On the optimal reachability problem in weighted timed games

Patricia Bouyer-Decitre

LSV, CNRS & ENS Cachan, France

Based on former works with Thomas Brihaye, Kim G. Larsen, Nicolas Markey, etc…
And on recent work with Samy Jaziri and Nicolas Markey
Outline

1. Introduction

2. Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3. Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4. Conclusion
Time-dependent systems

- We are interested in timed systems
Time-dependent systems

- We are interested in timed systems
Time-dependent systems

- We are interested in timed systems

- ... and in their analysis and control
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>Operation</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>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>

$P_2$ (slow):

<table>
<thead>
<tr>
<th>Operation</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>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>

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):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>+ 2 ps</td>
<td>$\times$ 3 ps</td>
</tr>
<tr>
<td>energy</td>
<td>idle</td>
<td>10 Watt</td>
</tr>
<tr>
<td></td>
<td>in use</td>
<td>90 Watts</td>
</tr>
</tbody>
</table>

$P_2$ (slow):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>+ 5 ps</td>
<td>$\times$ 7 ps</td>
</tr>
<tr>
<td>energy</td>
<td>idle</td>
<td>20 Watts</td>
</tr>
<tr>
<td></td>
<td>in use</td>
<td>30 Watts</td>
</tr>
</tbody>
</table>

Sch1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$T_2$</td>
<td>$T_3$</td>
<td>$T_5$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$T_1$</td>
<td>$T_4$</td>
<td>$T_6$</td>
</tr>
</tbody>
</table>

13 picoseconds 1.37 nanojoules

Sch2

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$T_2$</td>
<td>$T_3$</td>
<td>$T_4$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$T_5$</td>
<td>$T_6$</td>
<td>$T_3$</td>
</tr>
</tbody>
</table>

12 picoseconds 1.39 nanojoules

Sch3

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$T_2$</td>
<td>$T_3$</td>
<td>$T_4$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$T_5$</td>
<td>$T_6$</td>
<td>$T_3$</td>
</tr>
</tbody>
</table>

19 picoseconds 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:

**$P_1$ (fast):**

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th></th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>2 picoseconds</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>×</td>
<td>3 picoseconds</td>
<td>in use</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th></th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>5 picoseconds</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>×</td>
<td>7 picoseconds</td>
<td>in use</td>
</tr>
</tbody>
</table>

**$P_2$ (slow):**

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

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th></th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>2 picoseconds</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>×</td>
<td>3 picoseconds</td>
<td>in use</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th></th>
<th>energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>5 picoseconds</td>
<td>idle</td>
</tr>
<tr>
<td></td>
<td>×</td>
<td>7 picoseconds</td>
<td>in use</td>
</tr>
</tbody>
</table>

An example: The task graph scheduling problem

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

\[ P_1 \text{ (fast)}: \]

\[
\begin{array}{|c|c|}
\hline
\text{time} & \text{3 picoseconds} \\
\hline
\text{+} & 2 \text{ picoseconds} \\
\hline
\times & 2 \text{ picoseconds} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{energy} & \text{} \\
\hline
\text{idle} & 10 \text{ Watt} \\
\hline
\text{in use} & 90 \text{ Watts} \\
\hline
\end{array}
\]

\[ P_2 \text{ (slow)}: \]

\[
\begin{array}{|c|c|}
\hline
\text{time} & \text{7 picoseconds} \\
\hline
\text{+} & 5 \text{ picoseconds} \\
\hline
\times & 7 \text{ picoseconds} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{energy} & \text{} \\
\hline
\text{idle} & 20 \text{ Watts} \\
\hline
\text{in use} & 30 \text{ Watts} \\
\hline
\end{array}
\]
The model of timed automata
The model of timed automata

\[ x := 0 \leq 15 \]
\[ y := 0 \leq 25 \]
\[ x := 2 \leq y \leq 56 \]

\[ done, 22 \leq y \leq 25 \]
\[ repair, 2 \leq y \leq 25 \]
\[ delayed, y := 0 \]

\[ safe \rightarrow alarm \rightarrow repairing \rightarrow failsafe \]

<table>
<thead>
<tr>
<th>State</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>problem</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>alarm</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>alarm</td>
<td>15.6</td>
<td>38.6</td>
</tr>
<tr>
<td>repair</td>
<td>15.6</td>
<td>0</td>
</tr>
<tr>
<td>repairing</td>
<td>22.1</td>
<td>40</td>
</tr>
<tr>
<td>safe</td>
<td>40</td>
<td>22.1</td>
</tr>
</tbody>
</table>

\[ \cdots \]
Modelling the task graph scheduling problem
Modelling the task graph scheduling problem

**Processors**

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

\[ P_2: \]
\[ (y \leq 5) \]
\[ \begin{array}{c}
\text{idle} \\
y = 5 \\
\text{add}_2 \\
\text{done}_2 \\
x := 0
\end{array} \]
\[ \quad \Rightarrow \quad \begin{array}{c}
\text{idle} \\
y = 7 \\
\text{mult}_2 \\
\text{done}_2 \\
x := 0
\end{array} \]
\[ (y \leq 7) \]
Modelling the task graph scheduling problem

- **Processors**
  - $P_1$: $x = 2$ (idle), $x = 3$ (idle) with $x = 0$
  - $P_2$: $y = 5$ (idle), $y = 7$ (idle) with $x = 0$

- **Tasks**
  - $T_4$: $t_1 \land t_2$, $t_4 := 1$
  - $T_5$: $t_3$, $t_5 := 1$

A schedule is a path in the product automaton
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction
Efficient symbolic technics based on zones, implemented in tools...
Analyzing timed automata

Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction
Efficient symbolic technics based on zones, implemented in tools...
Analyzing timed automata

Theorem [AD94] Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools

Skip regions
Analyzing timed automata

**Theorem [AD94]**

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools...
Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools...
Analyzing timed automata

Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction
Efficient symbolic technics based on zones, implemented in tools Skip regions
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction
Efficient symbolic technics based on zones, implemented in tools...
Analyzing timed automata

Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

Theorem [AD94]

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction

Efficient symbolic technics based on zones, implemented in tools...
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

Technical tool: region abstraction
Efficient symbolic technics based on zones, implemented in tools.
Analyzing timed automata

**Theorem [AD94]**

Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction
- Efficient symbolic technics based on zones, implemented in tools
Analyzing timed automata

Theorem [AD94]
Reachability in timed automata is decidable (as well as many other important properties). It is PSPACE-complete.

- Technical tool: region abstraction
- Efficient symbolic technics based on zones, implemented in tools
Technical tool: Region abstraction

![Diagram showing a region abstraction grid with clock x and clock y axes, marked with points at (0,0), (1,1), (2,2), (0,1), (1,2), and (2,0). The diagram illustrates regions of state space with constraints on clocks x and y.]

- only constraints: $x \sim c$ with $c \in \{0, 1, 2\}$
- $y \sim c$ with $c \in \{0, 1, 2\}$
- The path $x = 1$, $y = 1$ can be fired from
- cannot be fired from
- This is a finite time-abstract bisimulation!
Technical tool: Region abstraction

only constraints: $x \sim c$ with $c \in \{0, 1, 2\}$
$y \sim c$ with $c \in \{0, 1, 2\}$

“compatibility” between regions and constraints
Technical tool: Region abstraction

The path $x=1$, $y=1$ can be fired from
- cannot be fired from

"compatibility" between regions and constraints
"compatibility" between regions and time elapsing
Technical tool: Region abstraction

This is a finite time-abstract bisimulation!
Technical tool: Region abstraction – An example [AD94]
Technical tool: Region abstraction – An example [AD94]
Technical tool: Region abstraction – An example [AD94]

\[
\begin{align*}
& s_0, x=y=0, \\
& s_1, 0=y<x<1, \\
& s_1, y=0, x=1, \\
& s_1, y=0, x>1, \\
& s_2, 1=y<x, \\
& s_3, 0<y<x<1, \\
& s_3, 0<y<1<x, \\
& s_3, 1=y<x, \\
& s_3, x>1, y>1
\end{align*}
\]
Outline

1 Introduction

2 Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3 Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4 Conclusion
Outline

1 Introduction

2 Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3 Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4 Conclusion
Modelling resources in timed systems

- System resources might be relevant and even crucial information
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  - price to pay,
  - bandwidth,
Modelling resources in timed systems

- System **resources** might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...

→ timed automata are not powerful enough!
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  $\Rightarrow$ timed automata are not powerful enough!

- A possible solution: use hybrid automata
  a discrete control (the mode of the system)
  $+$ continuous evolution of the variables within a mode
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...

\[ \dot{T} = -0.5T \quad (T \geq 18) \]
\[ \dot{T} = 2.25 - 0.5T \quad (T \leq 22) \]

\[ T \leq 19 \]
\[ T \geq 21 \]

\[ T \leq 19 \]
\[ T \geq 21 \]

\sim \text{ timed automata are not powerful enough!} \]

- A possible solution: use hybrid automata

The thermostat example
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...

→ timed automata are not powerful enough!

- A possible solution: use hybrid automata

The thermostat example

- Off: $\dot{T} = -0.5T$ for $T \geq 18$
- On: $\dot{T} = 2.25 - 0.5T$ for $T \leq 22$

Diagram showing temperature changes over time.
Ok...
Ok...

Easy...
Ok...

Easy...
Ok...
Ok... but?

Easy...

constraint

constraint

Easy...

constraint
Ok... but?

Easy...

Hard!
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...
  \[\rightarrow\] timed automata are not powerful enough!
- A possible solution: use hybrid automata

**Theorem** [HKPV95]
The reachability problem is undecidable in hybrid automata. Even for the simplest, the so-called stopwatch automata (clocks can be stopped).

[HKPV95] Henzinger, Kopke, Puri, Varaiya. What’s decidable w/bout hybrid automata? (*SToC’95*).
Modelling resources in timed systems

- System resources might be relevant and even crucial information
  - energy consumption,
  - memory usage,
  - ...

  \[\Rightarrow\] timed automata are not powerful enough!

- A possible solution: use hybrid automata

**Theorem** [HKPV95]

The reachability problem is **undecidable** in hybrid automata. Even for the simplest, the so-called stopwatch automata (clocks can be stopped).

- An alternative: **weighted/priced timed automata** [ALP01,BFH+01]

  \[\Rightarrow\] hybrid variables do not constrain the system
  hybrid variables are **observer** variables

[BFH+01] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata *(HSCC’01)*.
Modelling the task graph scheduling problem

**Processors**

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

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

**Tasks**

- **$T_4$:**
  - $t_1 \land t_2$
  - $t_4 := 1$
  - \( \text{add}_i \) \( \text{done}_i \)

- **$T_5$:**
  - $t_3$
  - $t_5 := 1$
  - \( \text{add}_i \) \( \text{done}_i \)
Modelling the task graph scheduling problem

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

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

- Tasks
  
  $T_4$: 
  
  \[ \begin{align*} 
  t_1 \land t_2 & \xrightarrow{\text{add}_i} \text{done}_i \\
  & \xrightarrow{t_4 := 1} \text{add}_i
  \end{align*} \]

  $T_5$: 
  
  \[ \begin{align*} 
  t_3 & \xrightarrow{\text{add}_i} \text{done}_i \\
  & \xrightarrow{t_5 := 1} \text{add}_i
  \end{align*} \]

- Modelling energy
  
  $P_1$: 
  
  \[ \begin{align*} 
  +90 & \xrightarrow{\text{add}_1} +10 \\
  (x \leq 2) & \xrightarrow{x := 0} +10 \\
  & \xrightarrow{x := 0} +90 \\
  & \xrightarrow{(x \leq 3)} +90
  \end{align*} \]

  $P_2$: 
  
  \[ \begin{align*} 
  +30 & \xrightarrow{\text{add}_2} +20 \\
  (y \leq 5) & \xrightarrow{x := 0} +20 \\
  & \xrightarrow{x := 0} +30 \\
  & \xrightarrow{(y \leq 7)} +30
  \end{align*} \]

A good schedule is a path in the product automaton with a low cost
Weighted/priced timed automata \([\text{ALP01,BFH+01}]\)

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


Weighted/priced timed automata [ALP01,BFH+01]

\[ \ell_0 + 5 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{x = 2, c} \ell_3 \xrightarrow{c} +1 \]

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

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC'01).

Weighted/priced timed automata \[\text{[ALP01,BFH+01]}\]

\[
\begin{array}{cccccc}
\ell_0 & \xrightarrow{1.3} & \ell_0 & \xrightarrow{c} & \ell_1 & \xrightarrow{u} & \ell_3 & \xrightarrow{0.7} & \ell_3 & \xrightarrow{c} & \text{smiley} \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 0 & 0.7 \\
y & 0 & 1.3 & 0 & 0 & & & \\
\end{array}
\]

cost :

\[6.5 + 0 + 0 + 0.7 = 14.2\]

\[\text{[ALP01]}\] Alur, La Torre, Pappas. Optimal paths in weighted timed automata \((HSCC'01)\).

\[\text{[BFH+01]}\] Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata \((HSCC'01)\).
Weighted/priced timed automata \[\text{[ALP01,BFH+01]}\]

\[
\begin{array}{ccccccc}
\ell_0 & \xrightarrow{1.3} & \ell_0 & \xrightarrow{c} & \ell_1 & \xrightarrow{u} & \ell_3 & \xrightarrow{0.7} & \ell_3 & \xrightarrow{c} & \text{\smiley} \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & 0 & 0 & 0.7 \\
y & 0 & 1.3 & 0 & 0 \\
\end{array}
\]

\text{cost} : 6.5

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC'01).
Weighted/priced timed automata [ALP01,BFH+01]

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

Weighted/priced timed automata [ALP01, BFH+01]

\[
\begin{align*}
\ell_0 & \xrightarrow{1.3} \ell_0 \\
\ell_0 & \xrightarrow{c} \ell_1 \\
\ell_1 & \xrightarrow{u} \ell_3 \\
\ell_3 & \xrightarrow{0.7} \ell_3 \\
\ell_3 & \xrightarrow{c} \smiley
\end{align*}
\]

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

cost : $6.5 + 0 + 0$
Weighted/priced timed automata \cite{ALP01,BFH+01}

\[ \ell_0 \xrightarrow{+5} \ell_1 \quad (y=0) \quad \ell_1 \xrightarrow{u} \ell_2 \quad x=2,c \quad \ell_2 \xrightarrow{+10} \ell_3 \quad x=2,c \quad \ell_3 \xrightarrow{+1} \]

\[
\begin{array}{c|c|c|c|c|c|c}
& \ell_0 & \ell_1 & \ell_2 & \ell_3 & \ell_3 & \ell_1 \\
x & 0 & 1.3 & 1.3 & 1.3 & 2 & \text{c} \\
y & 0 & 1.3 & 0 & 0 & 0.7 & \text{c} \\
\text{cost} & 6.5 & + & 0 & + & 0 & + & 0.7
\end{array}
\]
Weighted/priced timed automata \[\text{[ALP01,BFH+01]}\]

\[
\begin{align*}
\ell_0 & \xrightarrow{\text{x}\le2,\text{c},\text{y}:=}0 \ell_1 \\
\ell_1 & \xrightarrow{\text{y}:=}0 (\ell_0) \\
\ell_1 & \xrightarrow{\text{u}} \ell_2 \\
\ell_2 & \xrightarrow{\text{x}=:2,\text{c}} \ell_3 \\
\ell_3 & \xrightarrow{\text{c}} \ell_0 \\
\end{align*}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c}
\ell_0 & \ell_0 & \ell_1 & \ell_2 & \ell_3 & \ell_3 & \ell_0 \\
\hline
x & 0 & 1.3 & 1.3 & 1.3 & 2 & \\
y & 0 & 1.3 & 0 & 0 & 0.7 & \\
\hline
cost & 6.5 & + & 0 & + & 0 & + & 0.7 & + & 7
\end{array}
\]

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC'01).

Weighted/-priced timed automata [ALP01,BFH+01]

\[ \ell_0 \xrightarrow{+5} \ell_1 \xrightarrow{\leq 2, c, y:=0} \ell_2 \xrightarrow{u} \ell_3 \xrightarrow{x=2, c} +1 \xrightarrow{1} \]

<table>
<thead>
<tr>
<th>State</th>
<th>x</th>
<th>y</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ell_0</td>
<td>0</td>
<td>0</td>
<td>6.5</td>
</tr>
<tr>
<td>\ell_1</td>
<td>1.3</td>
<td>0</td>
<td>7.3</td>
</tr>
<tr>
<td>\ell_2</td>
<td>1.3</td>
<td>0</td>
<td>8.0</td>
</tr>
<tr>
<td>\ell_3</td>
<td>2</td>
<td>0.7</td>
<td>14.2</td>
</tr>
</tbody>
</table>

\[ \text{cost} : 6.5 + 0 + 0 + 0.7 + 7 = 14.2 \]

Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching ☺?
Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching 😊?

\[ 5t + 10(2 - t) + 1 \]
**Weighted/priced timed automata** [ALP01,BFH+01]

**Question:** what is the optimal cost for reaching 😊?

\[ 5t + 10(2 - t) + 1, \quad 5t + (2 - t) + 7 \]

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (*HSCC’01*).

Weighted/priced timed automata [ALP01,BFH+01]

Question: what is the optimal cost for reaching 😊?

$$\min ( 5t + 10(2 - t) + 1 , 5t + (2 - t) + 7 )$$

Weighted/priced timed automata [ALP01,BFH+01]

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

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

[ALP01] Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC'01).
Weighted/priced timed automata \cite{ALP01,BFH+01}

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

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

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

\cite{ALP01} Alur, La Torre, Pappas. Optimal paths in weighted timed automata (HSCC’01).
\cite{BFH+01} Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, Vaandrager. Minimum-cost reachability in priced timed automata (HSCC’01).
Optimal-cost reachability

Theorem [ALP01,BFH+01,BBBR07]
In weighted timed automata, the optimal cost is an integer and can be computed in PSPACE.

Technical tool: a refinement of the regions, the corner-point abstraction

Note on the corner-point abstraction

It is a very interesting abstraction, that can be used for many applications:

- for mean-cost optimization [BBL04,BBL08]
- for discounted-cost optimization [FL08]
- for all concavely-priced timed automata [JT08]
- for deciding frequency objectives [BBBS11,Sta12]
- ...

[BBL04] Bouyer, Brinksma, Larsen. Staying Alive As Cheaply As Possible (HSCC’04).
Outline

1. Introduction

2. Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3. Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4. Conclusion
Modelling the task graph scheduling problem

- **Processors**
  
  \[ P_1: \]
  
  \[ P_2: \]
  
  - **Tasks**
    
  \[ T_4: \]
  
  \[ T_5: \]
  
  - **Modelling energy**
    
  \[ P_1: \]
  
  \[ P_2: \]
Modelling the task graph scheduling problem

- **Processes**
  
  **$P_1$:**
  - Precondition: $x \leq 2$
  - Transitions:
    - $x = 2$ (addition $+90$, multiplication $\times 1$)
    - $x = 3$ (addition $+10$, multiplication $\times 1$)
  - Actions:
    - $x := 0$
    - $x := 0$
    - $x := 0$
  - States:
    - idle
    - done

- **$P_2$:**
  - Precondition: $y \leq 5$
  - Transitions:
    - $y = 5$ (addition $+30$, multiplication $\times 2$)
    - $y = 7$ (addition $+20$, multiplication $\times 2$)
  - Actions:
    - $x := 0$
    - $x := 0$
  - States:
    - idle
    - done

- **Tasks**
  
  **$T_4$:**
  - Action: $t_1 \land t_2$
  - States:
    - idle
    - done
  - Actions:
    - $t_4 := 1$

  **$T_5$:**
  - Action: $t_3$
  - States:
    - idle
    - done
  - Actions:
    - $t_5 := 1$

- **Modelling energy**
  
  **$P_1$:**
  - Precondition: $x \leq 2$
  - Transitions:
    - $x = 2$ (addition $+90$, multiplication $\times 1$)
    - $x = 3$ (addition $+10$, multiplication $\times 1$)
  - Actions:
    - $x := 0$
    - $x := 0$
    - $x := 0$
  - States:
    - idle
    - done

  **$P_2$:**
  - Precondition: $y \leq 5$
  - Transitions:
    - $y = 5$ (addition $+30$, multiplication $\times 2$)
    - $y = 7$ (addition $+20$, multiplication $\times 2$)
  - Actions:
    - $x := 0$
    - $x := 0$
  - States:
    - idle
    - done

- **Modelling uncertainty**
  
  **$P_1$:**
  - Precondition: $x \leq 2$
  - Transitions:
    - $x \geq 1$ (addition $+90$, multiplication $\times 1$)
    - $x \geq 1$ (addition $+10$, multiplication $\times 1$)
  - Actions:
    - $x := 0$
    - $x := 0$
  - States:
    - idle
    - done

  **$P_2$:**
  - Precondition: $y \leq 5$
  - Transitions:
    - $y \geq 3$ (addition $+30$, multiplication $\times 2$)
    - $y \geq 2$ (addition $+20$, multiplication $\times 2$)
  - Actions:
    - $x := 0$
    - $x := 0$
  - States:
    - idle
    - done
Modelling the task graph scheduling problem

- **Processors**
  - $P_1$: 
    - $x = 2$
    - $x := 0$
    - $x = 3$
    - $x := 0$
    - $(x \leq 2)$
    - $(x \leq 3)$
  - $P_2$: 
    - $y = 5$
    - $y := 0$
    - $y = 7$
    - $y := 0$
    - $(y \leq 5)$
    - $(y \leq 7)$

- **Tasks**
  - $T_4$: 
    - $t_1 \land t_2$
    - $t_4 := 1$
  - $T_5$: 
    - $t_3$
    - $t_5 := 1$

- **Modelling energy**
  - $P_1$: 
    - $x = 2$
    - $x := 0$
    - $x = 3$
    - $x := 0$
    - $y = 5$
    - $y := 0$
    - $y = 7$
    - $y := 0$
    - $(x \leq 2)$
    - $(x \leq 3)$
    - $(y \leq 5)$
    - $(y \leq 7)$

- **Modelling uncertainty**
  - $P_1$: 
    - $x \geq 1$
    - $x := 0$
    - $x \geq 1$
    - $x := 0$
    - $(x \leq 2)$
    - $(x \leq 3)$
  - $P_2$: 
    - $y \geq 3$
    - $y := 0$
    - $y \geq 2$
    - $y := 0$
    - $(y \leq 5)$
    - $(y \leq 7)$

A (good) schedule is a strategy in the product game (with a low cost)
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊

\[
\begin{align*}
\ell_0 & \xrightarrow{(x \leq 2)} \ell_1 \\
\ell_1 & \xrightarrow{x \geq 1, u_3} \ell_0 \\
\ell_1 & \xrightarrow{x \leq 1, c_1} \ell_2 \\
\ell_2 & \xrightarrow{x < 1, u_1} \ell_3 \\
\ell_3 & \xrightarrow{x \leq 1, c_3} \ell_2 \\
\ell_2 & \xrightarrow{x \geq 2, c_4} \ell_1 \\
\ell_1 & \xrightarrow{x < 1, u_2, x := 0} \ell_3 \\
\end{align*}
\]
An example of a timed game

Rule of the game

- **Aim**: avoid 🙁 and reach 🙂
- **How do we play?** According to a strategy:
An example of a timed game

Rule of the game

- **Aim**: avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]
An example of a timed game

Rule of the game
- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:
  \[ f : \text{history} \mapsto (\text{delay}, \text{cont. transition}) \]

A (memoryless) winning strategy
- from \((\ell_0, 0)\), play \((0.5, c_1)\)
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

A (memoryless) winning strategy

- from \((\ell_0, 0)\), play \((0.5, c_1)\)
  \(~\sim\) can be preempted by \(u_2\)
An example of a timed game

Rule of the game

- **Aim**: avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay}, \text{cont. transition}) \]

A (memoryless) winning strategy

- from \((\ell_0, 0)\), play \((0.5, c_1)\)
  
  \(\sim\) can be preempted by \(u_2\)
- from \((\ell_2, *)\), play \((1 - *, c_2)\)
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

\[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

A (memoryless) winning strategy

- From \((\ell_0, 0)\), play \((0.5, c_1)\)
- From \((\ell_2, \ast)\), play \((1 - \ast, c_2)\)
- From \((\ell_3, 1)\), play \((0, c_3)\)
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

A (memoryless) winning strategy

- From \((\ell_0, 0)\), play \((0.5, c_1)\)
  - can be preempted by \(u_2\)
- From \((\ell_2, \star)\), play \((1 - \star, c_2)\)
- From \((\ell_3, 1)\), play \((0, c_3)\)
- From \((\ell_1, 1)\), play \((1, c_4)\)
An example of a timed game

Rule of the game

- **Aim**: avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

Problems to be considered

- Weighted timed automata
- Timed games
- Weighted timed games
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

Problems to be considered

- Does there exist a winning strategy?
An example of a timed game

Rule of the game

- **Aim:** avoid 😞 and reach 😊
- **How do we play?** According to a strategy:

  \[ f : \text{history} \mapsto (\text{delay, cont. transition}) \]

Problems to be considered

- Does there exist a winning strategy?
- If yes, compute one (as simple as possible).
Decidability of timed games

**Theorem [AMPS98,HK99]**

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.
Decidability of timed games

Theorem [AMPS98,HK99]

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.

→ classical regions are sufficient for solving such problems
Decidability of timed games

Theorem [AMPS98,HK99]

Reachability and safety timed games are decidable and EXPTIME-complete. Furthermore memoryless and “region-based” strategies are sufficient.

\[ \sim \text{classical regions are sufficient for solving such problems} \]

Theorem [AM99,BHPR07,JT07]

Optimal-time reachability timed games are decidable and EXPTIME-complete.

Outline

1. Introduction

2. Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3. Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4. Conclusion
A simple timed game

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

\( \ell_1 \) (y = 0)

\[ \ell_1 \xrightarrow{u} \ell_2 \quad \ell_1 \xrightarrow{u} \ell_3 \]

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

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

Question: what is the optimal cost we can ensure while reaching \( \ell_2 \)?

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

strategy: wait in \( \ell_0 \), and when \( t = \frac{24}{5} \), go to \( \ell_1 \).
A simple weighted timed game

\[
\ell_0 \xleftarrow{+5} \xrightarrow{x \leq 2, c, y:=0} \ell_1 \xrightarrow{u} \ell_2 \xrightarrow{x=2, c} +1 \xrightarrow{u} \ell_3 \xrightarrow{x=2, c} +1 \xrightarrow{u} \text{happy face}
\]

**Question:**

What is the optimal cost we can ensure while reaching the happy face?

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

\[ \ell_0 + 5 \xrightarrow{x \leq 2, c, y := 0} \ell_1 \]

\[ (y = 0) \]

\[ \ell_1 \xrightarrow{u} \ell_2 + 10 \]

\[ x = 2, c \]

\[ \ell_2 \xrightarrow{u} \ell_3 + 1 \]

\[ x = 2, c \]

\[ \ell_3 \xrightarrow{u} \text{smiley} \]

Question: what is the optimal cost we can ensure while reaching \( \text{smiley} \)?
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 ☺️?

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

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

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

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

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

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

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

\sim strategy: \text{ wait in } l_0, \text{ and when } t = \frac{4}{3}, \text{ go to } l_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.
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.

[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.
Optimal reachability in weighted timed games (2)

**[BBR05, BBM06]**

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

**[BLMR06, Rut11, HIM13, BGK+14]**

Turn-based optimal timed games are decidable in \text{EXPTIME} (resp. \text{PTIME}) when automata have a single clock (resp. with two rates). They are \text{PTIME}-hard.
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

![Diagram](image-url)
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

![Diagram](image)  

... but memoryless almost-optimal strategies will be sufficient.
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

\[
\begin{align*}
\ell_0 & \\
& \xrightarrow{+2} \quad (x \leq 1) \\
& \xrightarrow{x<1} \quad x:=0 \\
& \xrightarrow{x>0} \\
\ell_1 & \\
& \xrightarrow{+1} \quad x=1
\end{align*}
\]

... but memoryless almost-optimal strategies will be sufficient.

- Key: resetting the clock somehow resets the history...
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

\[
\begin{align*}
\ell_0 & \xrightarrow{+2} \ \ (x \leq 1) \\
\ell_0 & \xrightarrow{x < 1} \ x := 0 \\
\ell_1 & \xrightarrow{+1} \\
\ell_1 & \xrightarrow{x > 0} \ x = 1 \\
\end{align*}
\]

... but memoryless almost-optimal strategies will be sufficient.

- Key: resetting the clock somehow resets the history...
- By unfolding and removing one by one the locations, we can synthesize memoryless almost-optimal winning strategies.
What is easier with a single clock?

- Memoryless strategies can be non-optimal...

  ![Diagram]

  (\(x \leq 1\))

  \(x = 1\)

  ... but memoryless almost-optimal strategies will be sufficient.

- Key: resetting the clock somehow resets the history...

- By unfolding and removing one by one the locations, we can synthesize memoryless almost-optimal winning strategies.

- Rather involved proofs of correctness
Weighted timed automata
Timed games
Weighted timed games

\[
\sigma(c_2, x) = \begin{cases} 
    c_2^{out} & \text{if } 0 \leq x < 2/5 \\
    c_2 & \text{if } 2/5 \leq x < 1/2 \\
    u_2 & \text{if } 1/2 \leq x \leq 1 
\end{cases}
\]
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 $x_0$.

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

The cost is increased by $1-x_0$. 

$$\text{Add}^+(x)$$

- $y=1, y:=0$
- $x=1, x:=0$
- $z=0$

$$\text{Add}^-(x)$$

- $y=1, y:=0$
- $x=1, x:=0$
- $z=0$
Computing the optimal cost: why is that hard?

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

![Diagram showing the process of computing the optimal cost with two branches and a decision based on the value of $y$.]
Computing the optimal cost: why is that hard?

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

- $\text{In } z = 0$, 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 $y = 2x$.

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

Given two clocks \( x \) and \( y \), we can check whether \( y = 2x \).

- If \( y_0 < 2x_0 \), player 2 chooses the first branch: \( \text{cost} > 3 \)
- \( \text{In } \), \( \text{cost} = 2x_0 + (1 - y_0) + 2 \)
- \( \text{In } \), \( \text{cost} = 2(1 - x_0) + y_0 + 1 \)

If \( y_0 < 2x_0 \), player 2 chooses the first branch: \( \text{cost} > 3 \)
Computing the optimal cost: why is that hard?

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

- In $\bigcirc$, cost = $2x_0 + (1 - y_0) + 2$
- In $\bigcirc$, 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 $y = 2x$.

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

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

- In $\bigcirc$, cost $= 2x_0 + (1 - y_0) + 2$
- In $\bigcirc$, 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 $= 3$

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

\[
x=1, x:=0 \quad \vee \quad y=1, y:=0
\]

Test $y(x=2z)$

\[
u:=0 \quad \quad \quad z:=0 \quad \quad \quad u=1, u:=0 \quad (u=0)
\]
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.

$$\begin{align*}
x = 1, & x := 0 \\
\lor & y = 1, y := 0
\end{align*}$$

Test$_y(x = 2z)$
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:

\[
\begin{align*}
  x &= \frac{1}{2c_1} \\
  y &= \frac{1}{3c_2}
\end{align*}
\]

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.
Are we done?
Outline

1. Introduction

2. Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3. Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4. Conclusion
Are we done?
Are we done? No! Let’s be a bit more precise!
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 $G$ 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 $G$ 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:
  \[
  \text{optcost}_G = \inf \{ \text{cost}(\sigma) \mid \sigma \text{ winning strat.} \}
  \]
  (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 \( \mathcal{G} \) 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)

Two problems of interest
- The value problem asks, given \( \mathcal{G} \) and a threshold \( \triangleright c \), whether \( \text{optcost}_{\mathcal{G}} \triangleright c \)?
- The existence problem asks, given \( \mathcal{G} \) and a threshold \( \triangleright c \), whether there exists a winning strategy in \( \mathcal{G} \) such that \( \text{cost}(\sigma) \triangleright c \)?
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:**

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

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

**Two problems of interest**

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

*Note: These problems are distinct...*
- 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.
Outline of the rest of the talk

1. Show that the value problem is undecidable in weighted timed games
Show that the value problem is undecidable in weighted timed games

This is intellectually satisfactory to not have this discrepancy in the set of results
Outline of the rest of the talk

1. Show that the value problem is undecidable in weighted timed games
   - This is intellectually satisfactory to not have this discrepancy in the set of results
   - An original undecidability proof, based on a diagonal construction
     - This method has been introduced in the context of quantitative temporal logics [BMM14]
     - It might be useful in some different contexts

Outline of the rest of the talk

1. Show that the value problem is undecidable in weighted timed games
   - This is intellectually satisfactory to not have this discrepancy in the set of results
   - An original undecidability proof, based on a diagonal construction
     - This method has been introduced in the context of quantitative temporal logics [BMM14]
     - It might be useful in some different contexts

2. Propose 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...

Outline

1. Introduction

2. Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3. Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4. Conclusion
What about the value problem?

\[ \mathcal{M} \sim \text{the previous game } G_M \]
What about the value problem?

\( \mathcal{M} \sim \text{the previous game } G_\mathcal{M} \)

It is always the case that \( \text{optcost}_{G_\mathcal{M}} \geq 3 \)
What about the value problem?

\[ \mathcal{M} \sim \text{the previous game } G_\mathcal{M} \]

It is always the case that \( \text{optcost}_{G_\mathcal{M}} \geq 3 \)

- If \( \mathcal{M} \) halts, then \( \text{optcost}_{G_\mathcal{M}} = 3 \)
What about the value problem?

\( \mathcal{M} \sim \) the previous game \( \mathcal{G}_M \)

It is always the case that \( \text{optcost}_{\mathcal{G}_M} \geq 3 \)

- If \( \mathcal{M} \) halts, then \( \text{optcost}_{\mathcal{G}_M} = 3 \)
- It might be the case that \( \mathcal{M} \) does not halt but \( \text{optcost}_{\mathcal{G}_M} = 3 \)
What about the value problem?

$\mathcal{M} \sim$ the previous game $G_{\mathcal{M}}$

It is always the case that $\text{optcost}_{G_{\mathcal{M}}} \geq 3$

- If $\mathcal{M}$ halts, then $\text{optcost}_{G_{\mathcal{M}}} = 3$
- It might be the case that $\mathcal{M}$ does not halt but $\text{optcost}_{G_{\mathcal{M}}} = 3$

$\mathcal{M}$ does not halt but $\text{optcost}_{G_{\mathcal{M}}} = 3$
What about the value problem?

$\mathcal{M} \sim$ the previous game $G_\mathcal{M}$

It is always the case that $\text{optcost}_{G_\mathcal{M}} \geq 3$

- If $\mathcal{M}$ halts, then $\text{optcost}_{G_\mathcal{M}} = 3$
- It might be the case that $\mathcal{M}$ does not halt but $\text{optcost}_{G_\mathcal{M}} = 3$
What about the value problem?

\[ \mathcal{M} \sim \text{the previous game } G_\mathcal{M} \]

It is always the case that \( \text{optcost}_{G_\mathcal{M}} \geq 3 \)

- If \( \mathcal{M} \) halts, then \( \text{optcost}_{G_\mathcal{M}} = 3 \)
- It might be the case that \( \mathcal{M} \) does not halt but \( \text{optcost}_{G_\mathcal{M}} = 3 \)

\[ \mathcal{M} \text{ does not halt but } \text{optcost}_{G_\mathcal{M}} = 3 \]
We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.
We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.

We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14].

We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.

We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14].

**Technical lemma**

We assume two halting states: accept and reject.
We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.

We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14].

**Technical lemma**

We assume two halting states: accept and reject. If $\text{optcost}_{G,M} = 3$ but no strategy has cost 3.
We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.

We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14].

**Technical lemma**

We assume two halting states: accept and reject. If $\text{optcost}_{G_M} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $M$ is not accepting),

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL (*CONCUR’14*).
We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.

We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14].

**Technical lemma**

We assume two halting states: accept and reject. If $\text{optcost}_{G_M} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $M$ is not accepting), then the unique valid run of $M$ is infinite.

[BMM14] Bouyer, Markey, Matteplackel. Averaging in LTL (*CONCUR’14*).
We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.

We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14].

**Technical lemma**

We assume two halting states: accept and reject. If optcost$_{G,M} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $M$ is not accepting), then the unique valid run of $M$ is infinite.

---

We need to be able to distinguish between machines that halt and machines that have a converging phenomenon.

We will use a diagonal argument, that has been developed recently in the context of quantitative temporal logic [BMM14].

**Technical lemma**

We assume two halting states: accept and reject. If $\text{optcost}_{G_M} = 3$ but no strategy has cost 3 (or equivalently, the unique valid run of $M$ is not accepting), then the unique valid run of $M$ is infinite.

The diagonal argument

\[ B \text{ det. Turing machine can either accept, reject, or not halt} \]
\[ \rightarrow M(B) \text{ two-counter machine which simulates } B \text{ on } B \]
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

$\rightarrow \quad \mathcal{M}(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} 
  \text{accept} & \text{if } \text{optcost}_{\mathcal{M}(B)} = 3 \\
  \text{reject} & \text{otherwise}
\end{cases}$$
The diagonal argument

\[ B \text{ det. Turing machine can either accept, reject, or not halt} \]

\[ \rightarrow \quad \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \]

We define the program

\[ \mathcal{H} : B \mapsto \begin{cases} \\text{accept} & \text{if } \text{optcost}_{\mathcal{G}_\mathcal{M}(B)} = 3 \\text{reject} & \text{otherwise} \end{cases} \]

The function \( \mathcal{H} \) is not computable.
The diagonal argument

\( B \) det. Turing machine can either accept, reject, or not halt
\[ \rightarrow \quad \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \]

We define the program

\[ \mathcal{H} : B \mapsto \begin{cases} 
    \text{accept} & \text{if } \text{optcost}_{\mathcal{M}(B)} = 3 \\
    \text{reject} & \text{otherwise}
\end{cases} \]

The function \( \mathcal{H} \) is not computable.

Towards a contradiction, assume it is computable by det. Turing machine \( T_{\mathcal{H}} \),...
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

$\rightarrow M(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$$H : B \mapsto \begin{cases} 
accept & \text{if } \text{optcost}_{M(B)} = 3 \\
reject & \text{otherwise}
\end{cases}$$

The function $H$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_H$, and define the program:

$C(B) :$ Simulate $T_H$ on $B$;
If $T_H$ accepts $B$ then reject, otherwise accept
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

$\rightarrow \ M(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$H : B \mapsto \begin{cases} 
\text{accept} & \text{if } \text{optcost}_{G_{M(B)}} = 3 \\
\text{reject} & \text{otherwise}
\end{cases}$

The function $H$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_H$, and define the program:

$C(B) : \begin{align*}
\text{Simulate } T_H \text{ on } B; \\
\text{If } T_H \text{ accepts } B \text{ then reject, otherwise accept}
\end{align*}$

Program $C$ is deterministic hence we can run $C$ on $C$
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

$\rightarrow \ M(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} 
    \text{accept} & \text{if } \text{optcost}_{G,M(B)} = 3 \\
    \text{reject} & \text{otherwise}
\end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_{\mathcal{H}}$, and define the program:

$$C(B) : \quad \text{Simulate } T_{\mathcal{H}} \text{ on } B;$$

If $T_{\mathcal{H}}$ accepts $B$ then reject, otherwise accept

Program $C$ is deterministic hence we can run $C$ on $C$

Program $C$ always terminates:

- Assume $C$ accepts $C$: 
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

$\rightarrow \quad \mathcal{M}(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} 
accept & \text{if } \text{optcost}_{\mathcal{M}(B)} = 3 \\ 
reject & \text{otherwise}
\end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_{\mathcal{H}}$, and define the program:

$$C(B) : \quad \text{Simulate } T_{\mathcal{H}} \text{ on } B;$$

If $T_{\mathcal{H}}$ accepts $B$ then reject, otherwise accept

Program $C$ is deterministic hence we can run $C$ on $C$.
Program $C$ always terminates:

- Assume $C$ accepts $C$: this means that $\mathcal{H}(C) = \text{reject}$,
The diagonal argument

Let $B$ be a Turing machine. Turing machine can either accept, reject, or not halt.

$\rightarrow$ \[ M(B) \] two-counter machine which simulates $B$ on $B$.

We define the program $H : B \mapsto \begin{cases} accept & \text{if optcost}_{G, M(B)} = 3 \\ reject & \text{otherwise} \end{cases}$

The function $H$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_H$, and define the program:

$C(B) :$ Simulate $T_H$ on $B$.

If $T_H$ accepts $B$ then reject, otherwise accept.

Program $C$ is deterministic hence we can run $C$ on $C$.

Program $C$ always terminates:

- Assume $C$ accepts $C$: this means that $H(C) = reject$, hence $\text{optcost}_{G, M(C)} > 3$. 

The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

→ $M(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$$H: B \mapsto \begin{cases} \text{accept} & \text{if optcost}_{M(B)} = 3 \\ \text{reject} & \text{otherwise} \end{cases}$$

The function $H$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_H$, and define the program:

$$C(B): \text{Simulate } T_H \text{ on } B;$$

If $T_H$ accepts $B$ then reject, otherwise accept

Program $C$ is deterministic hence we can run $C$ on $C$

Program $C$ always terminates:

- Assume $C$ accepts $C$: this means that $H(C) = \text{reject}$, hence $\text{optcost}_{M(C)} > 3$.
  This implies $M(C)$ does not accept,
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

$\rightarrow \ M(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$$\mathcal{H} : B \mapsto \begin{cases} 
  \text{accept} & \text{if } \text{optcost}_M(B) = 3 \\
  \text{reject} & \text{otherwise}
\end{cases}$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_{\mathcal{H}}$, and define the program:

$$C(B) : \text{Simulate } T_{\mathcal{H}} \text{ on } B;$$

$$\text{If } T_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept}$$

Program $C$ is deterministic hence we can run $C$ on $C$.

Program $C$ always terminates:

- Assume $C$ accepts $C$: this means that $\mathcal{H}(C) = \text{reject}$, hence $\text{optcost}_M(C) > 3$.

  This implies $M(C)$ does not accept, and therefore $C$ does not accept $C$. 


The diagonal argument

Let $B$ be a deterministic Turing machine. We can either accept, reject, or not halt.

$\rightarrow \quad M(B)$ is a two-counter machine that simulates $B$ on $B$.

We define the program $H: B \mapsto \begin{cases} \text{accept} & \text{if } \text{optcost}_{G_{M(B)}} = 3 \\ \text{reject} & \text{otherwise} \end{cases}$

The function $H$ is not computable.

Towards a contradiction, assume it is computable by a deterministic Turing machine $T_H$, and define the program:

$C(B) : \text{Simulate } T_H \text{ on } B$;

If $T_H$ accepts $B$ then reject, otherwise accept.

Program $C$ is deterministic hence we can run $C$ on $C$.

Program $C$ always terminates:

1. Assume $C$ accepts $C$: this means that $H(C) = \text{reject}$, hence $\text{optcost}_{G_{M(C)}} > 3$.
   This implies $M(C)$ does not accept, and therefore $C$ does not accept $C$.
   contradiction: $C$ rejects $C$.  

\[ \text{optcost}_{G_{M(C)}} = 3 \]. Since $C$ does not accept $C$, the unique valid computation of $M(C)$ is either infinite or rejecting. Applying the lemma on the previous slide, it is infinite, which contradicts the fact that $C$ always terminates.

Therefore, $H$ is not computable.
The diagonal argument

\[ B \text{ det. Turing machine can either accept, reject, or not halt} \]
\[ \rightarrow \quad \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \]

We define the program \( \mathcal{H} : B \mapsto \begin{cases} 
\text{accept} & \text{if } \text{optcost}_{G \mathcal{M}(B)} = 3 \\
\text{reject} & \text{otherwise}
\end{cases} \]

The function \( \mathcal{H} \) is not computable.

Towards a contradiction, assume it is computable by det. Turing machine \( T_{\mathcal{H}} \), and define the program:

\[ C(B) : \text{Simulate } T_{\mathcal{H}} \text{ on } B; \]
\[ \text{If } T_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \]

Program \( C \) is deterministic hence we can run \( C \) on \( C \)

Program \( C \) always terminates:

- Assume \( C \) accepts \( C \): this means that \( \mathcal{H}(C) = \text{reject} \), hence \( \text{optcost}_{G \mathcal{M}(C)} > 3 \).
  This implies \( \mathcal{M}(C) \) does not accept, and therefore \( C \) does not accept \( C \), contradiction: \( C \) rejects \( C \).

- \( \text{optcost}_{G \mathcal{M}(C)} = 3 \).
The diagonal argument

\[ B \text{ det. Turing machine can either accept, reject, or not halt} \]
\[ \implies \mathcal{M}(B) \text{ two-counter machine which simulates } B \text{ on } B \]

We define the program
\[ \mathcal{H} : B \mapsto \begin{cases} 
  \text{accept} & \text{if } \text{optcost}_{\mathcal{G}\mathcal{M}(B)} = 3 \\
  \text{reject} & \text{otherwise}
\end{cases} \]

The function \( \mathcal{H} \) is not computable.

Towards a contradiction, assume it is computable by det. Turing machine \( T_{\mathcal{H}} \), and define the program:
\[ C(B) : \text{Simulate } T_{\mathcal{H}} \text{ on } B; \]
\[ \text{If } T_{\mathcal{H}} \text{ accepts } B \text{ then reject, otherwise accept} \]

Program \( C \) is deterministic hence we can run \( C \) on \( C \).
Program \( C \) always terminates:
- Assume \( C \) accepts \( C \): this means that \( \mathcal{H}(C) = \text{reject} \), hence \( \text{optcost}_{\mathcal{G}\mathcal{M}(C)} > 3 \).
  This implies \( \mathcal{M}(C) \) does not accept, and therefore \( C \) does not accept \( C \), contradiction: \( C \) rejects \( C \).
- \( \text{optcost}_{\mathcal{G}\mathcal{M}(C)} = 3 \). Since \( C \) does not accept \( C \), the unique valid computation of \( \mathcal{M}(C) \) is either infinite or rejecting.
The diagonal argument

Let $B$ be a two-counter machine that simulates $B$ on $B$.

We define the program

$$H : B \mapsto \begin{cases} \text{accept} & \text{if optcost}_{G_M(B)} = 3 \\ \text{reject} & \text{otherwise} \end{cases}$$

The function $H$ is not computable.

Towards a contradiction, assume it is computable by a deterministic Turing machine $T_H$, and define the program:

$$C(B) : \text{Simulate } T_H \text{ on } B;$$

If $T_H$ accepts $B$ then reject, otherwise accept

Program $C$ is deterministic hence we can run $C$ on $C$.

Program $C$ always terminates:

- Assume $C$ accepts $C$: this means that $H(C) = \text{reject}$, hence $\text{optcost}_{G_M(C)} > 3$. This implies $M(C)$ does not accept, and therefore $C$ does not accept $C$, contradiction: $C$ rejects $C$.

- $\text{optcost}_{G_M(C)} = 3$. Since $C$ does not accept $C$, the unique valid computation of $M(C)$ is either infinite or rejecting. Applying the lemma on previous slide, it is infinite,
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt
$\rightarrow \ M(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$H : B \mapsto \begin{cases} 
    \text{accept} & \text{if } \text{optcost}_{\mathcal{G}M(B)} = 3 \\
    \text{reject} & \text{otherwise}
\end{cases}$

The function $H$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_H$, and define the program:

$C(B) : \text{Simulate } T_H \text{ on } B; \quad \text{If } T_H \text{ accepts } B \text{ then reject, otherwise accept}$

Program $C$ is deterministic hence we can run $C$ on $C$.
Program $C$ always terminates:

- Assume $C$ accepts $C$: this means that $H(C) = \text{reject}$, hence $\text{optcost}_{\mathcal{G}M(C)} > 3$. This implies $\mathcal{M}(C)$ does not accept, and therefore $C$ does not accept $C$, contradiction: $C$ rejects $C$.

- $\text{optcost}_{\mathcal{G}M(C)} = 3$. Since $C$ does not accept $C$, the unique valid computation of $\mathcal{M}(C)$ is either infinite or rejecting. Applying the lemma on previous slide, it is infinite, which contradicts the fact that $C$ always terminates.
The diagonal argument

$B$ det. Turing machine can either accept, reject, or not halt

$\rightarrow M(B)$ two-counter machine which simulates $B$ on $B$

We define the program

$$
\mathcal{H} : B \mapsto \begin{cases} 
  \text{accept} & \text{if } \text{optcost}_{G_{M(B)}} = 3 \\
  \text{reject} & \text{otherwise}
\end{cases}
$$

The function $\mathcal{H}$ is not computable.

Towards a contradiction, assume it is computable by det. Turing machine $T_\mathcal{H}$, and define the program:

$$
C(B) : \text{Simulate } T_\mathcal{H} \text{ on } B;
\quad \text{If } T_\mathcal{H} \text{ accepts } B \text{ then reject, otherwise accept}
$$

Program $C$ is deterministic hence we can run $C$ on $C$

Program $C$ always terminates:

- Assume $C$ accepts $C$: this means that $\mathcal{H}(C) = \text{reject}$, hence $\text{optcost}_{G_{M(C)}} > 3$. This implies $M(C)$ does not accept, and therefore $C$ does not accept $C$, contradiction: $C$ rejects $C$.

- $\text{optcost}_{G_{M(C)}} = 3$. Since $C$ does not accept $C$, the unique valid computation of $M(C)$ is either infinite or rejecting. Applying the lemma on previous slide, it is infinite, which contradicts the fact that $C$ always terminates.

Therefore, $\mathcal{H}$ is not computable.
Theorem [BJM15]

The value problem is undecidable in weighted timed games (with four clocks or more).

- Remark on the reduction:
  - Cost 0 within the core of the game
  - The rest of the game is acyclic

Outline

1. Introduction

2. Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3. Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4. Conclusion
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. \[\text{[BJM15]}\]

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

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

\text{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 \(\rho\) read on \(C\),

\[
\text{cost}(\rho) \geq \kappa
\]

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 \(\rho\) read on \(C\),

\[
\text{cost}(\rho) \geq \kappa \quad \text{or} \quad \text{cost}(\rho) = 0
\]

Note: In both cases, we can assume \(\kappa = 1\).

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 $v_\epsilon^- \leq \text{optcost}_G \leq v_\epsilon^+$
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]
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
  
  $$|v_\epsilon^+ - v_\epsilon^-| < \epsilon \text{ and } v_\epsilon^- \leq \text{optcost}_G \leq v_\epsilon^+$$

- 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
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

First: split the game along regions!

\[
\begin{align*}
g, Y &:= 0 \\
r_1, Y &:= 0 \\
r_2, Y &:= 0 \\
r_3, Y &:= 0 \\
r_4, Y &:= 0 \\
r_5, Y &:= 0
\end{align*}
\]
Semi-unfolding

Kernel $\mathcal{K}$

Hypothesis: \( \text{cost} > 0 \) implies \( \text{cost} \geq \kappa \)

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: \( \text{cost} > 0 \) implies \( \text{cost} \geq \kappa \)

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 \( \text{optcost} \)).
Semi-unfolding

Hypothesis: $\text{cost} > 0$ implies $\text{cost} \geq \kappa$

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 ≥ \( \kappa \)
Semi-unfolding

Hypothesis: cost > 0 implies cost ≥ κ

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

Exact computation

Approximation

Undecidability of the value problem
Approximation of the optimal cost
First step: Tree-like parts

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

\[ \ell, \ell', \ell'' \]

\[ g', Y' \]

\[ c', Y' \]

\[ g'', Y'' \]

\[ c'', Y'' \]

\[ \sim \text{ Goes back to } [\text{LMM02}] \]
First step: Tree-like parts

\[O(\ell, v) = \min\{g', Y'\leftarrow 0(v + t')\} = \min\{g'', Y''\leftarrow 0(v + t'')\}\]

\[g', Y', \ell' \quad g'', Y'', \ell''\]

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

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

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

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

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

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

\[ g', Y' \]

\[ c' \]

\[ g'', Y'' \]

\[ c'' \]

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

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

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

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

\[ O(\ell', v') = \text{[LMM02]} \]

\[ O(\ell'', v'') = \text{[LMM02]} \]

\[ \sim \text{ Goes back to [LMM02]} \]
First step: Tree-like parts

\[ O(\ell, v) = \inf_{t' \mid v + t' = g'} \max(\alpha, \beta) \]
\[ (\alpha) = t' c + c' + O(\ell', v') \]
\[ v' = [Y' \leftarrow 0](v + t') \]
First step: Tree-like parts

$$O(\ell, v) = \inf_{t' \mid v + t' \models g'} \max((\alpha), (\beta))$$

$$O(\ell, v) = \inf_{t' \mid v + t' \models g'} \max((\alpha), (\beta))$$

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

$$(\beta) = \sup_{t'' \leq t' \mid v + t'' \models g''} t''c + c'' + O(\ell'', v'')$$

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

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

---

[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

1. 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

Output cost functions $f$
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

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 \( f_{\epsilon}^+ \)

Output cost functions \( f \)
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

3. 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 $f_\epsilon$)

$f_\epsilon$: constant  $f_\epsilon$: 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

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

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

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 $f_\varepsilon$)

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

3. 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

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

1 Introduction

2 Overview of “old” results
   - Weighted timed automata
   - Timed games
   - Weighted timed games

3 Some recent developments
   - Undecidability of the value problem
   - Approximation of the optimal cost

4 Conclusion
Conclusion

Summary of the talk

- Quick overview of results concerning the optimal reachability problem in weighted timed games
- 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 ($2^{\exp(|G|)} \cdot \frac{1}{\epsilon} |X|$), and
- Extend to the whole class of weighted timed games, or understand why it is not possible

Assume stochastic uncertainty
Conclusion

Summary of the talk

- Quick overview of results concerning the optimal reachability problem in weighted timed games
- 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 \((2^{\exp(|G|)} \cdot \left(\frac{1}{\epsilon}\right)^{|X|})\), and implement it
- Extend to the whole class of weighted timed games, or understand why it is not possible
- Assume stochastic uncertainty