ON THE OPTIMAL REACHABILITY PROBLEM IN WEIGHTED TIMED AUTOMATA AND GAMES

Patricia Bouyer

LSV, CNRS & ENS Cachan, France
bouyer@lsv.ens-cachan.fr

Abstract
In these notes, we survey works made on the models of weighted timed automata and games, and more specifically on the optimal reachability problem.

1 Introduction

In thirty years computerized systems have widely spread in our society, from ubiquitous electronic appliances (communication devices, automotive equipment, *etc*), to internet transactions (e-banking, e-business, *etc*), to new technologies (like wireless communications), and to critical systems (medical devices, industrial plants, *etc*). Due to their rapid development, such systems have become more and more complex, and unfortunately this development has come with many bugs, from arithmetic overflow (which caused the crash of the Ariane 5 rocket in 1996) to race conditions (which caused the lethal dysfunction of the Therac-25 radiotherapy machine in the late 80's) or infinite loops (for instance the leap-year bug turning all Zune MP3 devices off on 31 December 2008). Many of those bugs could have been avoided if implemented softwares had been formally verified prior to their use. The need for formal methods for verifying and certifying computer-driven systems is therefore blatant.

Toward the development of more reliable computerized systems, several verification approaches have been developed, among which the so-called model-checking technique. Model-checking is a model-based approach to verification, which goes back to the late seventies [45, 31, 46]. Given a system $\mathcal{S}$ and a property P, the model-checking approach consists in constructing a mathematical model $\mathcal{M}_{\mathcal{S}}$ for the system and a mathematical model φ_P for the property, for which we will be able to *automatically* check that $\mathcal{M}_{\mathcal{S}}$ satisfies φ_P. If the models $\mathcal{M}_{\mathcal{S}}$ and φ_P are accurate enough with respect to $\mathcal{S}$ and P respectively, we will deduce with confidence that the system $\mathcal{S}$ satisfies the property P. This approach requires the development of expressive modelling formalisms (to increase faithfulness of models) and efficient algorithms.

These last twenty years a huge effort has been made to design expressive models for representing computerized systems. As part of this effort the model of timed automata has been proposed

in the early nineties [4, 5], as a powerful and suitable model to reason about (the correctness of) real-time computerized systems. Timed automata extend finite-state automata with several clocks, which can be used to enforce timing constraints between various events in the system. They provide a convenient formalism and enjoy reasonably-efficient algorithms (*e.g.* reachability can be decided using polynomial space), which explains the enormous interest that they provoked in the community of formal methods. Timed games [8] extend timed automata with a way of modelling systems interacting with external, uncontrollable components: some transitions of the automaton cannot be forced or prevented to happen. The reachability problem then asks whether there is a *strategy* to reach a given state, whatever the uncontrollable components do. This problem can also be decided, in exponential time.

Timed automata and games are not powerful enough for representing quantities like resources, prices, temperature, etc. The more general model of hybrid automata [3, 2, 37, 38] (see [47] for a survey) allows for accurate modelling of such quantities using hybrid variables. The evolution of these variables follow differential equations, depending on the state of the system, and this unfortunately makes the reachability problem undecidable [38], even in the restricted case of stopwatches, which are clocks that can be stopped and restarted.

Weighted (or priced) timed automata [6, 9] and games [42, 1, 17] have been proposed in the early 2000's as an intermediary model for modelling resource consumption or allocation problems in real-time systems (*eg* optimal scheduling [11]). As opposed to (linear) hybrid systems, an execution in a weighted timed model is simply one in the underlying timed model: the extra quantitative information is just an *observer* of the system, and it does not modify the possible behaviours of the system. Figure 4 displays an example of a weighted timed game: each location carries an integer, which is the rate by which the weight (we will also call it *cost* thereafter) increases when time elapses in that location. Some edges also carry a weight, which indicates how much the cost increases when crossing this edge. The cost of an execution is then the accumulated sum of the costs of all individual moves along the execution, and this cost value is a *quantitative measure* of the quality of the execution. Dashed edges are uncontrollable, and cannot be forced or prevented to occur; they appear in timed games only. Notice that the constraints on edges never depend on the value of the cost, but only on the values of the clocks.

In these notes, we investigate the optimal reachability problem in weighted timed automata and games: given a target location, we want to know what is the optimal (*i.e.* smallest) cost for reaching the target location, and what is a corresponding strategy? We will survey the main results that have been obtained on that problem. We will start with a motivating example (Section 2). In Section 3, we will focus on the automaton model, state the main decidability result, and give a glimpse of the new abstraction that may be used in this context. In Section 4, we will overview most of the results which have been obtained on this problem; we will also give some details on some of the technics that have been used. We will then show how we can use the models studied in this paper to model the initial motivating example (Section 5).

2 An example: The task graph scheduling problem

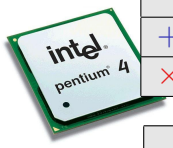
In this section, we give an example of problem, that we will be able to model and solve using the developments presented in this paper.

We want to compute the following arithmetical expression:

$$D{\times}(C{\times}(A{+}B)){+}(A{+}B){+}(C{\times}D)$$

using two processors, whose characteristics are given in Figure 1.

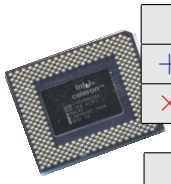
P_1 (fast):



time	
+	2 picoseconds
×	3 picoseconds

energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



time	
+	5 picoseconds
×	7 picoseconds

energy	
idle	20 Watts
in use	30 Watts

Figure 1: Characteristics of the two processors

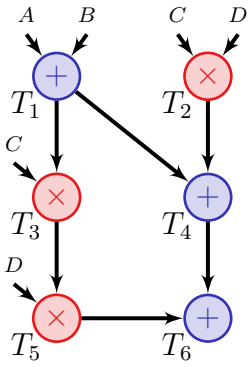


Figure 2: The task graph

The task graph giving the logical dependencies of the various atomic tasks that need to be done for computing the arithmetical expression is depicted on Figure 2. It reads as follows: Task T_3 corresponds to the outermost multiplication in sub-expression $C{\times}(A{+}B)$. It requires first the addition $A{+}B$ to be computed, which is why the gate T_3 has two inputs: C and the output of T_1 (implying that T_1 needs to be computed prior to T_3).

There are many possible schedules that satisfy the logical dependencies given by the task graph, and allow to compute the global arithmetical expression, three of them are given on Figure 3. Two of their characteristics are summarized (time of execution and energy consumption).

The theory we will describe in these notes will allow to model this system, and compute time-optimal, as well as cost-optimal, schedules. We will come back to this example in Section 5.

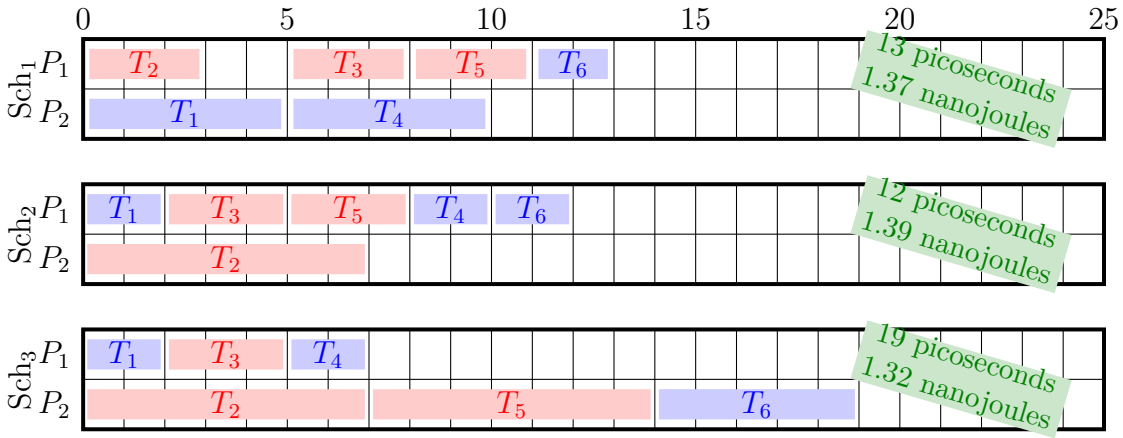


Figure 3: Three possible schedules and their characteristics

3 Optimal reachability in weighted timed automata

3.1 Weighted timed automata

In this section we introduce the weighted (also named priced) timed automaton model, that has been proposed in 2001 for representing resource consumption in real-time systems [6, 9]

We consider as time domain the set $\mathbb{R}_{\geqslant 0}$ of non-negative reals. We let X be a finite set of variables, called *clocks*. A *(clock) valuation* over X is a mapping $v : X \to \mathbb{R}_{\geqslant 0}$ that assigns to each clock a time value. The set of all valuations over X is denoted $\mathbb{R}_{\geqslant 0}^X$. Let $t \in \mathbb{R}_{\geqslant 0}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$ for every $x \in X$. For $Y \subseteq X$, we denote by $[Y \leftarrow 0]v$ the valuation assigning 0 (respectively $v(x)$) to every $x \in Y$ (respectively $x \in X \setminus Y$). We write $\mathbf{0}_X$ for the valuation which assigns 0 to every clock $x \in X$.

The set of *clock constraints* over X, denoted $\mathcal{C}(X)$, is defined by the grammar:

$$g \ ::= \ x \sim c \ \mid \ g \wedge g$$

where $x \in X$ is a clock, $c \in \mathbb{N}$, and $\sim \in \{<, \leqslant, =, \geqslant, >\}$.

Clock constraints are evaluated over clock valuations, and the satisfaction relation, denoted $v \models g$, is defined inductively by $v \models (x \sim c)$ whenever $v(x) \sim c$, and $v \models g_1 \wedge g_2$ whenever $v \models g_1$ and $v \models g_2$.

Definition 3.1. *A* weighted (or priced) timed automaton *is a tuple* $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{cost})$ *where* X *is a finite set of clocks,* L *is a finite set of locations,* $\ell_0 \in L$ *is the initial location,* $\mathsf{Goal} \subseteq L$ *is a set of goal (or final) locations,* $E \subseteq L \times \mathcal{C}(X) \times 2^X \times L$ *is a finite set of edges (or transitions), and* $\mathsf{cost} : L \cup E \to \mathbb{N}$ *is a* cost function *which assigns a value to each location and to each transition. The cost (function)* cost *is said* stopwatch *whenever* $\mathsf{cost}(L) \subseteq \{0, 1\}$ *and* $\mathsf{cost}(E) = \{0\}$.

In the above definition, if we forget about the cost function, we obtain the well-known model of *timed automata* [4, 5]. The semantics of a weighted timed automaton is that of the underlying timed automaton, and the role of the cost function will be to give a quantitative information to the moves and the executions in the system.

We therefore start by recalling the semantics of a timed automaton $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E)$. It is given as a timed transition system $\mathcal{T}_\mathcal{A} = (S, s_0, \to)$ where $S = L \times \mathbb{R}^X_{\geqslant 0}$ is the set of configurations (or states) of $\mathcal{A}$, $s_0 = (\ell_0, \mathbf{0}_X)$ is the initial configuration, and $\to$ contains two types of moves:

- *delay moves:* $(\ell, v) \xrightarrow{t} (\ell, v + t)$ if $t \in \mathbb{R}_{\geqslant 0}$;
- *discrete moves:* $(\ell, v) \xrightarrow{e} (\ell', v')$ if there exists an edge $e = (\ell, g, Y, \ell')$ in E such that $v \models g$, $v' = [Y \leftarrow 0]v$.

A *run* ϱ in $\mathcal{A}$ is a finite or infinite sequence of moves in the transition system $\mathcal{T}_\mathcal{A}$, with a strict alternation of delay moves (though possibly 0-delay moves) and discrete moves. In the following, we may write a run $\varrho = s \xrightarrow{t_1} s'_1 \xrightarrow{e_1} s_1 \xrightarrow{t_2} s'_2 \xrightarrow{e_2} s_2 \dots$ more compactly as $\varrho = s \xrightarrow{t_1, e_1} s_1 \xrightarrow{t_2, e_2} s_2 \dots$. A transition of the form $s \xrightarrow{t,e} s'$ will be called a *mixed move*. If ϱ is a finite run which ends in some $s = (\ell, v)$ with $\ell \in \mathsf{Goal}$, we say that ϱ is accepting. If $s \in S$ is a configuration, we write $\mathsf{Runs}(\mathcal{A}, s)$ (respectively $\mathsf{Runs}_\mathsf{f}(\mathcal{A}, s)$, $\mathsf{Runs}^\mathsf{acc}_\mathsf{f}(\mathcal{A}, s)$) the set of infinite (respectively finite, finite accepting) runs that start in s.

In the following we will assume timed automata are non-blocking, that is, from every reachable configuration s, there exists some delay t and some edge e, there exists some configuration s' such that $s \xrightarrow{t,e} s'$ is a mixed move of $\mathcal{A}$.

We can now give the semantics of a weighted timed automaton $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{cost})$. The value $\mathsf{cost}(\ell)$ given to location ℓ represents a cost rate, and delaying t time units in a location ℓ will then cost '$t \cdot \mathsf{cost}(\ell)$'. The value $\mathsf{cost}(e)$ given to edge e represents the cost of taking that edge. Formally, the cost of the two types of moves in a weighted timed automaton is defined as follows:[1]

$$\begin{cases} \mathsf{cost}\Big((\ell, v) \xrightarrow{t} (\ell, v + t)\Big) = t \cdot \mathsf{cost}(\ell) \\ \mathsf{cost}\Big((\ell, v) \xrightarrow{e} (\ell', v')\Big) = \mathsf{cost}(e) \end{cases}$$

A *run* ϱ of a weighted timed automaton is a run of the underlying timed automaton, *i.e.*, a finite or infinite sequence of moves in the transition system (with a strict alternation of delay and discrete moves). The cost of ϱ, denoted $\mathsf{cost}(\varrho)$, is the sum of the costs of all the simple moves along ϱ.

Example 3.2. *We consider the weighted timed automaton $\mathcal{A}$ depicted on Figure 4, where we do not distiguish between dashed and plain edges for the moment. When relevant (i.e., when the cost is non-null), we decorate each location with a value (like 5 for location ℓ_0), that*

[1] Note that we overload the notation cost, which designs both the cost assigned to a transition or a location in a weighted timed automaton, and the cost assigned to a move in the transition system. It will also be used to represent the cost of an execution.

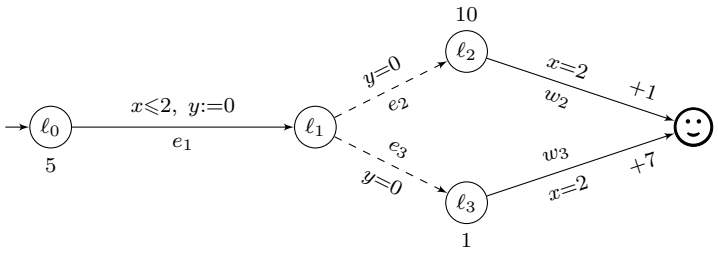


Figure 4: A first small example

represents the cost rate in that location, and we decorate each edge with a value (like +7 for edge w_3), that represents the discrete cost of taking that edge. A possible run in $\mathcal{A}$ is:

$$\varrho \;=\; (\ell_0,0) \xrightarrow{0.1} (\ell_0,0.1) \xrightarrow{e_1} (\ell_1,0.1) \xrightarrow{e_3} (\ell_3,0.1) \xrightarrow{1.9} (\ell_3,2) \xrightarrow{w_3} (\textcircled{\scriptsize\smile},2)$$

The cost of ϱ is $\mathsf{cost}(\varrho) = 5 \cdot 0.1 + 1 \cdot 1.9 + 7 = 9.4$ (the cost per time unit is 5 in ℓ_0, 1 in ℓ_3, and the cost of transition w_3 is 7).

3.2 Optimization problems

Unlike hybrid systems, in weighted timed automata, cost variables do not constrain the behaviours of the system, but are *'observer variables'*: they give a quantitative information on the quality (or performance) of an execution, but cannot impact on the possible executions. Several optimization criteria can then be thought of, like the optimal cost for reaching some goal in the system, or the optimal mean-cost that can be achieved along infinite executions of the system. These optimization problems are relevant for instance in scheduling problems, where the cost evolution can be viewed as resource consumption.

In this subsection we give an overview of the decidability and complexity results for the two optimization problems we have mentioned. In the next subsection we will give a rough idea why these results hold.

3.2.1 The optimal cost problem

Intuitively, the *optimal cost problem* asks what is the optimal cost for reaching the goal locations in a weighted timed automaton. We assume $\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{cost})$ is a weighted timed automaton. The optimal cost for reaching goal locations in $\mathcal{A}$ is defined as:

$$\mathit{opt\_cost}_{\mathcal{A}} = \inf\{\mathsf{cost}(\varrho) \mid \varrho \in \mathsf{Runs}_{\mathsf{f}}^{\mathsf{acc}}(\mathcal{A}, s_0)\}$$

By extension when we will speak of the complexity, we will mean the complexity of the corresponding decision problem, which asks, given a threshold $c \in \mathbb{Q}_{\geqslant 0}$, whether $\mathit{opt\_cost}_{\mathcal{A}} \leqslant c$. If $\varepsilon > 0$, a run $\varrho \in \mathsf{Runs}_{\mathsf{f}}(\mathcal{A}, s_0)$ is an *ε-optimal schedule* in $\mathcal{A}$ if $\mathit{opt\_cost}_{\mathcal{A}} \leqslant \mathsf{cost}(\varrho) \leqslant \mathit{opt\_cost}_{\mathcal{A}} + \varepsilon$.

Example 3.3. *We consider the weighted timed automaton of Example 3.2 (page 5). There are basically two choices that can be made: (i) when edge e_1 is fired, and (ii) go through ℓ_2 or through ℓ_3. Writing t for the value of clock x when e_1 is fired, the accumulated cost along plays of the game is either $5t + 10(2-t) + 1$ (through ℓ_2) or $5t + (2-t) + 7$ (through ℓ_3). The optimal cost is thus $\inf_{t \leqslant 2} \min(5t + 10(2-t) + 1, 5t + (2-t) + 7) = 9$, and the optimal time for firing transition e_1 is when $t = 0$. Then, the best choice is to go through ℓ_3.*

In this context, the first problem which has been solved already in the early nineties is the *optimal time problem*, where the cost represents the time that has elapsed (the cost rates in locations are equal to 1 — they increase at the same speed as the time — and discrete costs of transitions are set to 0): the problem then amounts to computing the optimal time for reaching one of the distinguished goal locations in a timed automaton.

Theorem 3.4 ([33]). *The optimal time in timed automata is computable in exponential time.*

Applying the further results (theorem 3.6) on weighted timed automata, we can refine this result, and computing the optimal time in timed automata can actually be solved in polynomial space. Moreover, we can prove that the corresponding decision problem is indeed PSPACE-complete (if there is an answer to the reachability problem, we can bound the duration of a witness run by an exponential, and then answering positively to the decision problem for that upper bound duration is equivalent to answering the reachability question, which is known to be PSPACE-hard).

Almost ten years after this first result, the general optimal cost optimal problem in weighted timed automata has been formulated and solved independently in [6] and in [9].

Theorem 3.5 ([6, 9]). *The optimal cost in weighted timed automata is computable (in exponential time).*

The algorithm developed in [6] is based on an extension of the classical region automaton, and yields an EXPTIME upper bound, whereas the algorithm developed in [9] is based on well-quasi-orders and gives no good information on the complexity of the problem.

Few years later, the precise complexity of that problem has been settled.

Theorem 3.6 ([13]). *The optimal cost problem in weighted timed automata is* PSPACE*-complete. Furthermore, for every $\varepsilon > 0$, ε-optimal schedules can be computed.*

Remark 3.7. *Note that the above result also holds when the costs of locations on transitions are taken in $\mathbb{Z} = \mathbb{N} \cup -\mathbb{N}$, the set of integers.*

⌟

3.2.2 The optimal mean-cost problem

The *optimal mean-cost problem* asks what is the optimal cost per time unit (mean-cost) that can be achieved (or approximated) in a weighted timed automaton. To define the most general

mean-cost problem, we assume that $\mathcal{A}$ is a weighted timed automaton with **two** cost functions, say cost and reward ($\mathcal{A} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{cost}, \mathsf{reward})$). Then, the optimal mean-cost of $\mathcal{A}$ with respect to cost and reward is formally defined as:

$$opt\_cost^{\omega}_{\mathcal{A}} = \inf\{\mathsf{mean\_cost}(\varrho) \mid \varrho \in \mathsf{Runs}(\mathcal{A}, s_0)\}$$

where $\mathsf{mean\_cost}(\varrho)$ is defined as $\liminf_{n\to+\infty} \frac{\mathsf{cost}(\varrho_n)}{\mathsf{reward}(\varrho_n)}$ (ϱ_n is the prefix of length n of ϱ). We use the 'lim inf' operator because the limit might not be properly defined. A particular case is when the reward corresponds to the time elapsed, in which case the value $\mathsf{mean\_cost}(\varrho)$ is the mean cost per time unit along run ϱ. If $\varepsilon > 0$, a run $\varrho \in \mathsf{Runs}(\mathcal{A}, s_0)$ is an *ε-optimal schedule* in $\mathcal{A}$ if $opt\_cost^{\omega}_{\mathcal{A}} \leqslant \mathsf{mean\_cost}(\varrho) \leqslant opt\_cost^{\omega}_{\mathcal{A}} + \varepsilon$. The following result has been proven:

Theorem 3.8 ([15, 16]). *Under some restrictions for the reward function, the optimal mean-cost problem is* PSPACE*-complete in weighted timed automata. Furthermore, for every $\varepsilon > 0$, ε-optimal schedules can be computed.*

The restrictions mentioned in the above theorem assume the function reward be strictly non-Zeno, *i.e.*, along any cycle of the region automaton, the reward increases by some positive lower-bounded amount. If we consider the time elapsed instead of a general reward-function, this amounts to the classical strongly non-Zeno hypothesis, that is for instance made in [8].

Note that this hypothesis is required to get the above result, as a counter-example to the algorithm has been exhibited, when this hypothesis is not satisfied, see Example 3.9.

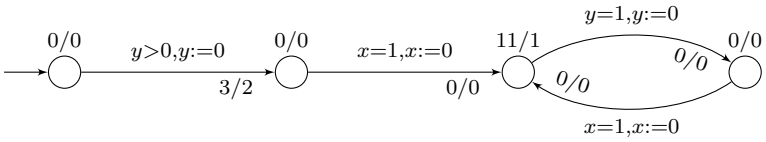


Figure 5: A counter-example (to the algorithm) when the reward is not strictly non-Zeno

Example 3.9. *We consider the weighted timed automaton depicted on Figure 5. We write α/β to indicate the* cost *and* reward *of the locations and the edges. We can notice that the* reward *is not strongly non-Zeno, because of the right-most cycle.*

In this automaton, for every (infinite) run ϱ, $\mathsf{mean\_cost}(\varrho) = +\infty$, whereas the algorithm (based on the corner-point abstraction, see next subsection) computes 2.

3.3 The corner-point abstraction

The two decidability results mentioned in the previous subsection can be proven using a refinement of the classical region abstraction construction [4, 5], which is the basic tool for proving the decidability of many timed models like timed automata. A region is a set of valuations which are time-abstract bisimilar: it means that if v and v' are two valuations belonging to the

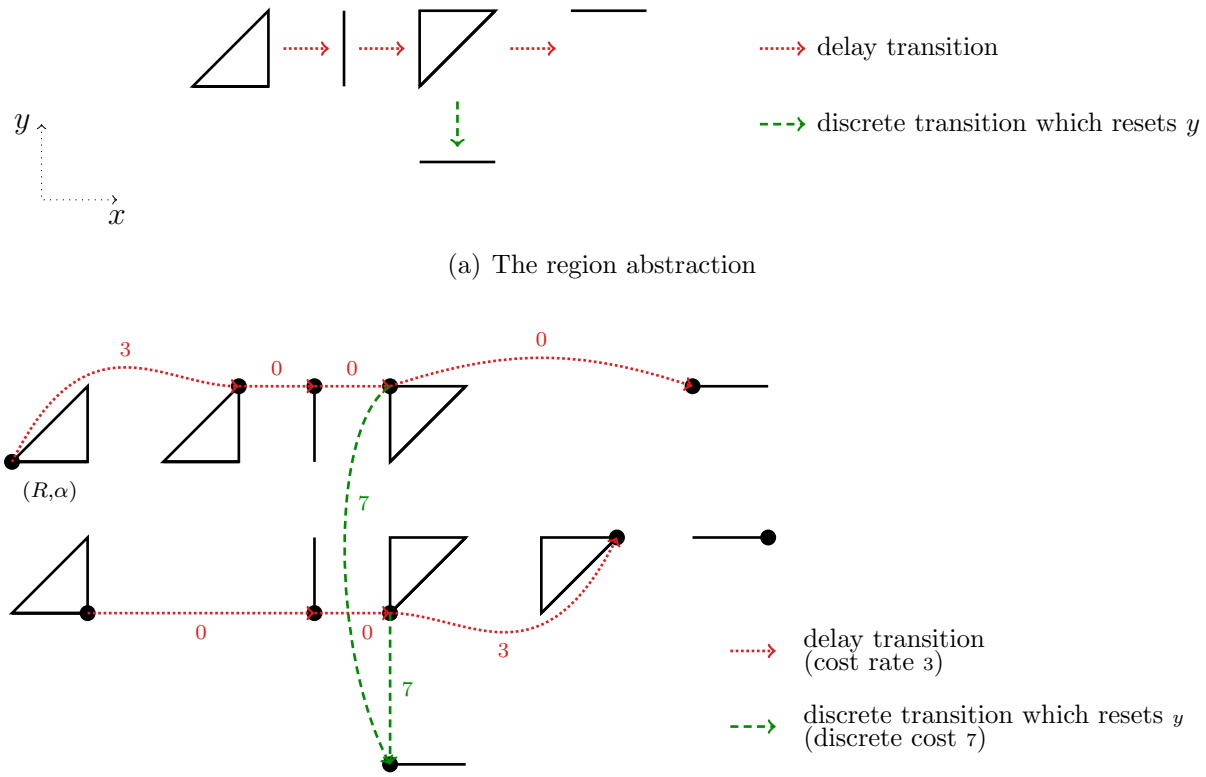
same region, for every location ℓ, similar behaviours will be possible from (ℓ, v) and (ℓ, v'). For the readers not familiar with this construction, we refer to [12, Chap. 2.3] for a presentation of this classical construction, which uses notations and drawings similar to the current notes.

We first notice that regions are not suitable for computing optimal (mean-)costs because costs of region-equivalent trajectories may have pretty different costs. For example, the cost of run ϱ given in Example 3.2 is 9.4 whereas the cost of the (region-equivalent) run delaying 0.9 time units in ℓ_0 and then 2.1 time units in ℓ_3 is 13.6. However we are not interested in computing the costs of all possible runs, but rather to compute extremal (*i.e.*, minimal and/or maximal) cost values. The idea is then to record the cost of moving through extremal points of the regions (those points which have integral coordinates). These points are called *corner-points*, and will annotate regions. We build a graph, called the *corner-point abstraction*, which refines the classical region automaton, and whose states are tuples (ℓ, R, α) where ℓ is a location of the original automaton, R is a region, and α is a corner-point of R. There will be a (delay) transition between (ℓ, R, α) and (ℓ, R', α') either when $R = R'$ and α' is a (strict) successor of α, or when R' is the next successor of R (in the region graph) and $\alpha' = \alpha$ is a corner of both R and R'. There will be a (switch) transition between (ℓ, R, α) and (ℓ', R', α') when (ℓ', R') is the region successor of (ℓ, R) by the reset of the transition, and α' is the image of α by the same reset. Intuitively, being in state (ℓ, R, α) of this graph will mean that we are in location ℓ, in region R, close to the extremal point α; And moving from one state to another through a delay transition means letting time elapse and be close to the designated corners. This construction is illustrated and explained with some more intuition in Example 3.10.

Example 3.10. *We illustrate the notion of corner-points in a two-dimensional clock space. We assume the reader is familiar enough with the classical region construction of [4, 5], or refer to [12, Chap. 2.3] for a description of this classical construction, which uses notations and drawings similar to the current notes. Classical evolution of regions can be schematized as in Figure 6: when time elapses, regions are visited following time successors (the immediate successor of a triangular region is a flat region while the immediate successor of a flat region is a triangular region), and when firing transitions, clocks may be reset, and regions are then somehow projected into regions of smaller dimensions.*

The corner-point abstraction refines the region abstraction and is depicted in Figure 6. Corners decorating regions are indicated with a black bold dot. We consider the top-left-most region (R, α) of the figure decorated with the corner in the bottom. When time elapses, it is transformed into the top corner of the same region which is almost one time unit later: thus, as the cost rate in the current location is supposed to be 3 per time unit, the cost of this move will be set to 3. The next move is to enter the next region (which is flat) but to stay close to the same corner: the cost is thus almost 0 (because almost no time has elapsed), that is why we label the move by 0. And so on. For discrete moves, regions are transformed as usual, and corners are also projected (the projection preserves the property of extremal points of polyhedra). Transitions are then labelled with the cost of the transition (7 in our example).

Given a weighted timed automaton $\mathcal{A}$, we write $\mathsf{CP}(\mathcal{A})$ for its *corner-point abstraction*. The result is a weighted finite graph (whose cost functions will also be denoted cost and reward), in which it will be possible to solve the (mean-)cost optimality problems [32, 41, 50].



(a) The region abstraction

(b) The corner-point abstraction

Figure 6: Region *vs* corner-point abstraction

An important property of this graph is that, given a finite run $\varrho : (\ell_0, v_0) \to (\ell_1, v_1) \to \ldots \to (\ell_n, v_n)$ in $\mathcal{A}$, there exist two finite paths $\pi : (\ell_0, R_0, \alpha_0) \to (\ell_1, R_1, \alpha_1) \to \ldots \to (\ell_n, R_n, \alpha_n)$ and $\pi' : (\ell_0, R_0, \alpha'_0) \to (\ell_1, R_1, \alpha'_1) \to \ldots \to (\ell_n, R_n, \alpha'_n)$ in $\mathsf{CP}(\mathcal{A})$ such that $v_i \in R_i$ for every i, α_i and α'_i are corners of R_i, and $\mathsf{cost}(\pi) \leqslant \mathsf{cost}(\varrho) \leqslant \mathsf{cost}(\pi')$. Conversely, for every finite path $\pi : (\ell_0, R_0, \alpha_0) \to (\ell_1, R_1, \alpha_1) \to \ldots \to (\ell_n, R_n, \alpha_n)$ in $\mathsf{CP}(\mathcal{A})$, for every $\varepsilon > 0$, we can construct a real run $\varrho : (\ell_0, v_0) \to (\ell_1, v_1) \to \ldots \to (\ell_n, v_n)$ in $\mathcal{A}$ such that for every index i, $v_i \in R_i$, and $|\mathsf{cost}(\varrho) - \mathsf{cost}(\pi)| < \varepsilon$.

There is thus a strong relation between finite runs in $\mathcal{A}$ and finite paths in $\mathsf{CP}(\mathcal{A})$. Computing the optimal cost for reaching a given goal in $\mathcal{A}$ reduces to computing the optimal cost for reaching a distinguished set of states in the discrete weighted graph $\mathsf{CP}(\mathcal{A})$.

The case of optimal mean-cost needs some more work, the corner-point abstraction can nonetheless be used to compute it. We first recall that in a finite weighted graph, the optimal mean-cost can be computed as the mean cost of a reachable (simple) cycle that minimizes that value [41] — we call such a cycle an *optimal cycle*. Then, we prove that the mean-cost of an infinite run in $\mathcal{A}$ cannot be any better than the optimal cycle in $\mathsf{CP}(\mathcal{A})$. This can be proven by taking longer and longer prefixes of an infinite run ϱ, and at the limit, the ratio will always be larger than the mean-cost of the optimal cycle in $\mathsf{CP}(\mathcal{A})$. Write ϱ_n for the prefix of length n of ϱ. Applying the previous result on finite runs, we can construct a finite path π_n in $\mathsf{CP}(\mathcal{A})$ such that $\mathsf{cost}(\pi_n) \leqslant \mathsf{cost}(\varrho_n)$. We can decompose π_n into cycles as schematically depicted on Fig-

ure 7. The linear part of π_n is cycle-free, hence has a bounded length, and its cost will somehow

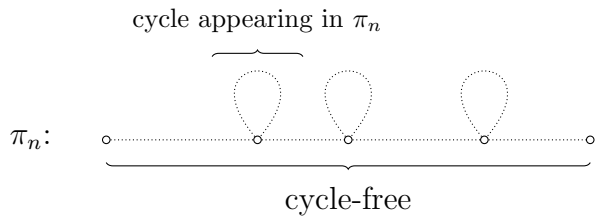


Figure 7: Decomposition of a long path in $\mathsf{CP}(\mathcal{A})$

become negligible when n tends to $+\infty$. The mean-cost of every cycle is no better than the optimal cycle of $\mathsf{CP}(\mathcal{A})$. Hence, at the limit, the mean-cost of ϱ will not be better than the mean-cost of the optimal cycle in $\mathsf{CP}(\mathcal{A})$. Conversely, paths in the corner-point abstraction can be approximated by real runs in the original automaton with costs and rewards that are very close to the one in the corner-point abstraction. The construction is presented in details in [16].

The size of the corner-point abstraction is exponential in the size of the original automaton (a region R has at most $|X|$ corner-points, where X is the set of clocks of the automaton), *i.e.*, as is the size of the region automaton. Using non-determinism, we can guess optimal paths (respectively cycles) in $\mathsf{CP}(\mathcal{A})$, without first computing the full graph. This non-deterministic algorithm uses polynomial space, hence the PSPACE upper bound for the two optimization problems. The PSPACE lower bound can be easily obtained by reduction to the reachability problem in timed automata, for appropriate cost functions.

3.4 Partial conclusion and related work

In this section, we have presented the decidability results for two basic optimization problems on weighted timed automata. This is really encouraging because the theoretical complexity of these problems is the same as standard reachability in timed automata.

In the context of (non-weighted) timed automata, regions are not used in implementations, but a symbolic approach based on zones is preferred and implemented. Similarly, a symbolic approach for the optimal reachability problem based on *priced zones*, an extension of standard zones, has been proposed [43]. The paper [10] reports algorithms and applications of the tool Uppaal-Cora,[2] which is based on this approach. Also, when several cost variables are defined, it is possible to compute Pareto-optimal points [44]. On the contrary, the optimal mean-cost is not implemented yet, since no good data structures have been developed. This is however a very challenging (and non-trivial) line of research.

We should emphasize that the corner-point abstraction is a very interesting abstraction, which has further been used for solving other optimization problems, like the time-discounted cost

[2] `http://www.cs.aau.dk/~behrmann/cora/publications.html`

optimal problem [34] (this extends the classical discounted payoff that we can find in the game theory literature [50]).

Finally, let us notice that even though one can compute the optimal (mean-)cost in weighted timed automata, only almost-optimal schedules can be synthesized. The corner-point abtractions does not allow to compute an optimal schedule, nor to decide that one exists.

4 Optimal reachability in weighted timed games

We have seen the optimal cost and the optimal mean-cost were both computable in weighted timed automata in polynomial space. This is really encouraging to consider more involved problems. In this section, we consider the very similar problems, but no more in the context of closed systems, as in the previous section, but in the context of open systems. An open system somehow models an interaction between the system itself and the environment it is embedded in. As often this is modelled as games [49] and we will use some terminology from game theory.

4.1 Weighted timed games

A *weighted timed game* $\mathcal{G} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{cost})$ is a weighted timed automaton in which edges are decoupled into controllable edges played by the *controller* (set $E_c \subseteq E$) and uncontrollable edges played by the environment (set $E_e \subseteq E$). W.l.o.g. we assume Goal locations are sink locations with cost 0 per time unit, and a loop on each of the locations with cost 0.

A *(controller) strategy* in $\mathcal{G}$ from the initial state $s_0 = (\ell_0, \mathbf{0}_X)$ is a partial function f that associates to a finite run $\varrho \in \mathsf{Runs_f}(\mathcal{G}, s_0)$ a pair $(d, e) \in \mathbb{R}_{\geqslant 0} \times E_c$ such that edge e can be taken after delaying d time units after ϱ; it describes the next move to be done after ϱ. A strategy f is said *memoryless* if for all runs $\varrho, \varrho' \in \mathsf{Runs_f}(\mathcal{G}, s_0)$, $\mathsf{last}(\varrho) = \mathsf{last}(\varrho')$ implies $f(\varrho) = f(\varrho')$. Memoryless strategies are somehow 'simple' strategies that do not take past into account to make the next decision.

A run $\varrho = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \dots$ is compatible with a strategy f whenever for every i, either $(t_{i+1}, e_{i+1}) = f(\varrho_{\leq i})$,[3] or $t_{i+1} \leq t$ and $e_{i+1} \in E_e$ where $f(\varrho_{\leq i}) = (t, e)$. That way, a strategy f defines a set of *(maximal) plays* denoted $\mathsf{plays}_{\mathcal{G}}(f)$. The strategy f is *winning* (for the reachability goal) if all (maximal) plays of $\mathsf{plays}_{\mathcal{G}}(f)$ end up in Goal.

The classical reachability game problem asks, given a timed game $\mathcal{G}$, whether there is a winning controller strategy for the reachability goal. Classical reachability games have been considered in the context of timed systems in the nineties, and deciding those games is EXPTIME-complete [8, 36]. For those games, memoryless region-uniform (that is, the same edge is given by the strategy within a region) strategies are sufficient.

[3] $\varrho_{\leq i}$ is the prefix of length i of ϱ: $\varrho_{\leq i} = s_0 \xrightarrow{t_1,e_1} \dots \xrightarrow{t_i,e_i} s_i$.

In the context of weighted timed games, an optimality criterion can be expressed. The cost of a winning strategy f is defined as:

$$\mathsf{cost}_{\mathcal{G}}(f) = \sup\{\mathsf{cost}(\varrho) \mid \varrho \in \mathsf{plays}_{\mathcal{G}}(f)\}$$

Note that if f is a winning strategy, then for every $\varrho \in \mathsf{plays}_{\mathcal{G}}(f)$, $\mathsf{cost}(\varrho) < +\infty$. However it might be the case that $\mathsf{cost}_{\mathcal{G}}(f) = +\infty$.

The aim of the controller is to optimize this value and we want to compute the optimal cost the controller can ensure, whatever the environment does, which can be formally written as:

$$\mathit{opt\_cost}_{\mathcal{G}} = \inf\{\mathsf{cost}_{\mathcal{G}}(f) \mid f \text{ winning strategy}\}$$

We will consider the two following decision problems:

- the first, called the *bounded cost problem*, asks, given a threshold $c \in \mathbb{Q}_{\geqslant 0}$, whether there is some strategy f such that $\mathsf{cost}_{\mathcal{G}}(f) \leqslant c$;
- the second, called the *optimal cost problem*, asks, given a threshold $c \in \mathbb{Q}_{\geqslant 0}$, whether $\mathit{opt\_cost}_{\mathcal{G}} \leq c$.

We will also be interested in synthesizing *almost-optimal* strategies, that is for every $\varepsilon > 0$, computing a strategy f_ε which is ε-*optimal*: $\mathit{opt\_cost}_{\mathcal{G}} \leqslant \mathsf{cost}_{\mathcal{G}}(f) \leqslant \mathit{opt\_cost}_{\mathcal{G}} + \varepsilon$.

Example 4.1. *We consider the weighted timed automaton of Example 3.2 (page 5). Dashed (respectively plain) arrows are now for uncontrollable (respectively controllable) transitions. Depending on the choice of the environment (going to location ℓ_2 or ℓ_3), the accumulated cost along plays of the game is either $5t + 10(2-t) + 1$ (through ℓ_2) or $5t + (2-t) + 7$ (through ℓ_3) where t is the delay elapsed in location ℓ_0. The optimal cost the controller can ensure is thus $\inf_{t \leqslant 2} \max(5t + 10(2-t) + 1, 5t + (2-t) + 7) = 14 + \frac{1}{3}$, and the optimal time for firing transition e_1 is when $t = \frac{4}{3}$. The controller has an optimal strategy, which consists in waiting in location ℓ_0 until $x = \frac{4}{3}$, and in entering location ℓ_1. Then, the environment chooses to go either to ℓ_2 or to ℓ_3, and finally when the value of x reaches 2, the controller goes to the goal location ☺.*

Remark 4.2. *Let us mention that in the above example, the optimal cost is non-integral, contrary to the case of closed systems. This means in particular that no region-based technology (and even corner-point abstraction) can be used to solve optimal timed games.*

Until recently [21], these two problems were used with no real distinction. However they can interestingly easily be distinguished, as shown in Example 4.3.

Figure 8: Two weighted timed games with optimal cost 1

Example 4.3. *Consider the two weighted timed games depicted on Figure 8. In the game on the left, for every $\varepsilon > 0$, the controller has a strategy to get cost $1 + \varepsilon$. In the game on the right, the controller has a strategy to ensure cost strictly less than 1. Hence, in both cases, the optimal cost is 1, but they generate quite different 'behaviours'.*

4.2 Decidability or undecidability?

In the late nineties, optimal-time timed games (*i.e.*, weighted timed games where cost represents time elapsing) have been considered [7], and the complexity has been made precise rather recently [39] using strategy improvement techniques.

Theorem 4.4 ([7, 39]). *Optimal-time in reachability timed games is computable. The corresponding decision problem is* EXPTIME*-complete.*

The reason is that the region abstraction needs not be refined to compute the optimal time.

Remark 4.5. *Note also that the* EXPTIME *upper bound could have been computed as follows: solve the reachability game classically, and record the corresponding memoryless winning strategy (using for instance a backward algorithm à la [8]), compute the maximal time τ for winning following that memoryless strategy (this needs to be bounded, otherwise it would not be winning), and then add an extra clock z which is never reset but is used in a guard $z \leqslant c$ (for c chosen non-deterministically not larger than τ) which constrains every transition leading to a location in* Goal*. The optimal time is the smallest c for which the transformed game is winning (because thanks to [7] we know that the optimal time is an integer). Finally as the value of τ is at most exponential (because the selected winning strategy is memoryless), this global algorithm only requires exponential time.* ⌟

Then, in [42], optimal timed games (with general costs) are considered, and a doubly-exponential time algorithm is designed for computing optimal cost (and synthesizing (almost-)optimal strategies) in *acyclic* timed games. The algorithm somehow extends classical min/max-algorithms for discrete games to timed games.

In [1], the 2EXPTIME upper bound mentioned above is improved to an EXPTIME upper bound. Note that this algorithm computes for every winning state the optimal cost for winning and provides a (possibly almost) optimal winning strategy. The algorithm which is proposed splits the state-space into polyhedra on which (roughly) optimal winning strategies are uniform, it is pretty involved, and relies on nice geometrical properties of the state-space. Moreover, a family of weighted timed games is given, for which it is unavoidable to split the set of winning states into an exponential number of pieces.

Theorem 4.6 ([42, 1]). *Optimal cost can be computed in* EXPTIME *in* acyclic *weighted timed games. Furthermore, almost-optimal winning strategies can be computed.*

As a consequence of the above results, and independently shown in [17] using different technics:

Theorem 4.7 ([1, 17]). *Under some restrictions for the cost function, the optimal cost and almost-optimal winning strategies can be computed in weighted timed automata.*

The restriction made in the above result is quite strong. It says that the cost needs to be *strongly non-Zeno*: there is a constant $\kappa > 0$ such that every run that is read over a cycle of the region automaton has cost larger than κ. That means that the longer is a play, the larger will be its cost; more precisely, if M is a bound on the cost of a given winning strategy (that we can choose memoryless and region-uniform), then we can unfold the game up to a depth which ensures that all runs will have cost larger than M, and solve this uncomplete unfolded game; this will preserve the optimal cost.

The first undecidability result has come as a surprise in [30]! It requires weighted timed games with five clocks. This result has been improved a bit later using a new encoding requiring only three clocks [14].

Theorem 4.8 ([30, 14]). *The bounded cost problem for weighted timed games with three clocks or more is undecidable.*

Note that, formally, this result does not speak of the optimal cost, but only of the existence of a strategy whose cost is bounded by some constant (which is the bounded cost problem). This is some intriguing discrepancy with all known decidability results, which speak of the optimal cost problem. It has taken almost ten years for finally transferring this undecidability to the optimal cost problem.

Theorem 4.9 ([21]). *The optimal cost problem for weighted timed games with four clocks or more is undecidable.*

In Subsection 4.3, we will describe the undecidability proof of [14], which will help understand why it is so hard to analyze weighted timed games.

Is that the end of the story?

The previous undecidability result has settled the status of the optimal reachability problem in arbitrary weighted timed games, and has launched a quest for decidable subclasses of weighted timed games. The first result in that direction is the following:

Theorem 4.10 ([30]). *The optimal cost in single-clock stopwatch timed games*[4] *is computable.*

In the restricted case mentioned in the above theorem, the semi-algorithm proposed in [17] terminates, because roughly, classical regions never need to be split and are thus correct.

[4]We recall that a stopwatch timed game is a weighted timed game where the cost is stopwatch, that is, can have rates 0 or 1.

Then, optimal cost in weighted timed games with one clock (but arbitrary cost) has been proven computable [25] (though in a restricted *turn-based* framework where locations are either controllable — *i.e.* all transitions leaving this location are controllable — or uncontrollable). The high complexity of the algorithm of [25] has later been improved in [48, 35], and a special subclass has been exhibited, in which optimal cost can be computed in PTIME. Technics used in these papers make either use of structural properties of the game (like in [25]) or of value iteration technics (like [35]).

Theorem 4.11 ([25, 48, 35, 28]). *The optimal cost in turn-based single-clock weighted timed games is computable in* EXPTIME *(*PTIME *if only two rates among* $\{0, -d, d\}$ *for some* d*). Note that the corresponding decision problem is* PTIME*-hard. Furthermore, for every* $\varepsilon > 0$*, we can compute* ε*-optimal and memoryless strategies.*

Another way to get around the undecidability results is to relax on the precision of the computation. A recent result [21] builds on that idea, and proposes an approximation algorithm for the optimal cost and for winning strategies. We believe that this is an interesting research direction: indeed, in all decidability results that have been proven so far, even when the optimal cost can be computed, only almost-optimal strategies (or schedules, in the case of weighted timed automata) can be synthesized. Hence, it seems that it is sufficient to compute an (arbitrary) approximate value of the optimal cost. More precisely, the result can be stated as follows:

Theorem 4.12 ([21]). *Under some restrictions over the cost function, an arbitrary approximation of the optimal cost can be computed in weighted timed games. Furthermore, almost-optimal strategies can be computed as well.*

We discuss now the restrictions mentioned in the theorem. They assume that the cost function satisfies the following condition: there exists some positive $\kappa > 0$ such that, if ρ is a run of the underlying timed automaton which is read over a cycle of the region automaton, then:

(a) either $\mathsf{cost}(\rho) = 0$;
(b) or $\mathsf{cost}(\rho) \geq \kappa$.[5]

This restriction relaxes the strictly non-Zeno hypothesis made in Theorem 4.7, where all runs are required to satisfy condition (b). We should then insist on two things:

- First, as stated in Theorem 4.7, if (b) is always satisfied, the optimal cost can be computed;
- Then, the undecidability proofs for Theorems 4.8 and 4.9 (presented in Subsection 4.3) only build games that satisfy the restriction.

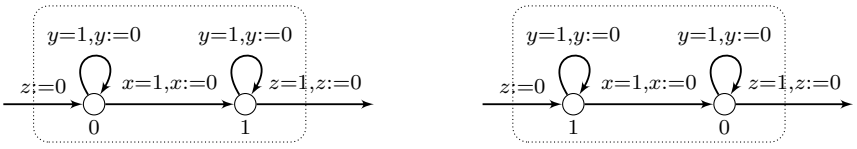
The complexity of the approximation is unfortunately not so good ($2\exp(|\mathcal{G}|) \cdot \left(1/\varepsilon\right)^{|X|}$, where ε is the required precision). However the scheme might probably be improved to be turned into a reasonably efficient algorithm. This is left as an open problem.

We will now give more details for the undecidability results, and for the approximation scheme.

[5] Similarly to the strongly non-Zeno hypothesis, we can take w.l.o.g. $\kappa = 1$.

4.3 A glimpse of the undecidability proof

We will present the basic ideas of the undecidability proof proposed in [14] for Theorem 4.8, which we think is quite instructive. First we consider the two small modules that are depicted on Figure 9. The module $\mathsf{Add}_z^{+x}(x,y)$ (respectively $\mathsf{Add}_z^{+(1-x)}(x,y)$) uses z as an extra clock, lets the values of x and y at the end of the module be the same as at the entry of the module, increases the cost by x_0 (respectively $1-x_0$) if x_0 is the value of x when entering the module.



(a) Module $\mathsf{Add}_z^{+x}(x,y)$ (b) Module $\mathsf{Add}_z^{+(1-x)}(x,y)$

Figure 9: Two interesting modules

Concatenating these modules, one can implement various cost functions (non-negative linear combinations of x_0, y_0, $1-x_0$, $1-y_0$ and 1). In particular, one can implement the two cost functions cost_1 and cost_2 defined as follows:

$$\mathsf{cost}_1(x_0,y_0) = 2x_0 + (1-y_0) + 2 \qquad \mathsf{cost}_2(x_0,y_0) = 2(1-x_0) + y_0 + 1$$

Now, it is easy to check that $2x_0 > y_0$ implies $\mathsf{cost}_1(x_0,y_0) > 3$, whereas $2x_0 < y_0$ implies $\mathsf{cost}_2(x_0,y_0) > 3$. Moreover, if $2x_0 = y_0$, then $\mathsf{cost}_1(x_0,y_0) = \mathsf{cost}_2(x_0,y_0) = 3$. Hence if we are in a state with $x = x_0$ and $y = y_0$, and if the choice of the cost function is given to the environment, it can enforce a cost (strictly) larger than 3 if and only if $2x_0 \neq y_0$. Otherwise, the cost will be 3, whatever is the choice of the environment. This will later serve as a module to check whether twice the value of x is equal to the value of y. We denote this test module $\mathsf{Test}_z(2x = y)$, with the subscript z to indicate that an extra clock z is used in the module.

To simulate a two-counter machine, the idea is to store the value of a counter c into a clock, whose value will be, at distinguished points in time, $\frac{1}{2^c}$. Hence, to store the values of two counters, one needs two clocks. Assume an instruction increments the first counter, and lets the second counter unchanged. Assume furthermore that the value of the first counter is c and is stored in clock x, whereas the value of the second counter is d and is stored in clock y. We consider the module depicted on Figure 10, which will simulate the above instruction (the value of the first counter is initially stored in clock x and finally in clock z).

The duration of an execution in that module is one time unit (condition checked by the extra clock u). It is not difficult to check that the final values for x and y correspond to their initial values. The final value for z has been non-deterministically guessed during the execution, so can be anything within the interval $[0,1]$, say α. An uncontrollable transition leads to the test module $\mathsf{Test}_y(x = 2z)$ that we have described earlier. If (and only if) the guess for z has been correct (or equivalently $2\alpha = \frac{1}{2^c}$) the environment has no strategy to get a cost value larger than 3.

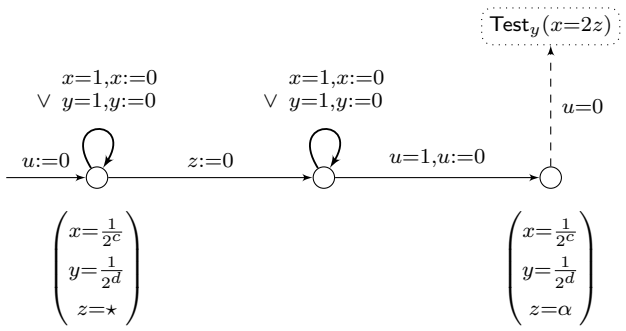


Figure 10: Simulation of the instruction which increments c and lets d unchanged

There is no cost labelling locations of the main game, we only add a discrete cost of $+3$ (or three time units with cost-rate 1) when reaching the halting state. In that reduction:

the two-counter machine halts if, and only if,
the controller has a winning strategy with cost no more than 3
in the weighted timed game.

It is worth noticing that the described reduction uses four clocks, and not three, as claimed. However, we can get rid of clock u using the following trick: the value of the second counter d is now stored by the value $\frac{1}{3^d}$ (note that the choice of $\frac{1}{2^c}$ and $\frac{1}{3^d}$ is arbitrary, it could be $\frac{1}{p^c}$ and $\frac{1}{q^d}$ for p and q relatively prime integers). Indeed we can prove that the constraint $u = 1$ at the end of the module can be replaced by the constraints that the value of x is a negative power of 2 and the value of y is a negative power of 3. Testing that the value of x is a negative power of 2 can be done by iteratively multiplying the value of x by 2 (done using the $\mathsf{Test}_y(z = 2x)$ module) and eventually reaching 1. Finally the constraint that the last location of the module be transient is done by adding a positive cost to that location, and requiring the controller to have a strategy with cost no more than 3.

We can finally notice that the cost in this constructed game is stopwatch (there is no discrete cost, and all cost rates are 0 or 1).

4.4 A glimpse of the approximation scheme

We quickly describe in this subsection the approximation scheme used in Theorem 4.12. We fix a weighted timed game $\mathcal{G} = (X, L, \ell_0, \mathsf{Goal}, E, \mathsf{cost})$, and we split it along regions.[6] The *kernel* $\mathcal{K}$ of $\mathcal{G}$ is the part in which all runs have cost 0: it is made of locations with cost-rate 0, and edges with cost 0. The idea is that sub-runs in $\mathcal{K}$ do not impact on the global cost of the execution, but it does impact on the clock values; on the other hand, sub-runs outside of the kernel have an important impact on the cost of the execution, hence they cannot be too long.

Using this intuition, we build a *semi-unfolding* of the game from the initial location: the game

[6]For every region r, for every transition $\ell \xrightarrow{g,Y} \ell'$, we add transitions $(\ell, r) \xrightarrow{g \wedge \overrightarrow{r}, Y} (\ell', r')$ for every time-successor region $\overrightarrow{r}$ of r, and $r' = [Y \leftarrow 0]\overrightarrow{r}$.

is unfolded, and once the kernel is entered, a (folded) copy of the kernel is plugged; the game is unfolded again from the output edges of the kernel. We stop unfolding when the depth of this semi-unfolding is $N = (M + 2) \cdot |\mathcal{R}(\mathcal{G})|$, where $|\mathcal{R}(\mathcal{G})|$ is the size of the region automaton of $\mathcal{G}$ and M is an upper bound on $opt\_cost_{\mathcal{G}}$.[7] This enforces that all runs from the root to a leaf has a cost larger than what should be generated by an (almost-)optimal strategy. Hence the optimal cost in the original game and in the semi-unfolding coincide. The construction is illustrated on Figure 11.

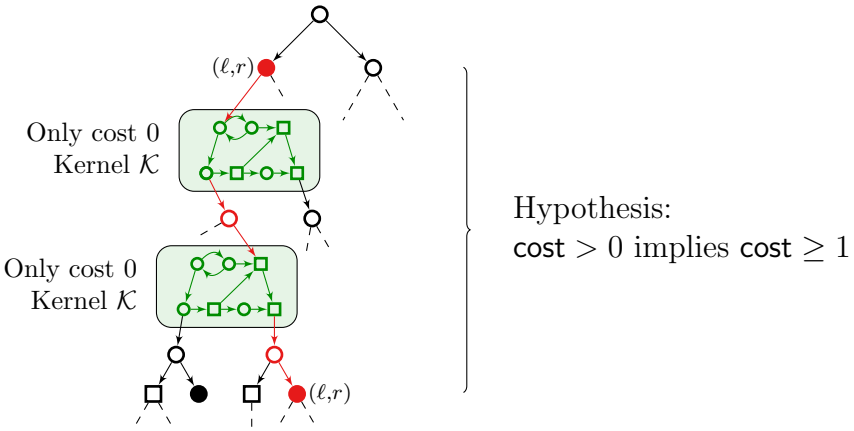


Figure 11: Semi-unfolding of game $\mathcal{G}$

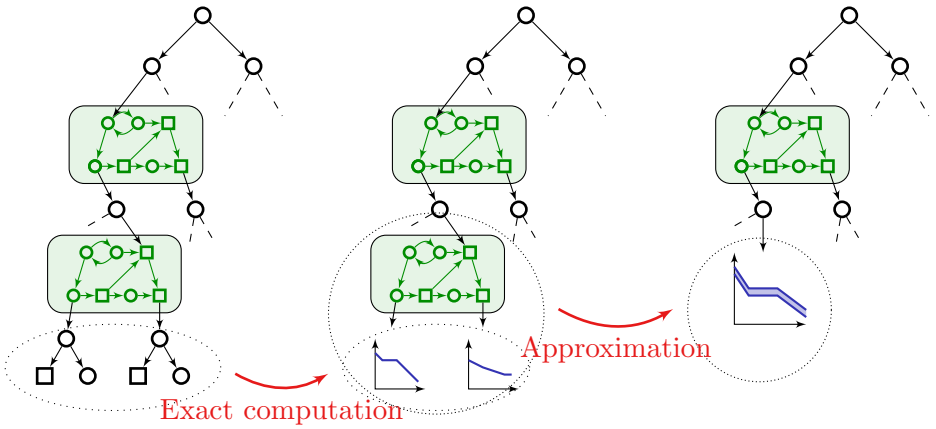


Figure 12: Approximation scheme

The approximation scheme then is as follows: an exact min-max algorithm is applied in tree-like parts of the semi-unfolding; an approximation algorithm is applied in the various copies of the kernel. This is schematized in Figure 12. We now give more details on each of these steps:

(i) *In the tree-like parts of the semi-unfolding.* The computation of the optimal cost can be done using the min-max algorithm of [42] or its refinement of [1].

[7] Such a bound can be precomputed by selecting a memoryless region-based winning strategy and by computing its cost.

(ii) *In the kernels.* A kernel surprisingly generates complex behaviours, even though cost cannot increase, and in particular the optimal cost cannot be computed. We therefore assume we have computed cost functions at the output edges, which correspond to approximations of the optimal cost one can achieve from those edges. Those functions are smooth enough to be under- and over-approximated by piecewise-constant functions, which are constant over a refinement of the standard set of regions (that is, regions obtained with a smaller granularity). Given such a piecewise constant function (which is an under- or an over-approximation), the game played in a kernel becomes a standard (non-weighted, since cost is 0 everywhere) timed game with an extended reachability winning condition: the preference order over output edges is given by the piecewise-constant function (the smaller, the better). Applying results on standard timed games, we easily get that those games can be solved, based on the given refinement of the regions. Hence, a piecewise-constant (under- or over-approximated) cost function can be computed at each entry of the current copy of the kernel.

Note that (arbitrary) almost-optimal winning strategies can be computed in parallel with an approximation of the optimal cost.

Remark 4.13. *We would like to point out that, in the games used in the undecidability proofs, there is a single kernel, which corresponds to the simulation of the two counter machine (that is, before leaving to a test gadget). And the piecewise-constant approximations (roughly) correspond to bounding the counters (that is, we can then not distinguish between large values of the counters).*

4.5 Partial conclusion and remarks

In this section, we have presented the problem of optimal timed games, where the aim of the controller is to optimize the cost for reaching some designated set of goal locations, whatever the environment does. The general problem is unfortunately undecidable, and only restricted classes of systems yield decidability.

We believe an important new insight has been given by an approximation scheme for computing arbitrary approximations of the optimal cost and of corresponding winning strategies. Having in hand these approximations is probably enough in practice. Also, even when the optimal cost can be computed, there is no algorithm to compute an optimal winning strategy, hence an approximation is sufficient.

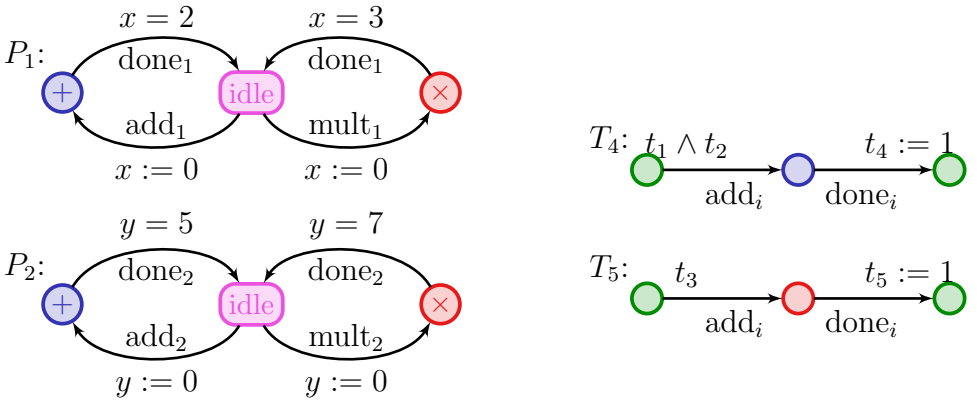
Current work includes investigating further the approximation scheme, and developing a symbolic algorithm that could be used in practice. One would also like to extend the scheme to the whole class of weighted timed games (with no restriction on the cost).

Finally, notice that average-time and mean-payoff objectives have been considered as well in the context of timed games, and while average-time optimal strategies can be computed [40], mean-cost optimal strategies cannot be computed, but interesting subclasses can be exhibited [27].

5 Back to the task graph scheduling example

We come back to the example we have described in Section 2.

A first model of the system, which ignores energy consumption is given by the (natural) product of the timed automata depicted on Figure 13. Each processor has three locations, which indicate whether the processor is idle, or is doing an addition or a multiplication. Timing constraints for the operations are given by clock constraints. The t_i's are Boolean variables that are initially set to 0, and when a task is finished, it is set to 1 (or true): task T_4 requires tasks T_1 and T_2 to be finished before it can start, hence the Boolean constraint '$t_1 \wedge t_2$' when starting Task T_4. Automata synchronize on labels in a standard way. On this model, one can ask whether there is a schedule that manages to compute the arithmetical expression, and exhibit one. One can also ask what is the time-optimal schedule. This is actually the second schedule of Figure 3.



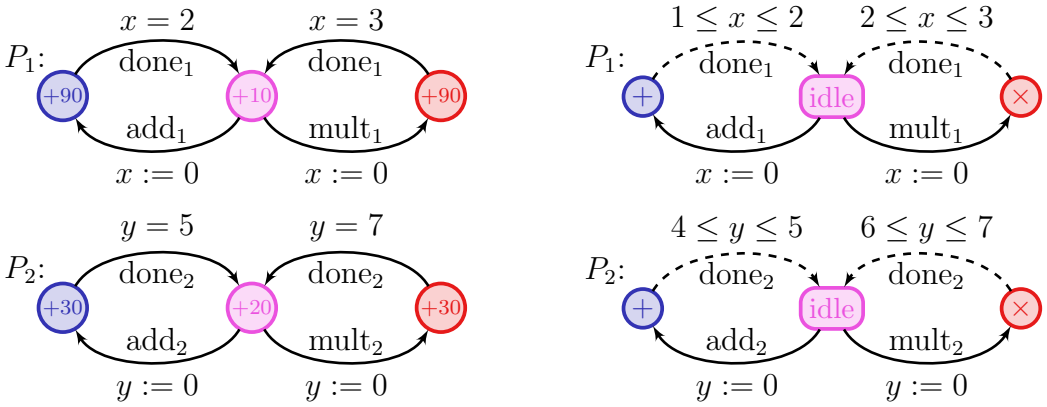
(a) A first model of the two processors

(b) Models for the tasks T_4 and T_5

Figure 13: A first model of the system as a timed automaton

One can refine the models of the processors by adding energy consumption information in the model, as is done on Figure 14(a): locations are decorated using the nominal information of the processors. The global product is now a weighted timed automaton, and we can find energy-optimal schedules using this model. The third schedule on Figure 3 is energy-optimal.

Another refined model of the processors is given on Figure 14(b). In this model, we assume the timing information is not precise, and an operation that takes 2 picoseconds can actually be performed within a delay of $\delta \in [1, 2]$ time units. This is modelled using uncontrollable edges (the controller cannot decide how long it will take to do the operation, but should adapt to any delay). The global product is now a timed game, and one looks for winning strategies, and even optimal winning strategies if one mixes the two last models.



(a) As a weighted timed automaton (b) As a timed game

Figure 14: Two refined models of the processors

6 Conclusion

Weighted timed automata and games have been extensively studied in the past fifteen years. We have given here an overview of results which have been obtained on the optimal reachability problem. While this problem can be solved reasonably efficiently for automata, it is undecidable in the case of weighted timed games. Some restricted classes of games have been described, for which the optimal reachability problem can be solved. More importantly (we believe), a large class of games has been described, for which an arbitrary approximation of the optimal cost can be computed. We believe this is a direction of research which should be investigated further.

There are some other problems that have been studied on the model of weighted timed automata and games. Few years back, temporal logics have been extended with cost constraints, yielding the logics WCTL [29, 22, 23] and WMTL [26], but results are mostly negative. We will not expand on those works here.

Another line of research extends the original models by allowing costs that can be negative or positive. Main challenges are now to synthesize schedules or strategies that will ensure indefinite safe operation with the additional guarantee that energy will always be available, yet never exceeds a possible maximum storage capacity. This energy management problem has been studied in [20, 18, 24], yielding various decidability and undecidability results. We do not expand either in these notes.

Finally we point out another (less recent) survey on the topic of these notes, see [19].

Acknowledgements

This work is supported by ERC Project EQualIS (FP7-308087). I would like to thank all my co-authors, who have worked with me on the model of weighted timed automata and games, and especially Nicolas Markey for our longstanding collaboration on that subject.

References

[1] R. ALUR, M. BERNADSKY, P. MADHUSUDAN, Optimal Reachability in Weighted Timed Games. In: *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*. Lecture Notes in Computer Science 3142, Springer, 2004, 122–133.

[2] R. ALUR, C. COURCOUBETIS, N. HALBWACHS, TH. A. HENZINGER, P.-H. HO, X. NICOLLIN, A. OLIVERO, J. SIFAKIS, S. YOVINE, The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science* 138 (1995) 1, 3–34.

[3] R. ALUR, C. COURCOUBETIS, TH. A. HENZINGER, P.-H. HO, Hybrid Automata: an Algorithmic Approach to Specification and Verification of Hybrid Systems. In: *Proc. Workshop on Hybrid Systems (1991 & 1992)*. Lecture Notes in Computer Science 736, Springer, 1993, 209–229.

[4] R. ALUR, D. L. DILL, Automata for Modeling Real-Time Systems. In: *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*. Lecture Notes in Computer Science 443, Springer, 1990, 322–335.

[5] R. ALUR, D. L. DILL, A Theory of Timed Automata. *Theoretical Computer Science* 126 (1994) 2, 183–235.

[6] R. ALUR, S. LA TORRE, G. J. PAPPAS, Optimal Paths in Weighted Timed Automata. In: *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*. Lecture Notes in Computer Science 2034, Springer, 2001, 49–62.

[7] E. ASARIN, O. MALER, As Soon as Possible: Time Optimal Control for Timed Automata. In: *Proc. 2nd International Workshop on Hybrid Systems: Computation and Control (HSCC'99)*. Lecture Notes in Computer Science 1569, Springer, 1999, 19–30.

[8] E. ASARIN, O. MALER, A. PNUELI, J. SIFAKIS, Controller Synthesis for Timed Automata. In: *Proc. IFAC Symposium on System Structure and Control*. Elsevier Science, 1998, 469–474.

[9] G. BEHRMANN, A. FEHNKER, TH. HUNE, K. G. LARSEN, P. PETTERSSON, J. ROMIJN, F. VAANDRAGER, Minimum-Cost Reachability for Priced Timed Automata. In: *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*. Lecture Notes in Computer Science 2034, Springer, 2001, 147–161.

[10] G. BEHRMANN, K. G. LARSEN, J. I. RASMUSSEN, Priced Timed Automata: Decidability Results, Algorithms, and Applications. In: *Proc. 3rd International Symposium on Formal Methods for Components and Objects (FMCO'04)*. Lecture Notes in Computer Science 3657, Springer, 2004, 162–186.

[11] G. BEHRMANN, K. G. LARSEN, J. I. RASMUSSEN, Optimal Scheduling using Priced Timed Automata. *ACM Sigmetrics Performancs Evaluation Review* 32 (2005) 4, 34–40.

[12] P. BOUYER, *From Qualitative to Quantitative Analysis of Timed Systems*. Ph.D. thesis, Université Paris Diderot, France, 2009.

[13] P. BOUYER, TH. BRIHAYE, V. BRUYÈRE, J.-F. RASKIN, On the Optimal Reachability Problem. *Formal Methods in System Design* 31 (2007) 2, 135–175.

[14] P. BOUYER, TH. BRIHAYE, N. MARKEY, Improved Undecidability Results on Weighted Timed Automata. *Information Processing Letters* 98 (2006) 5, 188–194.

[15] P. BOUYER, E. BRINKSMA, K. G. LARSEN, Staying Alive as Cheaply as Possible. In: *Proc. 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*. Lecture Notes in Computer Science 2993, Springer, 2004, 203–218.

[16] P. BOUYER, E. BRINKSMA, K. G. LARSEN, Optimal Infinite Scheduling for Multi-Priced Timed Automata. *Formal Methods in System Design* 32 (2008) 1, 2–23.

[17] P. BOUYER, F. CASSEZ, E. FLEURY, K. G. LARSEN, Optimal Strategies in Priced Timed Game Automata. In: *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*. Lecture Notes in Computer Science 3328, Springer, 2004, 148–160.

[18] P. BOUYER, U. FAHRENBERG, K. G. LARSEN, N. MARKEY, Timed Automata with Observers under Energy Constraints. In: *Proc. 13th International Conference on Hybrid Systems: Computation and Control (HSCC'10)*. ACM Press, 2010, 61–70.

[19] P. BOUYER, U. FAHRENBERG, K. G. LARSEN, N. MARKEY, Quantitative analysis of real-time systems using priced timed automata. *Communication of the ACM* 54 (2011) 9, 78–87.

[20] P. BOUYER, U. FAHRENBERG, K. G. LARSEN, N. MARKEY, J. SRBA, Infinite Runs in Weighted Timed Automata with Energy Constraints. In: *Proc. 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*. Lecture Notes in Computer Science, Springer, 2008, 33–47.

[21] P. BOUYER, S. JAZIRI, N. MARKEY, On the Value Problem in Weighted Timed Games. In: *Proc. 26th International Conference on Concurrency Theory (CONCUR'15)*. LIPIcs, Leibniz-Zentrum für Informatik, 2015. To appear.

[22] P. BOUYER, K. G. LARSEN, N. MARKEY, Model-Checking One-Clock Priced Timed Automata. In: *Proc. 10th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'07)*. Lecture Notes in Computer Science 4423, Springer, 2007, 108–122.

[23] P. BOUYER, K. G. LARSEN, N. MARKEY, Model Checking One-clock Priced Timed Automata. *Logical Methods in Computer Science* 4 (2008) 2:9.

[24] P. BOUYER, K. G. LARSEN, N. MARKEY, Lower-Bound Constrained Runs in Weighted Timed Automata. In: *Proc. 9th International Conference on Quantitative Evaluation of Systems (QEST'12)*. IEEE Computer Society Press, 2012, 128–137.

[25] P. BOUYER, K. G. LARSEN, N. MARKEY, J. I. RASMUSSEN, Almost Optimal Strategies in One-Clock Priced Timed Automata. In: *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*. Lecture Notes in Computer Science 4337, Springer, 2006, 345–356.

[26] P. BOUYER, N. MARKEY, Costs are Expensive! In: *Proc. 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'07)*. Lecture Notes in Computer Science 4763, Springer, 2007, 53–68.

[27] R. BRENGUIER, F. CASSEZ, J.-F. RASKIN, Energy and mean-payoff timed games. In: *Proc. 17th International Conference on Hybrid Systems: Computation and Control (HSCC'14)*. ACM, 2014, 283–292.

[28] T. BRIHAYE, G. GEERAERTS, S. N. KRISHNA, L. MANASA, B. MONMEGE, A. TRIVEDI, Adding Negative Prices to Priced Timed Games. In: *Proc. 25th International Conference on Concurrency Theory (CONCUR'14)*. Lecture Notes in Computer Science 8704, Springer, 2014, 560–575.

[29] TH. BRIHAYE, V. BRUYÈRE, J.-F. RASKIN, Model-Checking for Weighted Timed Automata. In: *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*. Lecture Notes in Computer Science 3253, Springer, 2004, 277–292.

[30] TH. BRIHAYE, V. BRUYÈRE, J.-F. RASKIN, On Optimal Timed Strategies. In: *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*. Lecture Notes in Computer Science 3821, Springer, 2005, 49–64.

[31] E. M. CLARKE, E. A. EMERSEN, Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Science of Computer Programming* 2 (1982) 3, 241–266.

[32] TH. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[33] C. COURCOUBETIS, M. YANNAKAKIS, Minimum and Maximum Delay Problems in Real-Time Systems. *Formal Methods in System Design* 1 (1992) 4, 385–415.

[34] U. FAHRENBERG, K. G. LARSEN, Discounting in Time. In: *Proc. 10th International Workshop on Verification of Infinite-State Systems (INFINITY'08)*. Electronic Notes in Theoretical Computer Science 253(3), 2009, 25–31.

[35] T. D. HANSEN, R. IBSEN-JENSEN, P. B. MILTERSEN, A Faster Algorithm for Solving One-Clock Priced Timed Games. In: *Proc. 24th International Conference on Concurrency Theory (CONCUR'13)*. Lecture Notes in Computer Science 8052, Springer, 2013, 531–545.

[36] TH. A. HENZINGER, P. W. KOPKE, Discrete-Time Control for Rectangular Hybrid Automata. *Theoretical Computer Science* 221 (1999), 369–392.

[37] TH. A. HENZINGER, P. W. KOPKE, A. PURI, P. VARAIYA, What's Decidable about Hybrid Automata? In: *Proc. 27th Annual ACM Symposium on the Theory of Computing (STOC'95)*. ACM, 1995, 373–382.

[38] TH. A. HENZINGER, P. W. KOPKE, A. PURI, P. VARAIYA, What's Decidable about Hybrid Automata? *Journal of Computer and System Sciences* 57 (1998) 1, 94–124.

[39] M. JURDZIŃSKI, A. TRIVEDI, Reachability-Time Games on Timed Automata. In: *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*. Lecture Notes in Computer Science 4596, Springer, 2007, 838–849.

[40] M. JURDZIŃSKI, A. TRIVEDI, Average-Time Games. In: *Proc. 28th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'08)*. LIPIcs, Leibniz-Zentrum für Informatik, 2008, 340–351.

[41] R. M. KARP, A Characterization of the Minimum Mean-Cycle in a Digraph. *Discrete Mathematics* 23 (1978) 3, 309–311.

[42] S. LA TORRE, S. MUKHOPADHYAY, A. MURANO, Optimal-Reachability and Control for Acyclic Weighted Timed Automata. In: *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*. IFIP Conference Proceedings 223, Kluwer, 2002, 485–497.

[43] K. G. LARSEN, G. BEHRMANN, E. BRINKSMA, A. FEHNKER, TH. HUNE, P. PETTERSSON, J. ROMIJN, As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata. In: *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*. Lecture Notes in Computer Science 2102, Springer, 2001, 493–505.

[44] K. G. LARSEN, J. I. RASMUSSEN, Optimal Conditional Scheduling for Multi-Priced Timed Automata. In: *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*. Lecture Notes in Computer Science 3441, Springer, 2005, 234–249.

[45] A. PNUELI, The Temporal Logic of Programs. In: *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*. IEEE Computer Society Press, 1977, 46–57.

[46] J.-P. QUEILLE, J. SIFAKIS, Specification and Verification of Concurrent Systems in Cesar. In: *Proc. 5th International Symposium on Programming*. Lecture Notes in Computer Science 137, Springer, 1982, 337–351.

[47] J.-F. RASKIN, *An Introduction to Hybrid Automata*, chapter Handbook of Networked and Embedded Control Systems. Springer, 2005, 491–518.

[48] M. RUTKOWSKI, Two-Player Reachability-Price Games on Single-Clock Timed Automata. In: *Proc. 9th Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*. Electronic Notes in Theoretical Computer Science 57, 2011, 31–46.

[49] W. THOMAS, Infinite Games and Verification. In: *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*. Lecture Notes in Computer Science 2404, Springer, 2002, 58–64. Invited Tutorial.

[50] U. ZWICK, M. PATERSON, The Complexity of Mean Payoff Games on Graphs. *Theoretical Computer Science* 158 (1996) 1–2, 343–359.