Simple Algorithm for Simple Timed Games*

Y. Abdeddaïm
ESIEE Paris, University of Paris-Est
y.abdeddaim@esiee.fr

E. Asarin M. Sighireanu
LIAFA, University Paris Diderot and CNRS
{asarin,sighirea}@liafa.jussieu.fr

Abstract

We propose a subclass of timed game automata (TGA), called Task TGA, representing networks of communicating tasks where the system can choose when to start the task and the environment can choose the duration of the task. We search to solve finite-horizon reachability games on Task TGA by building strategies in the form of Simple Temporal Networks with Uncertainty (STNU). Such strategies have the advantage of being very succinct due to the partial order reduction of independent tasks. We show that the existence of such strategies is an NP-complete problem. A practical consequence of this result is a fully forward algorithm for building STNU strategies. Potential applications of this work are planning and scheduling under temporal uncertainty.

1 Introduction

Timed Game Automata (TGA) model has been introduced in [21] in order to represent open timed systems, and is nowadays extensively studied both from a theoretical and applied viewpoints. As usual in the game theory, main problems for timed games are (1) search for winning strategies that allow to the protagonist player (or the controller) to reach his or her aims whatever the opponent player (or the environment) does, and (2) search for winning states (from which such a strategy exists). A winning strategy can be interpreted as a control program that respects its specification (or maximizes some value function) for any actions of the environment, and for this reason game-solving for TGA is often referred to as controller synthesis. Such a synthesis finds applications to, e.g., industrial plants [24], robotics [1], or multi-media documents [17].

A large class of applications of timed games, relevant to this work, is scheduling and planning under temporal uncertainty (see [3, 12]). For this kind of applications, the protagonist masters all the behaviour of the system, except the durations of its actions. The aim is to ensure a good functioning of the system for any timing (decided by the opponent/environment player in some predefined bounds). Thus a strategy is a control program, schedule or plan robust to timing variations in the controlled process. During the last decade, the planning community has developed several techniques for timed planning problems. These techniques (e.g., [18]) are based on constraint programming and give good performances in practice.

On the other hand, during first years of timed games, a major difficulty was related to the fast state explosion and non-scalability of algorithms and tools. Theoretically, this is not surprising since the upper bound complexity of solving reachability games on TGA has been proved EXPTIME [16]. But the main reason of these bad performances was that most algorithms explored (in a backward way [21, 2], or in a mixed one [24]) almost all the huge symbolic state space of the timed game automaton. Four years ago, an important practical progress in timed strategy synthesis has been attained by Cassez et al. [9], who adapted to the timed case a forward on-the-fly game-solving algorithm from [19], and implemented the resulting algorithm in the tool UPPAAL-TIGA [7]. This tool has a performance comparable to reachability analysis of timed automata, and thus makes timed game solving only as difficult as timed verification. However, in many practical cases, even this solution is not always satisfactory. One issue is still the state explosion preventing the tool from finding a strategy. Another issue is a big size and complexity of the strategy synthesized. It is often too heavy to be deployed on the controller. For distributed systems, the controller has to be centralized because the strategy obtained is difficult to distribute. Partial order based methods [8, 20] could help, but as far as we know, they have not yet been applied to timed games.

In this paper, we give a formal explanation of the

*This work has been supported by the French ANR project AMAES.

good results obtained by the planning community by (1) identifying a game model for the timed planning problems and by (2) showing that the complexity of solving such games becomes NP-complete if we search to build strategies of a special form. As a consequence, we obtain a fully forward algorithm for solving timed games by building compact and easy to distribute strategies.

More precisely, we consider games played on a special kind of TGA, so-called *task timed game automata* (TTGA). Such an automaton is a parallel composition of several TGA called tasks. In every task, the controller executes a (possibly infinite) sequence of steps. At each step, the controller decides when to launch some actions whose durations are chosen by the environment and it waits the end of actions launched before starting the next step. Figure 1 gives an example of a TTGA with five tasks. The task **Move** repeats three steps infinitely, at each step the controller launches one action using the controllable transition (solid arrow); the duration of this action is chosen by the environment in the interval given on the uncontrollable transition (dashed arrow). Notice that, in our games, the environment (the opponent player) has no discrete choice, it determines only some durations. The controller has only the choice of the task to execute, but tasks cannot do discrete choices. The aim of the controller is to reach some set of goal states before some deadline (the horizon of the game), and never leave the set of safe states until this goal.

For such games, we will consider strategies of a certain form. First, we disallow "conditional moves", i.e., the discrete choices of the controller are fixed, and the only way that it reacts to the choices of the environment is by adapting the time at which it launches its actions. This requirement limits somewhat the power of the controller. For example, we are not able to obtain optimality like in [2] because this needs conditional schedulers. Second, instead of a memory-less strategy saying for each state what to do (like in almost all the papers/tools on timed games), we represent a strategy by a compact data structure, called STNU [25], and a computational procedure saying for every game history what are possible next moves for the protagonist. In [1] we show that such a strategy is, by its nature, a more complex object than the usual "state feedback" (or memory-less), but it is in most cases smaller wrt the number of transitions needed to reach the goal, more permissive wrt the number of runs included, and easier to distribute in a multi-component timed system (although some parts of the execution shall be centralized).

Related work. A data structure similar to STNU, called event zones, has been used in [20] for the verification of timed automata. STNU extend event zones with uncontrollable edges. The IxTeT planner [14] uses the STNU data structure and an A^*-like search algorithms to obtain STNU plans from constraint-based specification. Our work gives a TGA model sufficient and yet more expressive for modeling pure[1] timed planning problems considered by IxTeT. Moreover, the completeness of the algorithm used by IxTeT, even for pure timed planning problems, has been a conjecture. We provide here a proof of this conjecture. Finally, our work fulfills the study of the relation between STNU and TGA started by Vidal in [26]. In his work, Vidal shows that TGA are more expressive than STNU and provides a semantics for the execution of STNU strategies in terms of reachability games on TGA.

2 Task Timed Game Automata

A TTGA is the parallel composition of a finite set of special TGA called tasks. In each task, the transitions form a sequence with a (possibly empty) final cycle. The sequence of transitions is built from subsequences where a controllable transition is followed by zero or more uncontrollable transitions. These subsequences model a step of the controller starting zero or more tasks with uncontrollable durations but with totally ordered finishing times. When executing a controllable transition, a task may synchronize with its peers only by inspecting their location.

First, let us recall the classical definition of TGA. Let X be a finite set of clocks with values in $\mathbb{R}_{\geq 0}$. We note by $\mathcal{B}(X)$ the set of *rectangular constraints* on variable in X which are possibly empty conjuncts of atomic constraints of the form $x \sim k$ where $k \in \mathbb{N}$, $\sim \in \{<, \leq, \geq, >\}$, and $x \in X$. A constraint in $\mathcal{B}(X)$ defines an interval in $\mathbb{R}_{\geq 0}$ for any $x \in X$.

Definition 2.1 *[21] A Timed Game Automata (TGA) is a tuple $A = (L, \ell_0, X, E, \texttt{Inv})$ where L is a finite set of locations, $\ell_0 \in L$ is the initial location, X is a finite set of real-valued clocks, $E \subseteq L \times \mathcal{B}(X) \times \{c, u\} \times 2^X \times L$ is a finite set of controllable (label c) and uncontrollable (label u) transitions, $\texttt{Inv} : L \rightarrow \mathcal{B}(X)$ associates to each location its invariant.*

For simplicity, this classical definition does not include discrete variables. However, TGA can be extended with discrete variables whose values can be tested and assigned in transitions. Also, TGA can be composed in parallel to form networks of TGA. The synchronization

[1] IxTeT models are in fact hybrid since they may relate timed variables (clocks) and discrete variables.

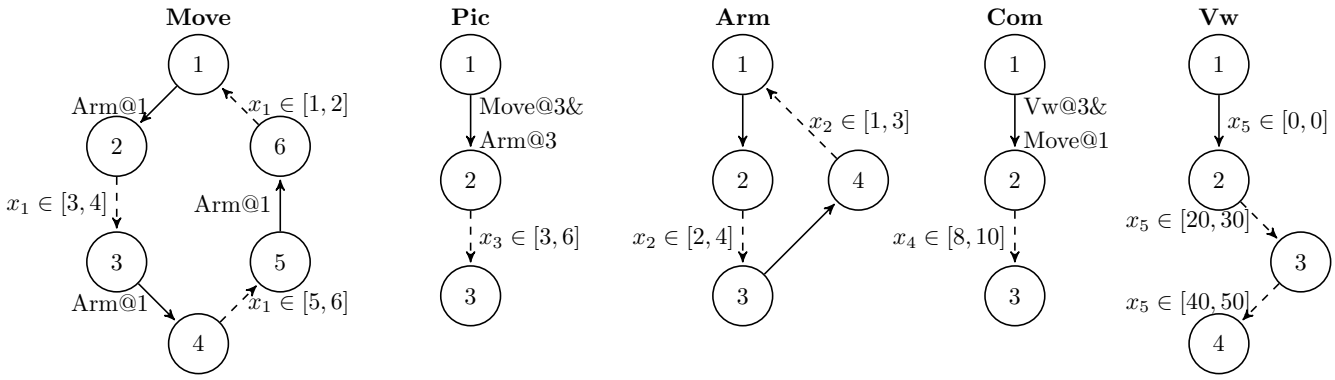

Figure 1. TTGA network for the Explore rover.

between parallel TGA can be done using shared clocks or discrete variables.

Then, a TTGA is a network of TGA where (1) each TGA has a special form, called task, (2) the clocks are not shared between parallel tasks, and (3) the synchronization is done only using location constraints. A *location constraint* $i@[m,n]$ requires that the task i is in any location between locations m and n. Let I be a finite set of tasks identifiers. We denote by $\mathcal{I}(I)$ the set of possibly empty conjunctions of locations constraints over the set of tasks I. Wlog, we consider that all location constraints in $\mathcal{I}(I)$ are well formed, i.e., they refer to existing locations in these tasks.

Definition 2.2 *A* task timed game automaton *(TTGA) is a finite set of tasks, $A = ||_{i\in I} T_i$, each task T_i being a tuple $(L_i, \ell_i^0, x_i, E_i)$ where L_i is a finite set of* locations*, $\ell_i^0 \in L_i$ is the* initial location*, x_i is the* local clock*, and $E_i : L_i \rightharpoonup \mathcal{I}(I\backslash\{i\}) \times \mathcal{B}(x_i) \times \{c,u\} \times L_i$ is a finite set of transitions satisfying the following constraints:*

(i) (chain or lasso) at most one location has no successor by E_i, i.e., $|L_i| - 1 \leq |Dom(E_i)|$,

(ii) (no synchronization for the environment) for any transition $\ell \mapsto (g_d, g_x, u, \ell') \in E_i$ the constraint g_d is empty (true),

(iii) (no time blocking for the environment) for any two consecutive transitions $\ell \mapsto (g_d, g_x, a, \ell')$ and $\ell' \mapsto (g'_d, g'_x, a', \ell'')$ s.t. $a = u$, g_x defines an interval in $\mathbb{R}_{\geq 0}$ which precedes the interval defined by g'_x.

Compared to TGA, tasks have only one clock and a transition relation of a special form (partial function). Moreover, it follows from (i) that a task is either a finite sequence (when $|L_i| - 1 = |Dom(E_i)|$) or a sequence with a final cycle (when $|L_i| = |Dom(E_i)|$). For this reason, we will denote locations $\ell \in L_i$ by a natural number representing the size of the shortest path between ℓ_i^0 and ℓ; by convention, we denote ℓ_i^0 by 1. The transitions of tasks have a (discrete) location constraint, but no set of clocks to be reset. Instead, this set is implicitly determined by the kind of transitions, since x_i is reset iff the transition is controllable. Another implicit definition in tasks is the invariant labeling for locations, `Inv`. For any $\ell \in L_i$, the invariant of ℓ is implicitly $x_i \leq M$, where M is the upper bound of the interval defined by the timed guard g_x in the transition $\ell \mapsto (g_d, g_x, a, \ell')$ in E_i (or empty if such transition does not exists). With this implicit definition for `Inv`, the constraint (iii) ensures that when the environment executes an uncontrollable transition, it can not be blocked by the invariant of the target location. Moreover, uncontrollable transitions are executed in a strict sequence and a controllable transition is executed after the termination of all previous uncontrollable transitions. This constraint has an important consequence: the cycle of a task shall contain at least one controllable transition to reset the clock. Wlog, we will suppose that cycles always start with a controllable transition. Also, like for TGA, we consider only tasks with non-Zeno cycles.

Since TTGA is a subclass of networks of TGA, we omit the definition of its semantics (see [4] for details). We only recall some notions needed to define game semantics. A *run* ρ of a TGA is a sequence of alternating time and discrete transitions also represented by a timed words $(a_1, \tau_1)\dots(a_m, \tau_m)$ where $a_1, \dots, a_m$ are discrete transitions executed at the global time given respectively by $\tau_1, \dots, \tau_m$ and such that when $i \leq j$ then $\tau_i \leq \tau_j$. We denote by $\rho[0]$, $\rho[i]$, and $last(\rho)$ the first state[2], the $(i+1)^{th}$ state (i.e., after the i^{th} discrete action), and resp. the last state of a run ρ. $\texttt{Runs}(A, s)$ denotes the set of runs of A starting in state s.

[2] A state of an automaton is given by the locations of components and valuations of clocks.

Example 2.1 *Our running example is an instance of the Explore system inspired from a Mars rover [5]. The rover explores an initially unknown environment and it can (a) move with the cameras pointing forward; (b) move the cameras (fixed to an arm); (c) take pictures (while still) with the cameras pointing downward, and (d) communicate (while still). The mission of the rover is to navigate in order to take pictures of predefined locations, to communicate with an orbiter during predefined visibility windows, and to return to its initial location before 14 hours. There is a lot of temporal uncertainties, especially in the duration of moves and communications. A TTGA model of this rover is given on Figure 1. Task* Move *models navigation between three predefined locations reached in locations* 1, 3, *resp.* 5. *Task* Pic *models taking a picture at the second location. Task* Arm *models arm moving between two positions: forward (location 1) and downward (location 3). Task* Com *models the communication with an orbiter when the rover is at the first location (the base) and the visibility window is active. Task* Vw *models the visibility window whose start and end are controlled by the environment and it is active in location 3.*

3 Simple Games

The games we consider on TTGA belong to a special subclass of *reachability games*. We first provide general definitions about reachability games [15].

A *reachability game structure* is a tuple $G = (\mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{S})$ where $\mathcal{A}$ is an automaton with edges labeled by controllable and uncontrollable actions, an *initial state* $\mathcal{I}$ in $\mathcal{A}$, a *goal* set of states $\mathcal{G}$, and a *safe* set of states $\mathcal{S}$. The game starts in the initial state $\mathcal{I}$ and, in every state, the controller and the environment choose between waiting or taking a transition they control. The state evolves according to these choices. If the current state is not in $\mathcal{S}$, then the environment wins. If the current state is in $\mathcal{G}$, then the controller wins. Solving a game G consists in finding a *strategy* f such that the automaton $\mathcal{A}$ starting from $\mathcal{I}$ and supervised by f satisfies at any point the constraints in $\mathcal{S}$ and reaches $\mathcal{G}$. A special class of reachability games are *finite horizon games* where the game stops after some finite *horizon* B. In TCTL, this means that $\mathcal{A}$ supervised by f satisfies the formula $\mathcal{I} \wedge \mathtt{A}[\mathcal{S}\ \mathtt{U}_{\leq B}\ (\mathcal{S} \wedge \mathcal{G})]$.

In this work, we consider a finite horizon reachability game defined as follows:

Definition 3.1 *A* simple game *is a finite horizon game $G = (\mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{S})$ such that:*
– $\mathcal{A}$ is a TTGA and $\mathcal{I}$ is its initial state,
– $\mathcal{G}$ is specified using a (non empty) conjunction of location constraints $i@[m,n]$ saying that any location of task i between m and n is part of the goal,
– $\mathcal{S}$ is specified using a conjunction of constraints of the form:
∘ $i@[m,n] \Longrightarrow j@[k,\ell]$ (embedding) *which requires that if task i is in a location in $[m,n]$ then task j is in a location in $[k,\ell]$, and*
∘ $i@[m,n] \sharp j@[k,\ell]$ (mutex) *which requires mutual exclusion between locations in $[m,n]$ of task i and locations in $[k,\ell]$ of task j.*

Wlog, we consider that safety and goal constraints are minimal, i.e., all redundant conjuncts are eliminated.

Example 3.1 *$\mathcal{G}$ for the Explore example is specified by the constraint:*

$\gamma = \text{Move@1} \wedge \text{Pic@3} \wedge \text{Com@3}$

*specifying that the rover has to take the picture, to communicate with the orbiter, and to return at its initial location. The horizon is $B = 14$ and the set $\mathcal{S}$ is given by the constraint δ below. In δ, the first two conjuncts forces the rover to take the picture (*Pic@2*) when the arm is downward (*Arm@3*) and the rover is still at the second location (*Move@3*); the next conjuncts ask that rover is moving (*Move *at locations* 2, 4, *or* 6*) with the arm oriented forwardly (*Arm@1*).*

$$\begin{aligned}\delta = \ & (\text{Pic@2} \Longrightarrow \text{Arm@3}) \wedge (\text{Pic@2} \Longrightarrow \text{Move@3}) \wedge \\ & \textstyle\bigwedge_{\ell \in \{2,4,6\}} (\text{Move@}\ell \Longrightarrow \text{Arm@1}) \wedge \\ & (\text{Com@2} \Longrightarrow \text{Move@1}) \wedge (\text{Com@2} \Longrightarrow \text{Vw@3})\end{aligned}$$

4 Strategies and STNU

We first recall some definitions concerning strategies. A *strategy* for a controller playing a game G is a relation f between $\mathtt{Runs}(\mathcal{A}, \mathcal{I})$ and the set of controllable transitions in $\mathcal{A}$ extended with a special symbol λ. Its semantics is given by the following three rules: (1) if $(\rho, \lambda) \in f$, the controller may wait in the last state of ρ, (2) if $(\rho, e) \in f$, the controller may take the controllable transition e in the last state of ρ, (3) if ρ is not related by f, the controller has no way to win for ρ with the strategy f. Given a strategy f, we define the *plays* of f, *plays*(f), to be the set of runs that are possible when the controller follows the strategy f. Given a reachability game structure G, a strategy f is a *winning strategy* for the game G if for all $\rho \in$ *plays*(f) such that $\rho[0] \in \mathcal{I}$, there exists a position $i \geq 0$ such that $\rho[i] \in \mathcal{G}$, and for all positions $0 \leq j \leq i$, $\rho[j] \in \mathcal{S}$.

4.1 STNU

For finite horizon games, the winning strategies have a finite representation (in absence of Zeno runs). A compact way to represent such strategies is the Simple Temporal Network with Uncertainties (STNU) [25]. We present shortly STNU, further details can be found in [25, 23].

An STNU is a weighted oriented graph in which edges are divided into two classes: controllable (or requirement) edges and uncontrollable (or contingent) edges. The weights on edges are non-empty intervals in $\mathbb{R}$. The nodes in the graph represent (discrete) events, called *time-points*. The edges correspond to (interval) constraints on the durations between events. The time-points which are target of an uncontrollable edge are controlled by the environment, subject to the limits imposed by the interval on the edge. All other time-points are controlled by the controller, whose goal is to satisfy the bounds on the controllable edges.

Definition 4.1 *An STNU Z is a 5-tuple $\langle N, E, C, l, u\rangle$, where N is a set of nodes, E is a set of oriented edges, C is a subset of E containing controllable edges, and $l : E \to \mathbb{R} \cup \{-\infty\}$ and $u : E \to \mathbb{R} \cup \{+\infty\}$ are functions mapping edges into extended real numbers that are the lower and upper bounds of the interval of possible durations. Each uncontrollable edge $e \in E\backslash C$ is required to satisfy $0 \leq l(e) < u(e) < \infty$. Multiple uncontrollable edges with the same finishing points are not allowed.*

Each STNU is associated with a distance graph [11] derived from the upper and lower bound constraints. An STNU is consistent iff the distance graph does not contain a negative cycle, and this can be determined by a single-source shortest path propagation such as in the Bellman-Ford algorithm [10].

Choosing one of the allowed durations for each edge in an STNU Z corresponds to a *schedule* of time-points and gives a distance graph which can be checked for consistency. Then schedules of Z represent finite runs over the events in Z. Choosing one of the allowed durations for each uncontrollable edge in an STNU Z determines a family of distance graphs called *projections* of Z. Each projection determines a set of finite runs where uncontrollable time-points are executed always at the same moments. An *execution strategy* f for an STNU Z is a partial mapping from projections of Z to schedules for Z such that for any choice p of execution time for uncontrollable time-points, f assigns an execution time for *all* time-points in Z such that (1) if the time-point is uncontrollable, the execution time is equal to one chosen in p, and (2) if the time-point is controllable, the execution time satisfies the constraints on edges in Z. An execution strategy is said *viable* if it produces a consistent schedule for any projection of Z. So an STNU represents several execution strategies.

Various types of execution strategies have been defined in [25]. We consider here the *dynamic execution strategies* which assign a time to each controllable time-point that may depend on the execution time of uncontrollable edges in the past, but not on those in the future (or present). An STNU is *dynamically controllable* (DC) if it represents a dynamic execution strategy. For this reason, we call in the following an *STNU strategy* an STNU having the DC property.

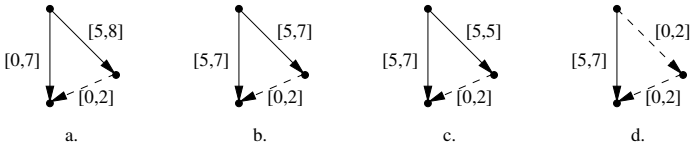

Figure 2. STNU (a) DC but not reduced, (b) reduced wrt shortest path, (c) reduced wrt DC, (d) not DC.

Definition 4.2 *An STNU strategy Z is winning if any schedule of Z is a winning run.*

In [23, 22] is shown that the DC property is tractable, and that a polynomial ($\mathcal{O}(n^5)$ with $n = |N|$) algorithm exists defined as follows:

$$\texttt{procedure}\ \Pi_{DC}\ (\text{STNU}\ Z)\ \texttt{returns}\ Z' \text{ or } \bot$$

The algorithm applies iteratively on Z a set of rewriting rules including the ones in the shortest path algorithm [10]. These rules introduce controllable edges and tighten the bounds of controllable edges in the input STNU. The algorithm returns the rewritten STNU Z' if it is DC, or an empty STNU $\bot$ otherwise.

Figure 2 shows four STNU with the same nodes and edges, but with different labels and properties (uncontrollable edges are represented by dashed arrows).

4.2 Interfacing STNU with simple games

In order to be a strategy for a simple game G, an STNU shall speak about the transitions of the TTGA A done before reaching some goal state in $\mathcal{G}$. We formalize here this relation by defining (1) the interface of a game G and (2) the satisfaction relation between an STNU and a game interface.

Intuitively, the interface of a simple game is an STNU containing a time-point for each transition of the game that can take place until the horizon of the game is reached; these time-points are related with edges labeled by the timing constraints in the TTGA of the game. For tasks with cycles, a transition may appear several times until the game end. Due to the special form of tasks, we can unfold these tasks and compute (see [4] for details) for each transition m of a task i its maximal number of occurrence in the horizon B, denoted by $\mu(i)(m) \geq 0$. Thus, we can identify each transition e that can be executed during the game horizon by a triple (i, m, u) where $i \in I$ is the identifier of the

task owning e, $m \in L_i$ is the source location of e, and $u \in [1, \mu(i)(m)]$ is the occurrence number computed for e. In the following, we denote by $prec(i, m, u)$ the transition preceding (i, m, u) in the unfolding of task i. Similarly, $prec_c(i, m, u)$ denotes the last controllable transition before (i, m, u) in the unfolding of task i. If no such preceding transition exists, both notations return $(i, 0, 1)$.

The *interface of* a simple game G with horizon B is an STNU $Z_G = \langle N_G, E_G, C_G, l_G, u_G \rangle$ and a labeling of time-points $\sigma_G : I \times \mathbb{N} \times \mathbb{N} \to N_G$ defined as follows:
– N_G contains two special time-points t_0 and t_G corresponding respectively to the initial moment and to the goal moment. These two time-points are related by an edge $(t_0, t_G) \in C_G$ labeled by $[0, B]$ to model the finite horizon constraint of the game. Since the time-point t_0 is the initial moment on all tasks, it is labeled by $(i, 0, 1)$ for any $i \in I$.
– For each task i and each transition m of i, N_G contains a number of time-points given by $\mu(i)(m)$.
– E_G relates the nodes in N_G wrt timing constraints and ordering of transitions in A. For example, a transition $e = m \mapsto (g_d, g_x, a, n)$ in some task i defines in E_G a first set of edges $\{prec_c(i, m, u) \xrightarrow{I_{g_x}} (i, m, u) \mid 1 \leq u \leq \mu(i)(m)\}$ where I_{g_x} is the interval on the local clock defined by g_x. This set is included in C_G iff $a = c$; it models the timing constraint g_x which defines a delay from the last reset of the local clock. The second set of edges defined by e is $\{(i, m, u) \xrightarrow{[0,+\infty)} (i, n, u) \mid 1 \leq u \leq \mu(i)(m)\} \subset C_G$ and it models the ordering of transitions starting from m and n in task i.

Example 4.1 *Figure 3 gives the interface of the game considered in Example 3.1. The time-points are put in clusters recalling the task owning the transitions represented by these time-points. Controllable edges without labeling intervals are implicitly labeled by* $[0, +\infty)$.

The interface is the "largest" STNU for G wrt the time-points represented and the constraints on edges, but it can not represent a strategy for G because neither goal nor safety constraints are satisfied. In fact, the interface is used as a sanity check for candidate STNU strategy: a candidate strategy shall contain a "consistent" subset of time-points and edges in the interface of G. Formally, an STNU $Z = \langle N, E, C, l, u \rangle$ is said to *satisfy the interface* (Z_G, σ_G) of a game G, denoted by $Z \models G$, iff (1) N contains t_0 and t_G, (2) N contains a subset of N_G closed by the precedence relation, i.e., if the time-point (i, m, u) is in N then all its predecessors in the unfolding of i are also present, and (3) the edges defined in E_G between the nodes in N are also defined in E with the same kind (controllable or uncontrollable) and the same labeling intervals for uncontrollable edges; for controllable edges, the labeling interval in Z shall be included in the corresponding one in Z_G. Intuitively, Z satisfies the interface of the game G if it puts exactly the same constraints as Z_G on common uncontrollable edges and it may squeeze the constraints on controllable edges.

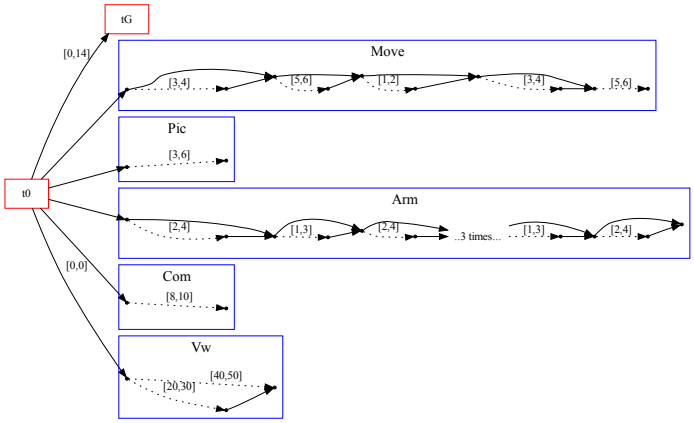

Figure 3. Interface for the Explorer game with horizon $B = 14$.

5 Computing STNU Strategies

We consider the following two problems for simple games G with horizon B given in unary:

G-Solve: Decide if a winning strategy exists for G.

G-Solve-STNU: Decide if a winning STNU strategy exists for G.

Our first result is that, despite the simplicity of the game considered, the problem of finding strategies for such games is still NP-hard.

Theorem 5.1 *The G-Solve problem is NP-hard.*

Proof: The proof consists in reducing the CliqueCover problem [13] to the synthesis of a strategy for simple games. An instance of the CliqueCover problem is an integer k and an undirected graph $P = (V, E)$ with n vertices ($|V| = n$) numbered 1 to n. The CliqueCover problem is to decide whether a graph can be partitioned into k cliques. We explain how to reduce this problem to G_k-Solve problem, where G_k is a simple game. The TTGA of G_k is built as follows: for each vertex $i \in [1, n]$ of P, we define a task indexed by i with the form: $1 \xrightarrow{x_i \geq 0, c, \{x_i\}} 2 \xrightarrow{0 \leq x_i < 1, u, \{\}} 3$. The set $\mathcal{S}$ of G_k is specified by $\bigwedge_{(i,j) \notin E} i@2 \sharp j@2$, i.e., for each

Require: STNU Z s.t. $Z \models G$.
Ensure: Returns true if Z is a winning strategy for G.
1: $\overline{Z} \leftarrow Z \cup Z_G$
2: $\widehat{Z} \leftarrow \Pi_{DC}(\overline{Z})$
3: **if** $\widehat{Z} = \bot$ **then**
4: **return** false
5: **end if**
6: **return** $\widehat{Z} \models shape_{\mathcal{G}oal}$ and $\widehat{Z} \models shape_{\mathcal{S}afe}$

Figure 4. Test for winning STNU strategy.

pair of vertices (i, j) such that there is no edge in E between i and j we ask mutual exclusion between the locations 2 of the corresponding tasks. The set $\mathcal{G}$ of G_k is specified by $\wedge_i i@3$, i.e., all tasks shall reach location 3. The horizon of G_k is fixed to k. It follows (details in [4]) that P can be covered by k cliques or less iff the instance of the G_k-Solve problem has a strategy to win. Moreover, an STNU strategy can be built for G_k. □

A direct consequence of the proof above is:

Corollary 5.1 *The G-Solve-STNU problem is NP-hard.*

The upper bound of complexity for G-Solve is given by the upper bound of solving reachability game problems for TGA, i.e., EXPTIME in [16]. This tight upper bound is obtained for general TGA with safety games and state based (memory less) strategies.

The following theorem says that this upper bound decreases for G-Solve-STNU problems to NP.

Theorem 5.2 *The G-Solve-STNU problem is NP-complete.*

Proof idea. We show that it exists a correct and complete polynomial test to check that an STNU strategy Z satisfying the interface of a simple game G is a winning STNU strategy. Then, the theorem follows since for a horizon B given in unary, the description of a candidate STNU strategy is polynomial in the number of transitions in G and in the horizon B (Lemma 5.2).

The polynomial test is given on Figure 4. First (line 1), Z is filled with the missing time-points and edges in the interface of G thus obtaining a new STNU $\overline{Z}$. Second (line 2), the DC algorithm is applied on $\overline{Z}$ to test its dynamic controllability and to compute its reduced form. Third (line 6), $\widehat{Z}$ is tested for satisfaction of goal and safety constraints in G. This test is done by searching in $\widehat{Z}$ a set of edges (called *shapes*) for each goal constraint in $\mathcal{G}$, for each discrete guard on controllable transitions of the TTGA of G, and for each safety constraint in $\mathcal{S}$.

The technical point in the proof is the definition of shapes such that the test is correct and complete. A shape is a positive boolean composition of precedence relations between two time-points. A time-point t precedes the time-point t' in an STNU Z, denoted by $Z \vdash t \prec t'$, iff Z contains an edge from t to t' labeled by an interval included in $(0, +\infty)$. For example, the shape used to test that the goal constraint $i@[m, n]$ is satisfied by $\widehat{Z}$ is $prec(i, m, u) \prec t_G \wedge t_G \prec (i, n, u)$ (see Figure 5). The test succeeds if such shape exists in $\widehat{Z}$ for some u. To test safety constraint and discrete

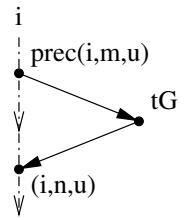

Figure 5. Shape for goal constraint $i@[m, n]$.

guards on transitions, the shapes shall be carefully defined in order to obtain completeness. For example, the shape for a constraint $i@[m, n] \Longrightarrow j@[k, l]$ is the disjunction of three cases which are not disjoint (see Figure 6). Intuitively, the constraint is satisfied if for any occurrence u of transitions in task i leading to location m (time-point $prec(i, m, u)$) and leaving the location n (time-point (i, n, u)) there exists an occurrence u' of the transition in task j leading to location k (time-point $prec(j, k, u')$) and leaving the location l (time-point (j, l, u')) such that one of the following cases holds:
(i) the safety constraint is entirely satisfied since $prec(j, k, u')$ precedes $prec(i, m, u)$ and (i, n, u) precedes (j, l, u'),
(ii) the safety constraint is satisfied at the beginning of the interval $[m, n]$ for i but nothing is asked for the end because the goal (time-point t_G) is reached before the end of the interval $j@[k, l]$; indeed, the game semantics asks that safety constraints are satisfied until the goal is reached but not beyond,
(iii) the safety constraint is not satisfied because the considered time-points are after the goal.

The correctness proof follows easily from the definition of shapes. The completeness is based on the convexity of the STNU strategies and the fact that the shapes defined can not exclude correct but convex STNU. The full proof is given in [4].

To finish the proof, we compute the complexity of the proposed algorithm. If the horizon B is given in unary, the number of points in the interface of G is polynomial, so the first two steps of the algorithm (lines

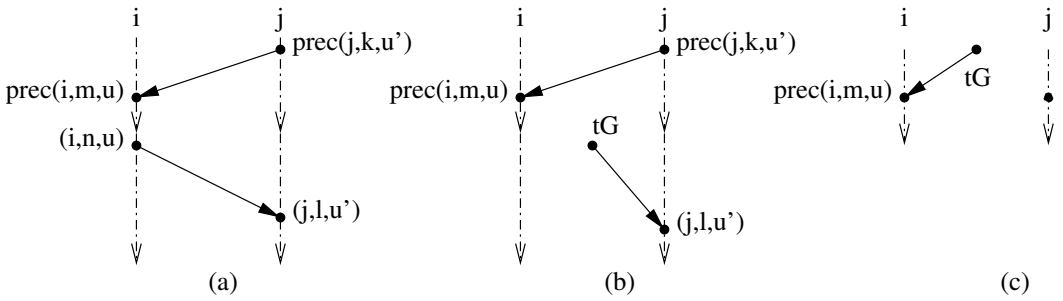

Figure 6. Shape for safety constraint $i@[m,n] \Longrightarrow j@[k,l]$.

1–2) are also polynomial (DC is polynomial in the number of time-points in $\tilde{Z}$). Testing shapes (line 5) is also polynomial. Indeed, to test the satisfaction of some constraint (in guards, goal, or safety), one has to find a set of edges of constant size (the shape) in the STNU. But the number of edges in STNU is quadratic wrt the number of nodes, so we obtain the following result:

Lemma 5.1 *Given an STNU Z satisfying the interface of a game G, the problem of deciding if Z is a winning strategy for G is in PTIME.*

Now, let us compute the size of a representation for STNU that are candidate strategies for a game G. If the TTGA of G has n transitions, the maximal number of time-points in the interface of G is $n \times B + 2$, limit reached when each transition takes one time unit. Then, the maximal number of edges that has to be specified for the STNU candidate is $O(n^2 \times B^2)$. Each of these edges is labeled by an interval given by two integer numbers. However, these numbers are limited by the horizon B.

Lemma 5.2 *The size of describing a candidate STNU strategy for a game G is $O(n^2 \times B^3)$ where n is the number of transitions in the network N and B is the horizon of the game.*

6 Algorithm for solving simple games

The proof of Theorem 5.2 gives also the sufficient conditions for an STNU to be a winning strategy for a simple game G: (1) it has to satisfy the interface of G, (2) it has to be DC, and (3) its reduced form by Π_{DC} has to implement some precedence relations (i.e., some shape) for each time-point concerned by a guard, goal, or safety constraint. The last condition defines a combinatorial space $\mathcal{W}$ for building winning STNU strategies: each strategy represents a choice of the shapes implementing the constraints of G for each time-point in the interface of G.

Therefore, we propose a fully forward algorithm for building winning STNU strategy, called in the following *Win_STNU*, which does a backtracking search in the combinatorial space $\mathcal{W}$. The algorithm starts with the STNU interface of G, Z_G (see Section 4.2). For each choice step in $\mathcal{W}$, it builds a partial solution by adding to Z_G the controllable edges given by the shape chosen. The algorithm may use Π_{DC} as a selection (cut off) test for the partially built solutions. Otherwise, when all choices are done, Π_{DC} is applied to test the DC property of the solution built.

Win_STNU does not find all winning strategies of G but only "standard" ones, i.e., STNU strategies which contain all the time-points of the interface and have maximal intervals on controllable transitions (see [4]). Moreover, the number of time-points of a solution is bounded by B times the number of transitions in G. The properties of our algorithm are summarized by the following proposition which proof is given in [4].

Proposition 6.1 *The algorithm* Win_STNU *is correct and complete wrt standard STNU strategies. It builds solutions of size (number of nodes and edges) quadratic in the size of the game.*

7 Conclusion

We define a class of timed games for which searching winning strategies in STNU form is NP-complete. As a corollary, we obtain a fully forward algorithm for solving the finite horizon reachability timed games. This algorithm builds STNU strategies which are finite memory strategies with no discrete choices but including several orderings between independent actions of the controller. Moreover, the size of these strategies is small, i.e., quadratic in the size of the game. Further works focus on providing a good implementation of our algorithm and on comparing the strategies obtained using other criteria, e.g., their volume [6].

References

[1] Y. Abdeddaïm, E. Asarin, M. Gallien, F. Ingrand, C. Lessire, and M. Sighireanu. Planning Robust Temporal Plans A Comparison Between CBTP and TGA Approaches. In *ICAPS*. AAAI, 2007.

[2] Y. Abdeddaïm, E. Asarin, and O. Maler. On optimal scheduling under uncertainty. In *TACAS*, volume 2619 of *LNCS*, pages 240–253. Springer-Verlag, 2003.

[3] Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *Theor. Comput. Sci.*, 354(2), 2006.

[4] Y. Abdeddaïm, E. Asarin, and M. Sighireanu. Simple algorithm for simple timed games. HAL preprint hal-00374700, 2009.

[5] M. Ai-Chang, J. Bresina, L. Charest, A. Jónsson, J. Hsu, B. Kanefsky, P. Maldague, P. Morris, K. Rajan, and J. Yglesias. MAPGEN: Mixed initiative planning and scheduling for the Mars 03 MER mission. In *ISAIRAS*, 2003.

[6] E. Asarin and A. Degorre. Volume and entropy of regular timed languages. HAL preprint hal-00369812, 2009.

[7] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. Uppaal-Tiga: Time for playing games! In *CAV*, volume 4590 of *LNCS*. Springer-Verlag, 2007.

[8] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *LNCS*, pages 485–500. Springer-Verlag, 1998.

[9] F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, volume 3653 of *LNCS*. Springer-Verlag, 2005.

[10] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1997.

[11] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Network. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[12] A. Fehnker. *Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. PhD thesis, KUNijmegen, 2002.

[13] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[14] M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *AIPS*, 1994.

[15] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *LNCS*. Springer-Verlag, 2002.

[16] T. Henzinger and P. Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221:369–392, 1999.

[17] N. Layaïda, L. Sabry-Ismaïl, and C. Roisin. Dealing with uncertain durations in synchronized multimedia presentations. *Multimedia Tools Appl.*, 18(3):213–231, 2002.

[18] S. Lemai and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *AAAI*, San Jose, CA, USA, 2004.

[19] X. Liu and S. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In *ICALP*, volume 1443 of *LNCS*. Springer-Verlag, 1998.

[20] D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.

[21] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS*, volume 900 of *LNCS*. Springer-Verlag, 1995.

[22] P. Morris. A structural characterization of temporal dynamic controllability. In *CP*, volume 4204 of *LNCS*. Springer-Verlag, 2006.

[23] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI*, 2001.

[24] S. Tripakis and K. Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In *FM*, volume 1706 of *LNCS*. Springer-Verlag, 1999.

[25] T. Vidal. A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *ICAPS*, Breckenridge, CO, USA, 2000.

[26] T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *JETAI*, 11(1), 1999.