:=0$

$\vee \quad y=1, y:=0$

$u:=0 \quad z:=0 \quad u=1, u:=0$
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

\[ x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2} \]

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:

- each instruction is encoded as a module;
- the counter values \( c_1 \) and \( c_2 \) are encoded by two clocks:

\[
x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2}
\]

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

$$x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2}$$

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values \( c_1 \) and \( c_2 \) are encoded by two clocks:

\[
x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2}
\]

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Computing the optimal cost: why is that hard?

Player 1 will simulate a two-counter machine:
- each instruction is encoded as a module;
- the counter values $c_1$ and $c_2$ are encoded by two clocks:

$$x = \frac{1}{2c_1} \quad \text{and} \quad y = \frac{1}{3c_2}$$

The two-counter machine has a halting computation iff player 1 has a winning strategy to ensure a cost no more than 3.
Shape of the reduction
Are we done?
Are we done? No! Let’s be a bit more precise!

Given a weighted timed game, a strategy $\sigma$ is winning whenever all its outcomes are winning; Cost of a winning strategy $\sigma$:

$$\text{cost}(\sigma) = \sup \left\{ \text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target} \right\}$$

Optimal cost:

$$\text{optcost}_G = \inf_{\sigma \text{ winning strat.}} \text{cost}(\sigma)$$
(set it to $+\infty$ if there is no winning strategy)

Two problems of interest

The value problem asks, given $G$ and a threshold $\Delta$, whether $\text{optcost}_G \leq \Delta$?

The existence problem asks, given $G$ and a threshold $\Delta$, whether there exists a winning strategy in $G$ such that $\text{cost}(\sigma) \leq \Delta$?

Note: These problems are distinct...
Are we done? No! Let’s be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy $\sigma$ is winning whenever all its outcomes are winning;
Are we done? No! Let’s be a bit more precise!

Given $\mathcal{G}$ a weighted timed game,

- a strategy $\sigma$ is winning whenever all its outcomes are winning;
- Cost of a winning strategy $\sigma$:

$$\text{cost}(\sigma) = \sup\{\text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target}\}$$
Are we done? No! Let’s be a bit more precise!

Given \( \mathcal{G} \) a weighted timed game,

- a strategy \( \sigma \) is winning whenever all its outcomes are winning;
- **Cost of a winning strategy** \( \sigma \):

  \[
  \text{cost}(\sigma) = \sup \{ \text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target} \}
  \]

- **Optimal cost:**

  \[
  \text{optcost}_\mathcal{G} = \inf_{\sigma \text{ winning strat.}} \text{cost}(\sigma)
  \]

  (set it to \(+\infty\) if there is no winning strategy)
Are we done? No! Let’s be a bit more precise!

Given $G$ a weighted timed game,
- a strategy $\sigma$ is winning whenever all its outcomes are winning;
- Cost of a winning strategy $\sigma$:
  \[
  \text{cost}(\sigma) = \sup\{\text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target}\}
  \]
- Optimal cost:
  \[
  \text{optcost}_G = \inf_{\sigma \text{ winning strat.}} \text{cost}(\sigma)
  \]
  (set it to $+\infty$ if there is no winning strategy)

Two problems of interest
- The value problem asks, given $G$ and a threshold $\bowtie c$, whether $\text{optcost}_G \bowtie c$?
Are we done? No! Let’s be a bit more precise!

Given $G$ a weighted timed game,
- a strategy $\sigma$ is winning whenever all its outcomes are winning;
- Cost of a winning strategy $\sigma$:

$$\text{cost}(\sigma) = \sup \{\text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target} \}$$

- Optimal cost:

$$\text{optcost}_G = \inf_{\sigma \text{ winning strat.}} \text{cost}(\sigma)$$

(set it to $+\infty$ if there is no winning strategy)

Two problems of interest

- The value problem asks, given $G$ and a threshold $\triangleright c$, whether $\text{optcost}_G \triangleright c$?
- The existence problem asks, given $G$ and a threshold $\triangleright c$, whether there exists a winning strategy in $G$ such that $\text{cost}(\sigma) \triangleright c$?
Are we done? No! Let’s be a bit more precise!

Given \( G \) a weighted timed game,
- a strategy \( \sigma \) is winning whenever all its outcomes are winning;
- Cost of a winning strategy \( \sigma \):
  \[
  \text{cost}(\sigma) = \sup \{ \text{cost}(\rho) \mid \rho \text{ outcome of } \sigma \text{ up to the target} \}
  \]
- Optimal cost:
  \[
  \text{optcost}_G = \inf_{\sigma \text{ winning strat.}} \text{cost}(\sigma)
  \]
  (set it to \(+\infty\) if there is no winning strategy)

Two problems of interest

- The value problem asks, given \( G \) and a threshold \( \triangleright c \), whether \( \text{optcost}_G \triangleright c \)?
- The existence problem asks, given \( G \) and a threshold \( \triangleright c \), whether there exists a winning strategy in \( G \) such that \( \text{cost}(\sigma) \triangleright c \)?

*Note:* These problems are distinct...
The value of the game is 3, but no strategy has cost 3.
The value of the game is 3, but no strategy has cost 3.
The value of the game is 3, but no strategy has cost 3.
Weighted timed automata

In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.
Weighted timed automata

In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE. The value problem is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.
Weighted timed automata

In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.
The value problem is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

Weighted timed games

Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.
- Weighted timed automata

In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE. The value problem is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

- Weighted timed games

Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock. The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.
Weighted timed automata

In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.
The value problem is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

Weighted timed games

Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.
The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.
- **Weighted timed automata**

  In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.
  The **value problem** is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

- **Weighted timed games**

  Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.
  The **value problem** is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

  There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.
  The **value problem** can be decided in EXPTIME in weighted timed games with a strongly non-Zeno cost. Almost-optimal winning strategies can be computed.
• Weighted timed automata

In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE. The value problem is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

• Weighted timed games

Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock. The value problem is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost. The value problem can be decided in EXPTIME in weighted timed games with a strongly non-Zeno cost. Almost-optimal winning strategies can be computed.

In weighted timed games, the optimal cost cannot be computed, as soon as games have three clocks or more.
- **Weighted timed automata**

  In weighted timed automata, the optimal cost is an integer, and can be computed in PSPACE.
  The *value problem* is PSPACE-complete in weighted timed automata. Almost-optimal winning schedules can be computed.

- **Weighted timed games**

  Turn-based optimal timed games are decidable in EXPTIME when automata have a single clock.
  The *value problem* is decidable in EXPTIME in single-clock weighted timed games. Almost-optimal memoryless winning strategies can be computed.

  There is a symbolic algorithm to solve weighted timed games with a strongly non-Zeno cost.
  The *value problem* can be decided in EXPTIME in weighted timed games with a strongly non-Zeno cost. Almost-optimal winning winning strategies can be computed.

  In weighted timed games, the optimal cost cannot be computed, as soon as games have three clocks or more.
  The *existence problem* is undecidable in weighted timed games.
Our recent developments

The value problem is undecidable in weighted timed games

- Intellectually satisfactory to not have this discrepancy in the set of results
- Proof based on a diagonal construction (originally proposed in the context of quantitative temporal logics [BMM14])
Our recent developments

1. The value problem is undecidable in weighted timed games
   - Intellectually satisfactory to not have this discrepancy in the set of results
   - Proof based on a diagonal construction (originally proposed in the context of quantitative temporal logics [BMM14])

2. An approximation algorithm for a large class of weighted timed games (that comprises the class of games used for proving the above undecidability)
   - Almost-optimality in practice should be sufficient
   - Even when we know how to compute the value, we are only able to synthesize almost-optimal strategies...

Optimal cost is computable...

... when cost is strongly non-zeno. \[\text{[AM04, BCFL04]}\]

That is, there exists $\kappa > 0$ such that for every region cycle $C$, for every real run $\varrho$ read on $C$,

$$\text{cost}(\varrho) \geq \kappa$$

Optimal cost is not computable...

... when cost is almost-strongly non-zeno.

That is, there exists $\kappa > 0$ such that for every region cycle $C$, for every real run $\varrho$ read on $C$,

$$\text{cost}(\varrho) \geq \kappa \text{ or } \text{cost}(\varrho) = 0$$

Note: In both cases, we can assume $\kappa = 1$. 
Optimal cost is computable...  
... when cost is strongly non-zeno. \[\text{[AM04, BCFL04]}\]

That is, there exists $\kappa > 0$ such that for every region cycle $C$, for every real run $\varrho$ read on $C$,
\[
\text{cost}(\varrho) \geq \kappa
\]

Optimal cost is not computable... but is approximable!  
... when cost is almost-strongly non-zeno. \[\text{[BJM15]}\]

That is, there exists $\kappa > 0$ such that for every region cycle $C$, for every real run $\varrho$ read on $C$,
\[
\text{cost}(\varrho) \geq \kappa \quad \text{or} \quad \text{cost}(\varrho) = 0
\]

*Note:* In both cases, we can assume $\kappa = 1$. 

Approximation of the optimal cost

**Theorem**

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v_\epsilon^-$ and $v_\epsilon^+$ such that

\[ |v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \text{optcost}_\mathcal{G} \leq v_\epsilon^+ \]
Approximation of the optimal cost

**Theorem**

Let $\mathcal{G}$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v_\epsilon^-$ and $v_\epsilon^+$ such that
  \[ |v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \text{optcost}_G \leq v_\epsilon^+ \]
- one strategy $\sigma_\epsilon$ such that
  \[ \text{optcost}_G \leq \text{cost}(\sigma_\epsilon) \leq \text{optcost}_G + \epsilon \]

It is an $\epsilon$-optimal winning strategy.
Approximation of the optimal cost

**Theorem**

Let $G$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v_\epsilon^-$ and $v_\epsilon^+$ such that
  
  $$|v_\epsilon^+ - v_\epsilon^-| < \epsilon$$
  
  and
  
  $$v_\epsilon^- \leq \text{optcost}_G \leq v_\epsilon^+$$

- one strategy $\sigma_\epsilon$ such that
  
  $$\text{optcost}_G \leq \text{cost}(\sigma_\epsilon) \leq \text{optcost}_G + \epsilon$$

It is an $\epsilon$-optimal winning strategy.

- Standard technics: unfold the game to get more precision, and compute two adjacency sequences
Approximation of the optimal cost

**Theorem**

Let $G$ be a weighted timed game, in which the cost is almost-strongly non-zeno. For every $\epsilon > 0$, one can compute:

- two values $v_\epsilon^-$ and $v_\epsilon^+$ such that
  
  \[ |v_\epsilon^+ - v_\epsilon^-| < \epsilon \quad \text{and} \quad v_\epsilon^- \leq \text{optcost}_G \leq v_\epsilon^+ \]

- one strategy $\sigma_\epsilon$ such that
  
  \[ \text{optcost}_G \leq \text{cost}(\sigma_\epsilon) \leq \text{optcost}_G + \epsilon \]

It is an $\epsilon$-optimal winning strategy.

- Standard technics: unfold the game to get more precision, and compute two adjacency sequences

  ~ This is not possible here

  There might be runs with prefixes of arbitrary length and cost 0 (e.g. the game of the undecidability proof)
Idea for approximation

Idea

Only partially unfold the game:

- Keep components with cost 0 untouched – we call it the **kernel**
- Unfold the rest of the game
Semi-unfolding

Hypothesis: cost $> 0$ implies cost $\geq \kappa$

Conclusion: we can stop unfolding the game after $N$ steps (e.g. $N = (M + 2) \cdot |R(A)|$, where $M$ is a pre-computed bound on optcost $G$)
Semi-unfolding

Hypothesis: cost > 0 implies cost ≥ κ

Conclusion: we can stop unfolding the game after N steps (e.g. N = (M + 2) · |R(A)|, where M is a pre-computed bound on optcost(G))
Semi-unfolding

Hypothesis: cost > 0 implies cost ≥ κ

Conclusion: we can stop unfolding the game after N steps (e.g., N = (M + 2) · |R(A)|, where M is a pre-computed bound on optcost G).

Only cost 0 Kernel $\mathcal{K}$

(\ell, r)
Semi-unfolding

Only cost $0$ Kernel $\mathcal{K}$

Hypothesis: cost $> 0$ implies cost $\geq \kappa$

Conclusion: we can stop unfolding the game after $N$ steps (e.g. $N = (M + 2) \cdot |R(A)|$, where $M$ is a pre-computed bound on $\text{optcost}$ $G$)
Semi-unfolding

Hypothesis: cost > 0 implies cost ≥ κ

Conclusion: we can stop unfolding the game after N steps
(e.g. N = (M + 2) · |R(A)|, where M is a pre-computed bound on optcost_\text{G})
Approximation scheme
Approximation scheme
Approximation scheme
Approximation scheme

Exact computation
Approximation scheme

Exact computation

Approximation
First step: Tree-like parts

\[ \ell, \ell' \]

\[ Y' \leftarrow 0 \] \[ v' = \left( Y' \leftarrow 0 \right) \]

\[ Y'' \leftarrow 0 \] \[ v'' = \left( Y'' \leftarrow 0 \right) \]

\[ \sim \text{ Goes back to } [LMM02] \]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ \text{ Goes back to [LMM02]} \]

\[ \ell, \ell' \]

\[ g', Y' \]

\[ c' \]

\[ c'' \]

\[ g'', Y'' \]

\[ \ell'' \]
First step: Tree-like parts

\[ O(\ell, v) = \]

\[ O(\ell', v') \]
\[ O(\ell'', v'') \]

\[ \leadsto \text{Goes back to [LMM02]} \]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' \mid v + t' = g} \]

~ Goes back to [LMM02]
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' | v + t' = g'} \max( , ) \]

\[ O(\ell', v') \]

\[ O(\ell'', v'') \]

\[ c', g', Y' \]

\[ c'', g'', Y'' \]

\[ \sim \text{ Goes back to [LMM02]} \]

[LMM02] La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).
First step: Tree-like parts

\[ O(\ell, \nu) = \inf_{t' \mid \nu + t' = g'} \max((\alpha), ) \]

\[ (\alpha) = t'c + c' + O(\ell', \nu') \]

\[ \nu' = [Y' \leftarrow 0](\nu + t') \]
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' | v + t' \models g'} \max(\alpha, \beta) \]

\[ (\alpha) = t' c + c' + O(\ell', v') \]

\[ (\beta) = \sup_{t'' \leq t' | v + t'' \models g''} \quad t'' c + c'' + O(\ell'', v'') \]

\[ t' = Y' \leftarrow 0 (v + t') \]

\[ t'' = Y'' \leftarrow 0 (v + t'') \]

\[ O(\ell, v) \Rightarrow \text{Goes back to [LMM02]} \]

[La Torre, Mukhopadhyay, Murano. Optimal-reachability and control for acyclic weighted timed automata (TCS@02).]
Second step: Kernels

Output cost functions $f$
Second step: Kernels

Output cost functions $f$

1. Refine the regions such that $f$ differs of at most $\epsilon$ within a small region
Second step: Kernels

1. Refine the regions such that $f$ differs of at most $\epsilon$ within a small region

Output cost functions $f$
Second step: Kernels

Refine the regions such that \( f \) differs of at most \( \epsilon \) within a small region.

Output cost functions \( f \)
Second step: Kernels

1. Refine the regions such that $f$ differs of at most $\epsilon$ within a small region.

2. Under- and over-approximate by piecewise constant functions $f_{\epsilon}^-$ and $f_{\epsilon}^+$.
Second step: Kernels

Refine/split the kernel along the new small regions and fix $f_\epsilon^-$ or $f_\epsilon^+$, write $f_\epsilon$

$f_\epsilon$: constant  $f_\epsilon$: constant
Second step: Kernels

- Refine/split the kernel along the new small regions and fix $f_\epsilon^-$ or $f_\epsilon^+$, write $f_\epsilon$
- Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by $f_\epsilon$)

$f_\epsilon$: constant  $f_\epsilon$: constant
Second step: Kernels

- Refine/split the kernel along the new small regions and fix $f_\epsilon^-$ or $f_\epsilon^+$, write $f_\epsilon$
- Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by $f_\epsilon$)
- Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output $f_\epsilon$) is constant within a small region

$f_\epsilon$: constant  $f_\epsilon$: constant
Second step: Kernels

Refine/split the kernel along the new small regions and fix \( f_\epsilon^- \) or \( f_\epsilon^+ \), write \( f_\epsilon \)

Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by \( f_\epsilon \))

Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output \( f_\epsilon \)) is constant within a small region.

\[ f_\epsilon : \text{constant} \quad f_\epsilon : \text{constant} \]
Second step: Kernels

4. Refine/split the kernel along the new small regions and fix $f^-_\epsilon$ or $f^+_\epsilon$, write $f_\epsilon$

4. Since cost is 0 everywhere, the resulting game is nothing more than a reachability timed game with an order on target (output) edges (given by $f_\epsilon$)

5. Those can be solved using standard technics based on attractors: small regions are sufficient, and the local optimal cost (for output $f_\epsilon$) is constant within a small region

7. We have computed $\epsilon$-approximations of the optimal cost, which are constant within small regions. Corresponding strategies can be inferred
Conclusion

Summary of the talk

- Very quick overview of results concerning the optimal reachability problem in weighted timed games
- Some new insight into the value problem for this model:
  - Undecidability of this problem
  - Approximability of the optimal cost (under some conditions)

Future work

- Improve the approximation scheme $2^{\text{EXP}(|G|)} \cdot \left(\frac{1}{\epsilon} \right)^{|X|}$
- Extend to the whole class of weighted timed games, or understand why it is not possible
- Assume stochastic uncertainty?
- Multiplayer setting?
Conclusion

Summary of the talk

- Very quick overview of results concerning the optimal reachability problem in weighted timed games
- Some new insight into the value problem for this model:
  - Undecidability of this problem
  - Approximability of the optimal cost (under some conditions)

Future work

- Improve the approximation scheme \((2\text{EXP}(|G|) \cdot \left(1/\epsilon\right)^{|X|})\)
- Extend to the whole class of weighted timed games, or understand why it is not possible
- Assume stochastic uncertainty?
- Multiplayer setting?