Controllers with Minimal Observation Power (Application to Timed Systems)*

Peter Bulychev[1], Franck Cassez[2], Alexandre David[1], Kim G. Larsen[1], Jean-François Raskin[3], Pierre-Alain Reynier[4]

[1] CISS, CS, Aalborg University, Denmark
{pbulychev,adavid,kgl}@cs.aau.dk
[2] National ICT Australia, Sydney, Australia
Franck.Cassez@nicta.com.au
[3] Computer Science Department, Université Libre de Bruxelles (U.L.B.), Belgium
jraskin@ulb.ac.be
[4] LIF, Aix-Marseille University & CNRS, France
pierre-alain.reynier@lif.univ-mrs.fr

Abstract. We consider the problem of controller synthesis under imperfect information in a setting where there is a set of available observable predicates equipped with a cost function. The problem that we address is the computation of a subset of predicates sufficient for control and whose cost is minimal. Our solution avoids a full exploration of all possible subsets of predicates and reuses some information between different iterations. We apply our approach to timed systems. We have developed a tool prototype and analyze the performance of our optimization algorithm on two case studies.

1 Introduction

Timed automata by Alur and Dill [2] is one of the most popular formalism for the modeling of real-time systems. One of the applications of Timed Automata is *controller synthesis*, i.e. the automatic synthesis of a controller strategy that forces a system to satisfy a given specification. For timed systems, the controller synthesis problem has been first solved in [18] and progress on the algorithm obtained in [9] has made possible the application on examples of a practical interest. This algorithm has been implemented in the Uppaal-Tiga tool [3], and applied to several case studies [1, 10, 11, 20].

The algorithm of [9] assumes that the controller has *perfect information* about the evolution of the system during its execution. However, in practice, it is common that the controller acquires information about the state of the system via a finite set of sensors each of them having only a finite precision. This motivates to study *imperfect information* games.

* Partly supported by ANR project ECSPER (JC09- 472677), ERC Starting Grant inVEST-279499, Danish-Chinese Center for Cyber Physical Systems (IDEA4CPS) and VKR Center of Excellence MT-LAB.

The first theoretical results on *imperfect information* games have been obtained in [22], followed by algorithmic progresses and additional theoretical results in [21], as well as application to timed games in [6, 8]. This paper extends the framework of [8] and so we consider the notion of *stuttering-invariant observation-based strategies* where the controller makes choice of actions only when changes in its observation occur. The observations are defined by the values of a finite set of *observable state predicates*. Observable predicates correspond, for example, to information that can be obtained through sensors by the controller. In [8], a symbolic algorithm for computing observation-based strategies for a *fixed* set of observable predicates is proposed, and this algorithm has been implemented in Uppaal-Tiga.

In the current paper, we further develop the approach of [8] and consider a set of *available* observation predicates equipped with a cost function. Our objective is to synthesize a winning strategy that uses a subset of the available observable predicates with a *minimal cost*. Clearly, this can be useful in the design process when we need to select sensors to build a controller.

Our algorithm works by iteratively picking different subsets of the set of the available observable predicates, solving the game for these sets of predicates and finally finding the controllable combination with the minimal cost. Our algorithm avoids the exploration of all possible combinations by taking into account the inclusion-set relations between different sets of observable predicates and monotonic properties of the underlying games. Additionally, for efficiency reasons, our algorithm reuses, when solving the game for a new set of observation predicates, information computed on previous sets whenever possible.

Related works Several works in the literature consider the synthesis of controllers along with some notion of optimality [5, 7, 4, 12, 16, 23, 13, 19] but they consider the minimization of a cost along the execution of the system while our aim is to minimize a static property of the controller: the cost of observable predicates on which its winning strategy is built. The closest to our work is [13] where the authors consider the related but different problem of turning on and off sensors during the execution in order to minimize energy consumption. In [15], the authors consider games with perfect information but the discovery of interesting predicates to establish controllability. In [14] this idea is extended to games with imperfect information. In those two works the set of predicates is not fixed a priori, there is no cost involved and the problems that they consider are undecidable. In [19], a related technique is used: a hierarchy on different levels of abstraction is considered, which allows to use analysis done on coarser abstractions to reduce the state space to be explored for more precise abstractions.

Structure of the paper In section 2, we define a notion of *labeled transition systems* that serves as the underlying formalism for defining the semantics of the two-player safety games. In the same section we define imperfect information games and show the reduction of [22] of these games to the games with complete information. Then in section 3 we define *timed game automata*, that we use as a modeling formalism. In section 4, we state the cost-optimal controller synthesis problem and show that a natural extension of this problem (that considers a

simple infinite set of observation predicates) is undecidable. In section 5, we propose an algorithm and in section 6, we present two case studies.

2 Games with Incomplete Information

2.1 Labeled Transition Systems

Definition 1 (Labeled Transition System). *A* Labeled Transition System *(LTS) A is a tuple* $(S, s_{init}, \Sigma, \rightarrow)$ *where:*

- S *is a (possibly infinite) set of states,*
- $s_{init} \in S$ *is the initial state,*
- Σ *is the set of actions,*
- $\rightarrow \subseteq S \times \Sigma \times S$ *is a transition relation, we write* $s_1 \xrightarrow{a} s_2$ *if* $(s_1, a, s_2) \in \rightarrow$.

W.l.o.g. we assume that a transition relation is total, i.e. for all states $s \in S$ *and actions* $a \in \Sigma$*, there exists* $s' \in S$ *such that* $s \xrightarrow{a} s'$.

A *run* of a LTS is a finite or infinite sequence of states $r = (s_0, s_1, \dots, s_n, \dots)$ such that $s_i \xrightarrow{a_i} s_{i+1}$ for some action $a_i \in \Sigma$. r^i denotes the prefix run of r ending at s_i. We denote by $Runs(A)$ the set of all finite runs of the LTS A and by $Runs^\omega(A)$ the set of all infinite runs of the LTS A.

A *state predicate* is a characteristic function $\varphi : S \rightarrow \{0, 1\}$. We write $s \models \varphi$ iff $\varphi(s) = 1$.

We use LTS as arenas for games: at each round of the game Player I (Controller) chooses an action $a \in \Sigma$, and Player II (Environment) resolves the nondeterminism by choosing a transition labeled with a. Starting from the state s_{init}, the two players play for an infinite number of rounds, and this interaction produces an infinite run that we call the *outcome* of the game. The objective of Player I is to keep the game in states that satisfy a state predicate φ, this predicate typically models the safe states of the system.

More formally, Player I plays according to a strategy λ (of Player I) which is a mapping from the set of finite runs to the set of actions, i.e. $\lambda : Runs(A) \rightarrow \Sigma$. We say that an infinite run $r = (s_0, s_1, s_2, \dots, s_n, \dots) \in Runs^\omega(A)$ is *consistent* with the strategy λ, if for all $0 \leq i$, there exists a transition $s_i \xrightarrow{\lambda(r^i)} s_{i+1}$. We denote by $\mathsf{Outcome}(A, \lambda)$ all the infinite runs in A that are consistent with λ and start in s_{init}. An infinite run $(s_0, s_1, \dots, s_n, \dots)$ *satisfies* a state predicate φ if for all $i \geq 0$, $s_i \models \varphi$. A (perfect information) *safety game* between Player I and Player II is defined by a pair (A, φ), where A is an LTS and φ is a state predicate that we call a *safety state predicate*. The *safety game problem* asks to determine, given a game (A, φ), if there exists a strategy λ for Player I such that all the infinite runs in $\mathsf{Outcome}(A, \lambda)$ satisfy φ.

2.2 Observation-Based Stuttering-Invariant Strategies

In the imperfect information setting, Player I observes the state of the game using a set of *observable predicates* $obs = \{\varphi_1, \varphi_2, \dots, \varphi_m\}$. An *observation* is a

valuation for the predicates in *obs*, i.e. in a state s, Player I is given the subset of observable predicates that are satisfied in that state. This is defined by the function γ_{obs}:

$$\gamma_{obs}(s) \equiv \{\varphi \in obs \mid s \models \varphi\}$$

We extend the function γ_{obs} to sets of states that satisfy the same set of observation predicates. So, if all the elements of some set of states $v \subseteq S$ satisfy the same set of observable predicates o (i.e. $\forall s \in v \cdot \gamma_{obs}(s) = o$), then we let $\gamma_{obs}(v) = o$.

In a game with imperfect information, Player I has to play according to *observation based stuttering invariant strategies* (OBSI strategies for short). Initially, and whenever the current observation of the system state changes, Player I proposes some action $a \in \Sigma$ and this intuitively means that he wants to play the action a whenever this action is enabled in the system. Player I is not allowed to change his choice as long as the current observation remains the same.

An *Imperfect Information Safety Game* (IISG) is defined by a triple (A, φ, obs).

Consider a run $r = (s_0, s_1, \ldots, s_n)$, and its prefix r' that contains all the elements but the last one (i.e. $r = r' \cdot s_n$). A *stuttering-free* projection $r \downarrow obs$ of a run r over a set of predicates *obs* is a sequence, defined by the following inductive rules:

- if r is a singleton (i.e. $n = 0$), then $r \downarrow obs = \gamma_{obs}(s_0)$
- else if $n > 0$ and $\gamma_{obs}(s_{n-1}) = \gamma_{obs}(s_n)$, then $r \downarrow obs = r' \downarrow obs$
- else if $n > 0$ and $\gamma_{obs}(s_{n-1}) \neq \gamma_{obs}(s_n)$, then $r \downarrow obs = r' \downarrow obs \cdot \gamma_{obs}(s_n)$

Definition 2. *[8] A strategy λ is called obs-Observation Based Stuttering Invariant (obs-OBSI) if for any two runs r' and r'' such that $r' \downarrow obs = r'' \downarrow obs$, the values of λ on r' and r'' coincide, i.e. $\lambda(r') = \lambda(r'')$.*

We say that Player I wins in IISG (A, φ, obs), if there exists a *obs*-OBSI strategy λ for Player I such that all the infinite runs in $\mathsf{Outcome}(A, \lambda)$ satisfy φ.

2.3 Knowledge Games

The solution of a IISG (A, φ, obs) can be reduced to the solution of a *perfect information* safety game (G, ψ), whose states are sets of states in A and represent the *knowledge* (beliefs) of Player I about the current possible states of A.

We assume that $\varphi \in obs$, i.e. the safety state predicate is observable for Player I. This is a reasonable assumption since Player I should be able to know whether he loses the game or not.

Consider an LTS $A = (S, s_{init}, \Sigma, \rightarrow)$. We say that a transition $s_1 \xrightarrow{a} s_2$ in A is *obs*-visible, if the states s_1 and s_2 have different observations (i.e. $\gamma_{obs}(s_1) \neq \gamma_{obs}(s_2)$), otherwise we call this transition to be *obs*-invisible. Let $v \subseteq S$ be *a knowledge* (belief) of Player I in A, i.e. it is some set of states that satisfy the same observation. The set $Post_{obs}(v, a)$ contains all the states that are accessible from the states of v by a finite sequence of a-labeled *obs*-invisible transitions followed by an a-labeled *obs*-visible transition. More formally, $Post_{obs}(v, a)$ contains all

the states s', such that there exists a run $s_1 \xrightarrow{a} s_2 \xrightarrow{a} \ldots \xrightarrow{a} s_n$ and $s_1 \in v$, $s_n = s'$, $\gamma_{obs}(s_i) = \gamma_{obs}(s)$ for all $1 \leq i < n$, and $\gamma_{obs}(s_n) \neq \gamma_{obs}(s)$.

The set $Post_{obs}(v, a)$ contains all the states that are visible for Player I after he continuously offers to play action a from some state in v. Player I can distinguish the states s_1 and s_2 from $Post_{obs}(v, a)$ iff they have different observations, i.e. $\gamma_{obs}(s_1) \neq \gamma_{obs}(s_2)$. In other words, the set $\{Post_{obs}(v, a) \cap \gamma_{obs}^{-1}(o) \mid o \in \mathcal{P}(obs)\} \setminus \{\varnothing\}$ consists of all the beliefs that Player I might have after he plays the a action from the knowledge set v[5].

A game can *diverge* in the current observation after playing some action. To capture this we define the boolean function $Sink_{obs}(v, a)$ whose value is true iff there exists an infinite run $(s_0, s_1, \ldots, s_n, \ldots) \in Runs(A)$ such that $s_0 \in v$ and for each $i \geq 0$ we have $s_i \xrightarrow{a} s_{i+1}$ and $\gamma_{obs}(s_i) = \gamma_{obs}(s_0)$.

Definition 3. *We say, that a game (G, ψ) is the knowledge game for (A, φ, obs), if $G = (V, v_{init}, \Sigma, \rightarrow_g)$ is an LTS and*

- $V = \{v \in \mathcal{P}(S) \mid \forall s_1, s_2 \in v \cdot \gamma_{obs}(s_1) = \gamma_{obs}(s_2)\} \setminus \{\varnothing\}$ *is the set of all the beliefs of Player I in A,*
- $v_{init} = \{s_{init}\}$ *is the initial game state,*
- $\rightarrow_g$ *represents the game transition relation; a transition $v_1 \xrightarrow{a}_g v_2$ exists iff:*
  - $v_2 = Post_{obs}(v_1, a) \cap \gamma_{obs}^{-1}(o)$ *and $v_2 \neq \varnothing$ for some $o \subseteq obs$, or*
  - $Sink_{obs}(v_1, a)$ *is true and $v_2 = v_1$.*
- $v \models \psi$ *iff $\varphi \in \gamma_{obs}(v)$.*

Theorem 1 ([8]). *Player I wins in a IISG (A, φ, obs) iff he has a winning strategy in the safety game (G, ψ) which is the knowledge game for (A, φ, obs).*

This theorem gives us the algorithm of solution of a IISG for the case when the knowledge games for it is finite and can be automatically constructed.

3 Timed Game Automata

The knowledge game (G, ψ) for (A, φ, obs) is finite when the source game A is finite [22]. The converse is not true and there are higher level formalisms that can induce *infinite* games for which knowledge games are still *finite* and can be automatically constructed. One of such formalisms is Timed Game Automata [17], that we use as a modeling formalism and that has been proved in [8] to have finite state knowledge games.

Let X be a finite set of real-valued variables called clocks. We denote by $\mathcal{C}(X)$ the set of constraints ψ generated by the grammar: $\psi ::= x \sim k \mid x - y \sim k \mid \psi \wedge \psi$ where $k \in \mathbb{N}$, $x, y \in X$ and $\sim \in \{<, \leq, =, >, \geq\}$. $\mathcal{B}(X)$ is the set of constraints generated by the following grammar: $\psi ::= \top \mid k_1 \leq x < k_2 \mid \psi \wedge \psi$ where $k, k_1, k_2 \in \mathbb{N}$, $k_1 < k_2$, $x \in X$, and $\top$ is the boolean constant *true*.

A *valuation* of the clocks in X is a mapping $X \mapsto \mathbb{R}_{\geq 0}$. For $Y \subseteq X$, we denote by $v[Y]$ the valuation assigning 0 (respectively, $v(x)$) for any $x \in Y$ (respectively,

[5] the powerset $\mathcal{P}(S)$ is equal to the set of all subsets of S

$x \in X \setminus Y$). We also use the notation $\mathbf{0}$ for the valuation that assigns 0 to each clock from X.

Definition 4 (Timed Game Automata). *A* Timed Game Automaton *(TGA) is a tuple* $(L, l_{init}, X, E, \Sigma_c, \Sigma_u, I)$ *where:*

- L *is a finite set of locations,*
- $l_{init} \in L$ *is the initial location,*
- X *is a finite set of real-valued clocks,*
- Σ_c *and* Σ_u *are finite the sets of controllable and uncontrollable actions (of Player I and Player II, correspondingly),*
- $E \subseteq (L \times \mathcal{B}(X) \times \Sigma_c \times 2^X \times L) \cup (L \times \mathcal{C}(X) \times \Sigma_u \times 2^X \times L)$ *is partitioned into controllable and uncontrollable transitions*[6]*,*
- $I : L \to \mathcal{B}(X)$ *associates to each location its* invariant*.*

We first briefly recall *the non-game* semantics of TGA, that is the semantics of Timed Automata (TA) [2]. A state of TA (and TGA) is a pair (l, v) of a location $l \in L$ and a valuation v over the clocks in X. An automaton can do two types of transitions, that are defined by the relation $\hookrightarrow$:

- a **delay** $(l, v) \overset{t}{\hookrightarrow} (l, v')$ for some $t \in \mathbb{R}_{>0}$, $v' = v + t$ and $v' \models I(l)$, i.e. to stay in the same location while the invariant of this location is satisfied, and during this delay all the clocks grow with the same rate, and
- a **discrete transition** $(l, v) \overset{a}{\hookrightarrow} (l', v')$ if there is an element $(l, g, a, Y, l') \in E$, $v \models g$ and $v' = v[Y]$, i.e. to go to another location l' with resetting the clocks from Y, if the guard g and the invariant of the target location l' are satisfied.

In the remainder of this section, we define the *game semantics* of TGA. As in [8], for TGA, we let observable predicates be of the form (K, ψ), where $K \subseteq L$ and $\psi \in \mathcal{B}(X)$. We say that a state (l, v) satisfies (K, ψ) iff $l \in K$ and $v \models \psi$.

Intuitively, whenever the current observation of the system state changes, Player I proposes a controllable action $a \in \Sigma_c$ and as long as the observation does not change Player II has to play this action when it is enabled, and otherwise he can play any uncontrollable actions or do time delay. Player I can also propose a special action **skip**, that means that he lets Player II play any uncontrollable actions and do time delay. Any time delay should be stopped as soon as the current observation is changed, thus giving a possibility for Player I to choose another action to play.

Formally, the semantics of TGA is defined by the following definition:

Definition 5. *The semantics of TGA* $(L, l_{init}, X, E, \Sigma_c, \Sigma_u, I)$ *with the set of observable predicates obs is defined as the LTS* $(S, s_{init}, \Sigma_c \cup \{\mathbf{skip}\}, \to)$*, where* $S = L \times \mathbb{R}^X_{\geq 0}$*,* $s_{init} = (l_{init}, \mathbf{0})$ *and the transition relation is: (*$\hookrightarrow$ *denotes the non-game semantics of M)*

[6] We follow the definition of [8] that also assumes that the guards of the controllable transitions should be of the form $k_1 \leq x < k_2$. This allows us to use the results from that paper. In particular, we use *urgent* semantics for the controllable transitions, i.e. for any controllable transition there is an exact moment in time when it becomes enabled.

– $s \xrightarrow{\textbf{skip}} s'$ *exists, iff* $s \overset{a_u}{\hookrightarrow} s'$ *for some* $a_u \in \Sigma_u$*, or there exists a delay* $s \overset{t}{\hookrightarrow} s'$ *for some* $t \in \mathbb{R}_{>0}$ *and any smaller delay doesn't change the current observation (i.e. if* $s \overset{t'}{\hookrightarrow} s''$ *and* $0 \leq t' < t$ *then* $\gamma_{obs}(s) = \gamma_{obs}(s'')$*).*
– *for* $a \in \Sigma_c$*,* $s \xrightarrow{a} s'$ *exists, iff:*
  - *a is enabled in s and there exists a discrete transition* $s \overset{a}{\hookrightarrow} s'$*, or*
  - *a is not enabled in s, but there exists a discrete transition* $s \overset{a_u}{\hookrightarrow} s'$ *for some* $a_u \in \Sigma_u$*, or*
  - *there exists a delay* $s \overset{t}{\hookrightarrow} s'$ *for some* $t \in \mathbb{R}_{>0}$*, and for any smaller delay* $s \overset{t'}{\hookrightarrow} s''$ *(where* $0 \leq t' < t$*) the observation is not changed, i.e.* $\gamma_{obs}(s) = \gamma_{obs}(s'')$*, and action a is not enabled in* s''*.*

For a given TGA M, set of observable predicates *obs* and a safety state-predicate φ (that can be again of the form (K, ψ)), we say that Player I wins in the Imperfect Information Safety Timed Game (IISTG) (M, φ, obs) iff he wins in the IISG (A, φ, obs), where A defines the semantics for M and *obs*.

The problem of solution of IISTG is decidable since the knowledge games are finite for TGA [8]. The paper [8] proposes a symbolic Difference Bounded Matrices (DBM)-based procedure to construct them.

4 Problem Statement

Consider that several observable predicates are available, with assigned costs, and we look for a set of observable predicates allowing controllability and whose cost is minimal. This is formalized in the next definition:

Definition 6. *Consider a TGA* M*, a finite set of* available *observable predicates Obs over* M*, a safety observable predicate* $\varphi \in Obs$ *and a monotonic with respect to set inclusion function* $\omega : \mathcal{P}(Obs) \to \mathbb{R}_{\geq 0}$*. The optimization problem for* $(M, \varphi, Obs, \omega)$ *consists in computing a set of observable predicates* $obs \subseteq Obs$ *such that Player I wins in the Imperfect Information Safety Timed Game* (M, φ, obs) *and* $\omega(obs)$ *is minimal.*

We present in the next section our algorithm to compute a solution to the optimization problem. In this paper, we restrict our attention to *finite* sets of available predicates. We justify this restriction by the following undecidability result: considering a reasonable infinite set of observation predicates, the easier problem of the existence of a set of predicates allowing controllability is undecidable:

Theorem 2. *Consider a TGA* M *with clocks* X*, and an (infinite) set of available predicates* $Obs = \{x < \frac{1}{q} \mid x \in X, q \in \mathbb{N}, q \geq 1\}$ *and the safety objective* φ*. Determining whether there exists a finite set of predicates* $obs \subset Obs$ *such that Player I wins in IISTG* (M, φ, obs) *is undecidable.*

Algorithm 1 Lattice-based algorithm

//input: TGA M, a set of observable predicates Obs, a safety predicate φ
//output: a solution with a minimal cost
function $Optimize(M, \varphi, Obs, \omega)$:
1. $candidates := \mathcal{P}(Obs)$ // initially, $candidates$ contains all subsets of Obs
2. $best\_candidate := None$
3. **while** $candidates \neq \varnothing$:
4. **pick** $obs \in candidates$
5. **if** $Solve(M, \varphi, obs)$:
6. $best\_candidate := obs$
7. $candidates = candidates \setminus \{c : c \in \mathcal{P}(Obs) \wedge \omega(c) \geq \omega(obs)\}$
8. **else**:
9. $candidates = candidates \setminus \{c : c \in \mathcal{P}(Obs) \wedge c \subseteq obs\}$
10. **return** $best\_candidate$

5 The Algorithm

The naive algorithm is to iterate through all the possible solutions $\mathcal{P}(Obs)$, for each $obs \in \mathcal{P}(Obs)$ solve IISTG (M, φ, obs) via the reduction to the finite-state knowledge games, and finally pick a solution with the minimal cost.

In section 5.1 we propose the more efficient algorithm that avoids exploring all the possible solutions from $\mathcal{P}(Obs)$. Additionally, in sections 5.2 we describe the optimization that reuses the information between different iterations.

5.1 Basic Exploration Algorithm

Consider, that we already solved the game for the observable predicates sets $obs_1, obs_2, \ldots, obs_n$ and obtained the results $r_1, r_2, \ldots, r_n$, where r_i is either $true$ or $false$, depending on whether Player I wins in IISTG (M, φ, obs_i) or not.

From now on we don't have to consider any set of observable predicates with a cost larger or equal to the cost of the optimal solution found so far. Additionally, if we know, that Player I loses for the set of observable predicates obs_i (i.e. $r_i = false$), then we can conclude that he also loses for any coarser set of observable predicates $obs \subset obs_i$ (since in this case Player I has less observation power). Therefore we don't have to consider such obs as a solution to our optimization problem. This can be formalized by the following definition:

Definition 7. *A sequence $(obs_1, r_1), (obs_2, r_2) \ldots (obs_n, r_n)$ is called a non-redundant sequence of solutions for a set of available observable predicates Obs and cost function ω, if for any $1 \leq i \leq n$ we have $obs_i \subseteq Obs$, $r_i \in \{true, false\}$, and for any $j < i$ we have:*

- *$\omega(obs_j) > \omega(obs_i)$ if $r_j = true$,*
- *$obs_i \not\subseteq obs_j$, otherwise.*

Algorithm 1 solves the optimization problem by iteratively solving the game for different sets of observable predicates so that the resulting sequence of solutions is non-redundant. The procedure $Solve(M, \varphi, obs)$ uses the knowledge

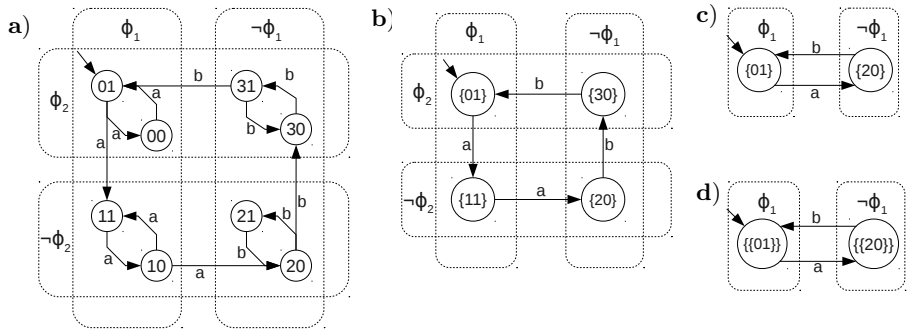

Fig. 1: a) The original LTS A and two observable predicates φ_1 and φ_2,
b) the knowledge game G_f for A with observable predicates $\{\varphi_1, \varphi_2\}$,
c) the knowledge game G_c^1 for A with observable predicates $\{\varphi_1\}$,
d) the knowledge game G_c^2 for G_f with observable predicates $\{\varphi_1\}$

game-reduction technique described in section 2. The algorithm updates the set *candidates* after each iteration and when the algorithm finishes, the *best_candidate* variable contains a reference to the solution with the minimal cost.

Algorithm 1 doesn't state, in which order we should navigate through the set of candidates. We propose the following heuristics:

- *cheap first* (and *expensive first*) — pick any element from the *candidates* with the maximal (or minimal) cost,
- *random* — pick a random element from the *candidates*,
- *midpoint* — pick any element, that will allow us to eliminate as many elements from the *candidates* set as it is possible. In other words, we pick an element that maximizes the value of
  $\min(|\{c : c \in candidates \wedge w(c) \geq w(obs)\}|, |\{c : c \in candidates \wedge c \subseteq obs\}|)$.

Algorithm 1 doesn't specify how we store the set of possible solutions *candidates*. An explicit way (i.e. store all elements) is expensive, because the *candidates* set initially contains $2^{|Obs|}$ elements. However, an efficient procedure for obtaining a next candidate may not exist as a consequence of the following theorem:

Theorem 3. *Let $seq_n = (obs_1, r_1), (obs_2, r_2), \ldots, (obs_n, r_n)$ be a non-redundant sequence of solutions for some set Obs and cost function $\omega : \mathcal{P}(Obs) \to \mathbb{R}_{\geq 0}$. Consider that the value of ω can be computed in polynomial time. Then the problem of determining whether there exists a one-element extension $seq_{n+1} = (obs_1, r_1), (obs_2, r_2), \ldots, (obs_n, r_n), (obs_{n+1}, r_{n+1})$ of seq that is still non-redundant for Obs and ω is NP-complete.*

5.2 State space reusage from finer observations

Intuitively, if we have already solved a knowledge game (G_f, ψ_f) for a set obs_f of observable predicates, then we can view a knowledge game (G_c, ψ_c) associated with a *coarser* set of observable predicates $obs_c \subset obs_f$ as an imperfect information game with respect to (G_f, ψ_f). Thus we can solve the knowledge game

for *obs* without exploring the state space of the TGA M and therefore without using the expensive DBM operations. Moreover, we can build another game on top of G_c (for an observable predicates set that is coarser than *obs*) and thus construct a "Russian nesting doll" of games. This is an important contribution of our paper, since this construction can be applied not only to Timed Games, but also to any modeling formalism that have finite knowledge games.

The state space reusage method is demonstrated on a simple LTS A at Fig. 1. Suppose, that we already built the knowledge game G_f for the observable predicates $\{\varphi_1, \varphi_2\}$. Now, if we want to build a knowledge game for $\{\varphi_1\}$, we can do that in two ways. First, we can build it from scratch based on the state space of A, and the resulting knowledge game G_c^1 is given at subfigure c. Alternatively, we can build the knowledge game G_c^2 on the top of G_f (see subfigure d). The states of G_c^1 are sets of states of A and the states of G_c^2 are sets of sets of states of A. The games G_c^2 and G_c^1 are bisimilar, thus Player I wins in G_c^1 iff he wins in G_c^2 (for any safety predicate). The latter is true for any LTS A, that is stated by the following theorem and corollary:

Theorem 4. *Suppose that* $obs_c \subset obs_f$, (G_f, ψ_f) *is the knowledge game for* (A, φ, obs_f), (G_c^1, ψ_c^1) *is the knowledge game for* (A, φ, obs_c) *and* (G_c^2, ψ_c^2) *is the knowledge game for* (G_f, ψ_f, obs_c). *Then the relation* $R = \{(v, v') | v = \bigcup_{s' \in v'} s'\}$ *between the states of* G_c^1 *and* G_c^2 *is a bisimulation.*

Corollary 1. *Player I wins in* (G_c^1, ψ_c^1) *iff Player I wins in* (G_c^2, ψ_c^2).

This reusage method is also correct for the case when an input model is defined as a TGA (since we can apply the theorem to the underlying LTS).

Implementation Our Python prototype implementation of this algorithm (see https://launchpad.net/pytigaminobs) explicitly stores the set of candidates and uses the on-the-fly DBM-based algorithm of [8] for the construction and solution of knowledge games for IISTG (the algorithm stops early when it detects that the initial state is losing).

6 Case studies

We applied our implementation to two case studies.

The first is a "Train-Gate Control", where two trains tracks merge together on a bridge and the goal of the controller is to prevent their collision. The trains can arrive in any order (or don't arrive at all), thus the challenge for the controller is to handle all possible cases.

The second is "Light and Heavy boxes", where a box is being processed on the conveyor in several steps, and the goal of the controller is to move the box to the next step within some time bound after it has been processed at the current step.

6.1 Train-Gate control

The model of a single (first) train is depicted at Fig. 2. There are two semaphore lights before the bridge on each track. A train passes the distance between

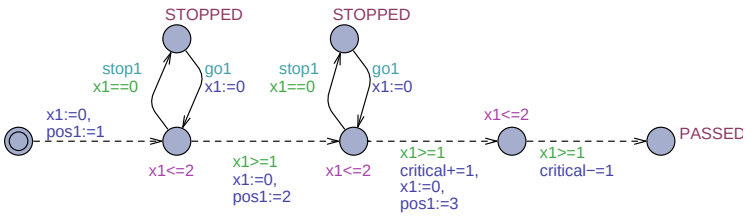

Fig. 2: A model of a single train

semaphores within 1 to 2 time units. A controller can switch the semaphores to red (actions `stop1` and `stop2` depending on the track number), and to green (actions `go1` and `go2`). These semaphores are intended to prevent the trains from colliding on the bridge. When the red signal is illuminated, a train will stop at the next semaphore and wait for the green signal.

It is possible to mount sensors on the semaphores, and these sensors will detect if a train approaches the semaphore. This is modeled with observable predicates $(pos1 \geq 1)$, $(pos2 \geq 1)$, $(pos1 \geq 2)$ and $(pos2 \geq 2)$.

exploration order	expensive first		cheap first		midpoint		random	
state space reusage	with	without	with	without	with	without	with	without
minimum	10m	1h03m	50m	49m	24m	41m	10m	48m
maximum	11m	1h36m	1h30m	1h34m	55m	1h36m	1h26m	1h44m
average	10m	1h18m	1h0m	1h12m	33m	1h03m	37m	1h05m

(a) Running time (the average is computer on 10 runs)

exploration order	expensive first	cheap first	midpoint	random
without state space reusage	1	21.69	5.27	6.17
with state space reusage	7.1	0	2.7	3.46

(b) The average number of iterations

Fig. 3: Results for the Train-Gate model

The controller has a discrete timer that is modeled using the clock y. At any time this clock can be reset by the controller (action `reset`). There is an available observable predicate $(y < 2)$ that becomes false when the value of y reaches 2. This allows the controller to measure time with a precision 2 by resetting y each time this predicate becomes false and counting the number of such resets.

The integer variable *critical* contains the number of trains that are currently on the bridge. The safety property is that no more than one train can be at the critical section (bridge) at the same time and the trains should not be stopped for more than 2 time units:

$$(critical < 2) \wedge ((Train1.STOPPED) \rightarrow (x1 \leq 2)) \wedge ((Train2.STOPPED) \rightarrow (x2 \leq 2))$$

The optimal controller uses the following set of observable predicates: ($pos1 \geq 2$), ($pos2 \geq 2$) and ($y < 2$). Such a controller waits until the second (in time) train comes to the second semaphore, then pauses this train and lets it go after 2 time units.

Figure 3a reports the time needed to find this solution for different parameters of the algorithm. Figure 3b contains the average number of iterations of Algorithm 1 (i.e. game checks for different sets of observable predicates). You can see that it requires only a fraction of the total number of all possible solutions $2^5 = 32$. Additionally, the state space reusage heuristic allows to improve the performance, especially for the "expensive first" exploration order. For this model the most efficient way to solve the optimization problem is to first solve the game with all the available predicates being observed, and then always reuse the state space of this knowledge game. The numbers of 0 and 1 at Figure 3b reflect that we *don't* reuse the state space exactly once for the "expensive first" order, and we never reuse the state space for the "cheap first" exploration order.

The game size ranges from 5 states for the game when only the safety state predicate is observable to 9202 for the case when all the available predicates are observable. The number of the symbolic states of TGA (i.e. different pairs of reachable locations and DBMs that form the states of a knowledge game) ranges from 1297 to 31171, correspondingly.

6.2 Light and Heavy Boxes

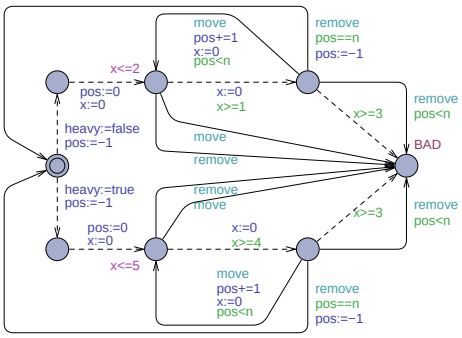

Fig. 4: Light and heavy boxes model

Consider a conveyor belt on which Light and Heavy boxes can be put. A box is processed in n steps (n is a parameter of the model), and the processing at each step takes from 1 to 2 time units for the Light boxes, and from 4 to 5 time units for the Heavy boxes. The goal of the controller is to move a box to the next step (by rotating the conveyor, with an action `move`) within 3 time units after the box has been processed at the current step. At the last step the controller should remove (action `remove`) the box from the conveyor within 3

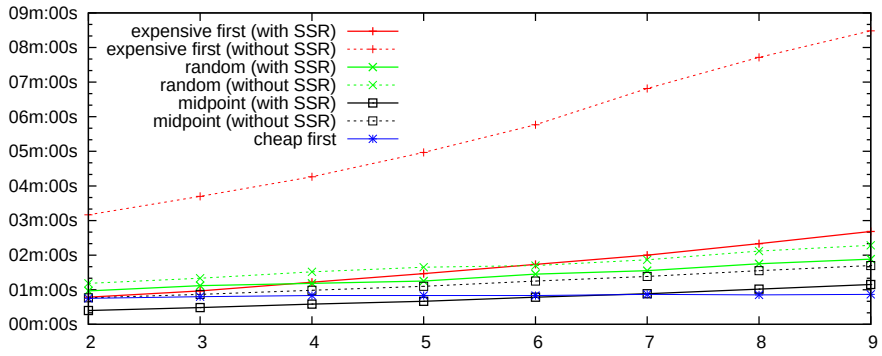

Fig. 5: Average running time (SSR states for State Space Reusage)

time units. If the controller rotates the conveyor too early (before the box has been processed), too late (after more than 3 time units), or does not move it at all, then the Controller loses (similar is true for the removing of the box at the last step). Additionally, the controller should not rotate the conveyor when there is no box on it, and should not try to remove the box when the box is not at the last step. Our model is depicted at Fig. 4, and the goal of the controller is to avoid the `BAD` location.

A box can arrive on the conveyor at any time, and there is an observable predicate $(pos = 0)$ with cost 1 which becomes true when the box is put on the conveyor. Additionally, there is predicate $(heavy = true)$ with cost 1 that becomes true if a heavy box arrives. The model is cyclic, i.e. another box can be put on the conveyor after the previous box has been removed from it.

As in the Traingate model, the controller can measure time using a special clock y. We assume that a controller can measure time with different granularity, and more precise clocks cost more. We model this by having three available observable predicates: $(y < 1)$ with cost 3, $(y < 2)$ with cost 2, and $(y < 3)$ with cost 1.

A naive controller works with the observable predicates $\{(heavy = true), (pos = 0), (y < 1)\}$, resets the clock y each time a new box is arrived, and then move it to the next step (remove after the last iteration) each 2 time units if the box is *light* and 5 time units if the box is *heavy*. However, it is not necessary to use the expensive $(y < 1)$ observable predicate, since a controller can move a box after each 3 (6 for heavy box) time units, thus the time granularity of 3 is enough and there is a controller that uses the observable predicates $\{(heavy = true), (pos = 0), (y < 3)\}$. Our implementation detects such an optimal solution, and Fig. 5 demonstrates an average time needed to compute this solution for different numbers of box processing steps n. You can see that the state space reusage heuristics improves the performance of the algorithm.

The game size for this model ranges from 4 knowledge game states and 51 symbolic NTA states when there are 2 processing steps and only safety predicate is observable to 6417 knowledge game states and 15554 symbolic NTA states for 9 processing steps and when all the available predicate are observable.

7 Conclusions

In this paper we have developed, implemented and evaluated an algorithm for the cost-optimal controller synthesis for timed systems, where the cost of a controller is defined by its observation power.

Our important contributions are two optimizations: the one that helps to avoid exploration of all possible solutions and the one that allows to reuse the state space and solve the imperfect information games on top of each other. Our experiments showed that these optimizations allow to improve the performance of the algorithm.

In the future, we plan to apply our method to other modeling formalisms that have finite state knowledge games.

References

1. Israa AlAttili, Fred Houben, Georgeta Igna, Steffen Michels, Feng Zhu, and Frits W. Vaandrager. Adaptive scheduling of data paths using uppaal tiga. In *QFM*, pages 1–11, 2009.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
3. Gerd Behrmann, Agnes Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. UPPAAL-TIGA: Time for playing games! In *Proceedings of the 19th International Conference on Computer Aided Verification*, number 4590 in LNCS, pages 121–125. Springer, 2007.
4. Patricia Bouyer, Thomas Brihaye, Vronique Bruyre, and Jean-Franois Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.
5. Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim Guldstrand Larsen. Optimal strategies in priced timed game automata. In *FSTTCS*, pages 148–160, 2004.
6. Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2003.
7. Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *FORMATS*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.
8. F. Cassez, A. David, K. G. Larsen, D. Lime, and J.-F. Raskin. Timed control with observation based and stuttering invariant strategies. In *Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 192–206. Springer, 2007.
9. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR'05*, volume 3653 of *LNCS*, pages 66–80. Springer–Verlag, August 2005.

10. Franck Cassez, Jan J. Jessen, Kim G. Larsen, Jean-François Raskin, and Pierre-Alain Reynier. Automatic synthesis of robust and optimal controllers — an industrial case study. In *Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control*, HSCC '09, pages 90–104, Berlin, Heidelberg, 2009. Springer-Verlag.
11. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible timeline-based plan verification. In Brbel Mertsching, Marcus Hund, and Zaheer Aziz, editors, *KI 2009: Advances in Artificial Intelligence*, volume 5803 of *Lecture Notes in Computer Science*, pages 49–56. Springer Berlin / Heidelberg, 2009.
12. Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Rohit Singh 0002. Quasy: Quantitative synthesis tool. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 267–271. Springer, 2011.
13. Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. Controller synthesis with budget constraints. In *HSCC*, volume 4981 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2008.
14. Rayna Dimitrova and Bernd Finkbeiner. Abstraction refinement for games with incomplete information. In *FSTTCS*, volume 2 of *LIPIcs*, pages 175–186. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.
15. Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Counterexample-guided control. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 886–902. Springer, 2003.
16. Rupak Majumdar and Paulo Tabuada, editors. *Hybrid Systems: Computation and Control, 12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009. Proceedings*, volume 5469 of *Lecture Notes in Computer Science*. Springer, 2009.
17. Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In *in E.W. Mayr and C. Puech (Eds), Proc. STACS'95, LNCS 900*, pages 229–242. Springer, 1995.
18. Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, pages 229–242, 1995.
19. Janusz Malinowski, Peter Niebert, and Pierre-Alain Reynier. A hierarchical approach for the synthesis of stabilizing controllers for hybrid systems. In *Proc. 9th International Symposium on Automated Technology for Verification and Analysis (ATVA'11)*, volume 6996 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2011.
20. Andrea Orlandini, Alberto Finzi, Amedeo Cesta, and Simone Fratini. TGA-based controllers for flexible plan execution. In Joscha Bach and Stefan Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence*, volume 7006 of *Lecture Notes in Computer Science*, pages 233–245. Springer Berlin / Heidelberg, 2011.
21. Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(3), 2007.
22. John H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, October 1984.
23. Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.