Temporal Planning with extended Timed Automata

Christine Largouët, Omar Krichen, Yulong Zhao

Temporal Planning with extended Timed Automata

Christine Largouët
Agrocampus Ouest - IRISA
UMR6074
Rennes, France
Email: clargoue@irisa.fr

Omar Krichen
INRIA of Rennes
UMR6074
Rennes, France

Yulong Zhao
University of Rennes 1
IRISA-UMR6074
Rennes, France

***Abstract*—**

We consider a system modeled as a set of interacting agents evolving along time according to explicit timing constraints. In this kind of system, the planning task consists in selecting and organizing actions in order to reach a goal state in a limited time and in an optimal manner, assuming actions have a cost. We propose to reformulate the planning problem in terms of *model-checking* and *controller synthesis* on interacting agents such that the state to reach is expressed using temporal logic. We have chosen to represent each agent using the formalism of Priced Timed Game Automata (PTGA). PTGA is an extension of Timed Automata that allows the representation of cost on actions and uncontrollable actions. Relying on this domain description, we define a planning algorithm that computes the best strategy to achieve the goal. This algorithm is based on recognized model-checking and synthesis tools from the UPPAAL suite. The expressivity of this approach is evaluated on the classical *Transport Domain* which is extended in order to include timing constraints, cost values and uncontrollable actions. This work has been implemented and performances evaluated on benchmarks.

INTRODUCTION

Planning in multiagent domains, where multiple intelligent entities (called agents) are interacting under *temporal constraints*, is a challenging problem. This is particularly true if the agents define some *cost* on their actions, and have to cooperate in order to achieve a shared-goal. Studies on temporal multiagent systems declined during these last years. However, there is still a lack of a simple and expressive formalism to model temporal multiagent systems. In this paper, we propose an efficient framework to represent non-deterministic systems, defined as a group of interacting agents. The first issue is to model, for each agent, its proper dynamics subject to explicit timing constraints and cost of actions that can eventually be uncontrollable. The second issue is to compute the best plan among a set of possible plans that lead to a specific goal in limited time.

Temporal planning introduced with PDDL 2.1 [1] extends classical planning schemes with the modeling of durative actions and the expression of concurrent plans. More recently, the planner TFPOP [2] is a fully centralized approach that combines temporal and forward-chaining planning. In TFPOP minimum and maximum duration are specified on actions. When the number of agents and interactions grows, the resulting complexity makes difficult the planning task: this is known as the *state explosion problem*. For the exploration of large state spaces, symbolic model-checking has been introduced and is now widely applied for the verification of real-time systems. The connection between planning and model-checking, first proposed by [3], [4] is relatively recent and generated numerous papers on a variety of planning domains. These works have emphasized that this promising approach can tackle the problem of generating plans on nondeterministic models for extended goals. The planner MIPS [5] implements heuristic search algorithms to solve a large number of problems including temporal constraints on actions. More recently works have proposed temporal models such as timed automata in timeline-based planning [6], [7], [8], [9]. Combining network of automata [10] or merging finite state machines [11] are proposed approaches to come up with the scalability of multiagent planning problems.

Following this idea, we propose to represent the system as a network of interacting agents, each agent being described as an extended timed automaton. To cope with the complexity of large systems, we reformulate the planning problem in terms of model-checking and controller synthesis. To model the system we chose the formalism of Priced Timed Game Automata (PTGA), an extension of timed automata [12]. PTGA appears to be a simple and expressive representation of planning multiagent systems that define: non-determinism, timing constraints and costs on actions. The interaction between agents can be performed through synchronized actions. However, existing model-checking tools rely on simple representation of extended timed automata: Timed Game Automata (TGA) and Priced Timed Automata (PTA). The idea is to reformulate the PTGA description model into TGA and PTA and to combine them to answer the planning problem. Whereas time and costs are key issues, the goal is expressed using the temporal logic TCTL (Timed Computational Tree Logic). Given this

representation framework, we propose a planning algorithm, relying on the efficient and recognized tools for the model-checking and the synthesis, to answer a temporal goal. The following requirement is asked: "What is the best strategy to guide the system to a specific goal at a specific time?". In our case, "best" means that a criterion should be minimized and this criterion is the strategy cost. We introduce a classical planning problem, the Transport Domain, distributed to the interacting agents case and extended to include temporal constraints, costs on actions and some uncontrollable actions. The idea here is not to automatically encode any PDDL domain into PTGA but to illustrate our approach with a classical well-known planning benchmark to highlight the benefit of the formalism and to show experimental results.

The paper is structured as follows. Section 2 introduces the formalism of PTGA. Section 3 presents some of the background in model-checking and controller synthesis and gives the definitions of *decision rule* and *strategy*. Section 4 describes our planning algorithm based on PTGA. Section 5 presents the Transport Domain and its adaptation to a multiagent planning problem defined as a PTGA model. Section 6 reports on experimental results while conclusion and perspectives are given Section 7.

Extended Timed Automata Formalism

Timed Automata [13] are automata enriched with a set of variables called clocks. Let $\mathcal{X}$ be a set of *clocks*. A *(clock) valuation* v for a set $\mathcal{X}$ assigns a real value to each clock. The set of *clock constraints* over $\mathcal{X}$, denoted $\Phi(\mathcal{X})$ is defined by the grammar : $\varphi ::= x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2$ where φ, φ_1 and φ_2 belong to $\Phi(\mathcal{X})$; $x \in \mathcal{X}$ is a clock and $c \in \mathbb{N}$ a constant. Clock constraints are evaluated over clock valuations. A constraint φ can be viewed as the set of valuations that satisfy it, hence, we say that v satisfies φ, denoted $v \vDash \varphi$, if $v \in \varphi$.

In timed automata [13], the vertices of the graph are called *locations*, the clock constraint associated to a location is called an *invariant*, whereas the one associated to an edge is called a *guard*. An edge is decorated with an action label and allows the resetting of clocks. In a location, a transition can be triggered if the guard of the outgoing edge is satisfied. The triggering of a transition is instantaneous and takes no time. Priced Timed Automata (PTA) [14] extend the formalism of timed automata by adding *cost* to the behavior of timed systems. In PTA, locations and edges are annotated by cost. The cost label on a location represents the price per time unit while staying in this location whereas the cost label on an edge expresses the cost when the transition is triggered. Timed Game Automaton (TGA) [15] is an extension of timed automata where actions on edges are partitioned into *controllable* and *uncontrollable* actions. A TGA can be used to model a two-player game between an *agent*, that can be called a controller, and the *environnement*. A TGA can represent the *controllable* actions of the controller and the *uncontrollable* actions of the environnement. Based on these principles, Priced Timed Game Automata (PTGA) is an extension of both PTA and TGA [12].

*Definition 1 (***Priced Timed Game Automata (PTGA)):** A Priced Timed Game Automaton $\mathcal{A}$ is a tuple $< \mathcal{S}, \mathcal{X}, \mathcal{A}ct, \mathcal{E}, \mathcal{I}, \mathcal{P} >$ where:

- $\mathcal{S}$ is a finite set of locations and $s_o \in S$ is the initial location.
- $\mathcal{X}$ is a finite set of clocks.
- $\mathcal{A}ct = Act\_c \cup Act\_u$ is a finite set of actions divided into Act_c, the *controllable* actions (played by the controller), and Act_u, the *uncontrollable* actions (played by the environment).
- $\mathcal{E} \subseteq \mathcal{S} \times \mathcal{A}ct \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}} \times \mathcal{S}$ is a finite set of edges. Each edge e is a tuple $(s, a, \varphi, \delta, s')$ such that e connects the location $s \in \mathcal{S}$ to the location $s' \in \mathcal{S}$ on the action label $a \in \mathcal{A}ct$. The enabling condition (called the *guard*) is captured in $\varphi \in \Phi(\mathcal{X})$. $\delta \subseteq \mathcal{X}$ gives the set of clocks to be reset when the transition is triggered.
- $\mathcal{I} : \mathcal{S} \rightarrow \Phi(\mathcal{X})$ maps each location s with a clock constraint called an *invariant*.
- $\mathcal{P} : \mathcal{S} \cup \mathcal{E} \rightarrow \mathbb{N}$ assigns cost rates and costs to locations and edges, respectively.

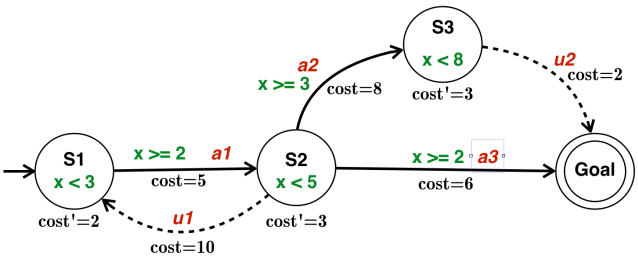

Figure 1. A PTGA example

Figure 1 shows a PTGA with four locations $S1, S2, S3, Goal$ where $S1$ is the initial location. Solid arrows represent the transitions labeled with *controllable* actions ($a1$, $a2$, $a3$) and dashed arrows represent the transitions labelled with *uncontrollable* actions ($u1$, $u2$). The PTGA defines one clock x that allows the definitions of invariants on locations, and guards on edges (in green). For instance, $x < 5$ is the invariant of the location $S2$ and $x \geq 2$ is the guard of its outgoing edge towards the location $Goal$. Each edge can be associated to a guard, the name of the action (in red) and the cost $cost$ of triggering this transition. Another cost, called $cost'$, associated to locations, defines the cost per time unit while staying in this location.

The semantics of a PTGA is defined as a *priced transition system* whose states $q \in \mathcal{Q}$ are the pairs $q = (s, v)$ such that $s \in \mathcal{S}$, is a location, and $v \in \mathbb{N}$ is a clock valuation where v satisfies the invariant $\mathcal{I}(s)$ of the location. We write $q \xrightarrow{a}_c q'$ the transition between two states q and q' labeled by the action a while c is the cost of this transition.

*Definition 2 (***Semantics of a PTGA):** The semantics of a PTGA $A =< \mathcal{S}, \mathcal{X}, \mathcal{A}ct, \mathcal{E}, \mathcal{I}, \mathcal{P} >$ over clocks $\mathcal{X}$ and actions $\mathcal{A}ct$ is given by a priced transition system $\mathcal{T} =< \mathcal{Q}, q_0, \mathcal{A}ct, \rightarrow >$ where $\mathcal{Q} = \{(s, v) \in \mathcal{S} \times \mathbb{N} \mid v \models \mathcal{I}(s)\}$ is the set of sets satisfying the invariants, $q_0 = (s_0, v_0)$ is the initial states for v_0 evaluating to zero for all the clocks in $\mathcal{X}$, $\mathcal{A}ct$ is the set of actions, and $\rightarrow$ consists of discrete and delay transitions as defined below.

*Definition 3 (***Discrete transition):** A transition $(s, v) \xrightarrow{a}_c (s', v')$ is a discrete transition iff there is an edge $e = (s, a, \varphi, \delta, s')$ from the location $s \in \mathcal{S}$ to the location $s' \in \mathcal{S}$ such that the enabling condition (called the *guard*), captured in $\varphi \in \Phi(\mathcal{X})$, is true. $\delta \subseteq \mathcal{X}$ gives the set of clocks to be reset when the transition is triggered and $c = \mathcal{P}(e)$ is the cost of the edge.
The cost value does not impact the triggering of the edge. In a discrete transition, the cost of the transition is the cost of the triggered edge.

A delay transition is performed during a passage of time in a same location without any change of location. We call Δ the time duration elapsed in the location. The cost of a delay transition is computed as the product of the time duration and the cost rate of the active location.

*Definition 4 (***Delay transition):** A delay transition $(s, v) \xrightarrow{\Delta}_c (s, v')$ is a delay transition iff $c = \Delta \times \mathcal{P}(s)$, $v' = v + \Delta$ and the invariant I(s) is satisfied by the source state, the target state, and all the intermediary states such that $v + \Delta' \models \mathcal{I}(s)$ with $\Delta' \leq \Delta$

An execution of a PTGA is a path in the priced transition system defined by the PTGA as defined previously.

*Definition 5 (***Run):** A run of a PTGA is a sequence $q_0 \xrightarrow{\Delta_1}_{c_1} q_1 \xrightarrow{a1}_{c_2} q_2 \xrightarrow{\Delta_2}_{c_3} q_3 \xrightarrow{a2}_{c_4} q_4 \cdots$ of alternating delay and discrete transitions.

If ϱ is a finite run, we denote $last(\varrho)$ the last state of the run and $duration(\varrho)$ the total elapsed time during the run. The cost of ϱ, denoted $cost(\varrho)$ is the sum of all the costs along the execution such that $cost(\varrho) = \sum_i c_i$. The minimum cost of reaching a location s from s_0 is the minimum cost of all the finite runs from s_0 to s

Now consider the PTGA given Figure 1, we propose a possible (not optimal) run ϱ to reach the location $Goal$. For the delay transitions, the duration spent in the location is shown in brackets.

$$\varrho : (s_1, 0) \xrightarrow{\Delta_{(2)}} (s_1, 2) \xrightarrow{a1} (s_2, 2) \xrightarrow{\Delta_{(2)}} (s_2, 4) \xrightarrow{a3} (Goal, 4)$$

The cost of ϱ is $cost(\varrho) = 2 \times 2 + 5 + 2 \times 3 + 6 = 21$

MODEL-CHECKING, CONTROLLER SYNTHESIS AND TOOLS

When considering finite-states machines the planning problem consists in finding a run (a sequence of actions) to reach a goal state. When adding $cost$ and $game$ theory in the model like in PTGA, the planning problem consists in finding the *best plan*, whatever the actions played by the environment (the uncontrollable actions). To face the large combinatorial size of the state-space, symbolic efficient data structures provide a very compact encoding for large sets of states [16]. When a system is described as a network of timed automata (extended or not), it can be treated using two successful techniques: *model-checking* [17] and *controller synthesis* [18], [19].

Timed Computation Tree Logic (TCTL)

Both these techniques require the expression of a property to be satisfied and expressed in a temporal logic. The most popular is the Timed Computation Tree Logic (TCTL) [20], a convenient formalism to specify properties over timed automata. The grammar of TCTL is the following:

$$f ::= p \mid x \in I \mid \neg p \mid p_1 \vee p_2 \mid \exists \Diamond_I p \mid \forall \Diamond_I p \mid \exists \Box_I p \mid \forall \Box_I p$$

where p is a property, $x \in \mathcal{X}$ is a clock and I is a time interval. I is an interval with integer bounds of the form $[n, n']$ with $n, n' \in \mathbb{N}$. The diamond operator $\Diamond p$ expresses that a path (i.e. a sequence of states) leads to a state satisfying the property p. The box operator $\Box p$ means that all the states along a path satisfy the property p. These modal operators can be combined with the universal quantifiers $\exists$ or $\forall$ over the paths. The formula $\exists \Diamond_{[0,3]} p$ expresses that there is at least one path leading to a state satisfying p within 3 units of time.

Model-checking

Model-checking is performed using efficient algorithms called *model-checkers* dedicated to answer whether or not a property is satisfied by the system. The property is expressed using specification languages such as temporal logics. The problem of model-checking can be expressed as follows: given a system model M and a property φ to be checked, does the model M satisfy φ?

Controller synthesis

Controller synthesis is the problem of finding a way to control the system so that the behavior of the system satisfies a given property. The objective is then to *synthesize* a controller. This controller coupled with the system has to respect the given specification. On a PTGA, we denote by Σ the set of possible actions that could be proposed by the controller so that $\Sigma = Act\_c \cup \lambda$ with Act_c the set of

controllable actions and λ the action of letting time pass. A strategy is winning if, when following these rules, the controller always wins whatever the environment does (by the way of non-controllable actions).

Control synthesis distinguishes the two kinds of actions: controllable and uncontrollable actions. Only controllable actions can be controlled by the controller. The interaction between the system and the controller can be seen as a two-players game. On such a game, a *strategy* is a set of decision rules that indicates to the controller which action to choose among the possible several ones.

*Definition 6 (***Decision Rule):** A decision rule gives an action to be performed on a specific system state at a particular time. A decision rule is a tuple (s, φ, σ) such that $s \in \mathcal{S}$ is a location of the PTGA, $\varphi \in \Phi(\mathcal{X})$ is a clock constraint and $\sigma \in \Sigma$ is a controllable action.

A strategy depicts the behavior of the controller, the sequence of actions to be performed, in order to win the game.

*Definition 7 (***Strategy):** In a PTGA called $\mathcal{A}$, a strategy f from the state (s, v) is a partial function from the set of runs in $\mathcal{A}$ starting from (s, v) into the set of controllable actions Σ. A strategy f is a set of decision rules.

A strategy is winning if, when following the decision rules, the controller always wins whatever the environnement does (by the way of non-controllable actions). A strategy f is *state-based* whenever $\forall \varrho\, \varrho' \in all$ the runs $\varrho((s, v), \mathcal{A})$, $last(\varrho) = last(\varrho')$ implies that $f(\varrho) = f(\varrho')$. State-based strategies are also called *memory-less strategies* in the game theory [21].

A controller C_f on a PTGA called $\mathcal{A}$ is a system so that coupled with $\mathcal{A}$ controls the behavior of $\mathcal{A}$ according to a strategy f. We denote by $C_f \;||\; \mathcal{A}$, the PTGA $\mathcal{A}$ controlled by C_f. Controller synthesis distinguishes two kinds of control objectives depending on the property Φ to satisfy: *reachability* and *safety* that can be expressed as following:

- *Reachability:* Given a PTGA $\mathcal{A}$ and a property φ to reach, the controller synthesis is to find a strategy f such that $(C_f \;||\; \mathcal{A}) \models \forall\Diamond\varphi$.
- *Safety:* Given a PTGA $\mathcal{A}$ and a property φ to avoid, the controller synthesis is to find a strategy f such that $(C_f \;||\; \mathcal{A}) \models \forall\Box\neg\varphi$.

Several strategies can potentially fulfill the property Φ to satisfy. We call $C^* = \{C_f \mid (C_f \;||\; \mathcal{A}) \models \Phi\}$ the set of possible controllers. Two of them are of interest:

- *Complete controller:* The complete controller $C_{fmax} \in C^*$ corresponds to the *complete strategy* containing all the possible decision rules. The controller C_{fmax} is related to the complete strategy f_{max} such that $\forall C_f \in C^*, f \subseteq f_{max}$. In a complete strategy, more than one action can be potentially eligible for each location. The controller C_{fmax} needs to choose one action among all the possible ones.
- *Minimal controller:* The minimal controller C_{fmin} corresponds a the *minimal strategy* such that: $\neg\exists C_f \in C^* \mid f \subset f_{min}$. A minimal strategy is a minimal set of decision rules. In that case, there is only one decision rule for a location. A minimal strategy is not necessary unique.

The cost of a strategy f from (s, v) is defined by: $cost(f, (s, v)) = sup\{cost(\varrho \mid \varrho \in Outcome(f, (s, v))\}$. The output of a strategy f, denoted $Outcome((s, v), f)$, from a state (s, v) is a subset of all the runs starting from (s, v) and satisfying f (see complete definition in [12]).

Tools

UPPAAL [22] is a collection of tools dedicated to the analysis of timed automata and its extended formalisms. In all tools, properties are expressed using the logic TCTL. UPPAAL TIGA [23] performs controller synthesis on timed game automata (TGA) with respect to reachability and safety properties. UPPAAL TIGA can provide a complete controller or one of the minimal controllers, randomly chosen among all the possible minimal controllers. When dealing with priced timed automata (PTA), UPPAAL CORA [24] is a model-checking tool that can be used to explore all runs that answer a reachability property in order to retrieve the optimal controller.

PLANNING WITH PTGA

The planning algorithm proposed in this section is applied on a multi-agent system represented by a set of interacting agents, each agent being described by a PTGA. The planning algorithm is looking for the optimal strategy associated to the complete controller as defined above.

Model. The model consists of a network of agents, each agent being described as a PTGA. A model $\mathcal{M}$ is a set of interacting agents A_i with $i \leq n$ and n the number of agents such that $\mathcal{M} = \{A_i :< PTGA_i >\}$. The interaction between the agents is realized by the PTGA according to the CCS-style binary synchronization [25] where exactly two processes synchronize on a matching pair of actions. We assume that a controllable transition and an uncontrollable transition never share the same action label. This synchronization allows interleaving of actions and hand-shake synchronization (on specified actions through communication channels).

Our strategy search algorithm uses the efficient UPPAAL tools that only rely, for computational reasons, on timed game automata (TGA) and priced timed automata (PTA). Given this constraint, we make an assumption in the design of the PTGA: the non-determinism should be only restricted to controllable actions. Given a PTGA as a model, PTA and TGA can be easily derived according to the following definitions.

*Definition 8 (***PTA derived from a PTGA):** A Priced Timed Automaton (PTA) is derived from a PTGA if all the actions $a \in \mathcal{A}ct$, controllable and uncontrollable are replaced by actions with any notion of controllability (or uncontrollability).

We denote by $\oplus$ the operation that add discrete variables to a classical TGA.

*Definition 9 (***TGA derived from a PTGA):** A Timed Game Automaton (TGA) is derived from a PTGA by removing all the cost, both on locations and edges. A variable called $varCost$ can be added to a derived TGA, $\mathcal{A}$, such that $\mathcal{A} \oplus varCost$ enables the definition of the costs on $\mathcal{A}$ from the original PTGA.

Planning Algorithm. The Algorithm *BestStratSearch* computes on a model $\mathcal{M}$ the optimal strategy to reach a specific goal expressed by the system state g. This strategy (called $Strategy$ in the algorithm) corresponds to the complete controller. The algorithm relying on the PTGA part of each agent A_i of the model $\mathcal{M}$ follows two steps: 1) the search for the optimal cost and 2) the computation of the strategy corresponding to this cost. In a first part, the PTGA of each agent is derived in a PTA and the synchronized product is computed. The model-checker UPPAAL CORA is called with the following arguments: the synchronized product of the model and a TCTL formula meaning "Is there a path leading to the state g?". If this path exists, the model-checker returns the optimal cost called $OptCost$. In a second step, the algorithm derives the model in a set of TGA extended with a variable of cost, $varCost$, and computes the synchronized product of TGA. UPPAAL TIGA searches for the strategy corresponding to the previously found optimal cost. The strategy provided by UPPAAL TIGA as a result of this algorithm is the complete strategy associated to the complete controller, a strategy defined as a set of decision rules: $\{(s, \varphi, \sigma)\}$.

Algorithm 1 BestStratSearch

Require:
- $\mathcal{M} = \{A_i =< PTGA_i >\}$, /* $i \in [1, n]$ with n the number of agents */
- $g \leftarrow state\_to\_reach$

1- Search for the best cost
for all $PTGA_i$ **do**
  $PTA_i \leftarrow DerivedPTA(PTGA_i)$
end for
$Prod1 \leftarrow SynchroProduct(\{PTA_i\})$
$Query1 \leftarrow$ "$\exists\Diamond\, g$"
$OptCost \leftarrow UppaalCora(Prod1, Query1)$

2- Search for the controller related to the best cost
for all $PTGA_i$ **do**
  $TGA_i \leftarrow DerivedTGA(PTGA_i) \oplus varCost$
end for
$Prod2 \leftarrow SynchroProduct(\{TGA_i\})$
$\varphi \leftarrow varCost = OptCost$
$Query2 \leftarrow$ "$\exists\Diamond g \wedge \varphi$" /* Is there a run leading to g such that $varCost$ equals the optimal cost $OptCost$ previously computed */
$Strategy \leftarrow UppaalTiga(Prod2, Query2)$

CLASSICAL PLANNING PROBLEM: TRANSPORT DOMAIN

This section presents the translation and adaptation of a classical planning problem: the *Transport Domain* to a multi-agent planning problem defined using the PTGA formalism. After the description of the standard benchmark, we present the enriched version of the domain in order to incorporate the features captured by PTGA.

Domain Description

The *logistics* domain has been a classical planning benchmark for several years at the IPC (International Planning Competition). It models a problem where trucks deliver packages to different locations. This benchmark has been transformed to the *transport-numeric* domain when it became possible to model truck capacities allowing trucks to load multiple packages. The domain is then defined as a set of n *trucks*, p *packages* and l *locations*. The trucks are specified by their capacity of storage, fuel level and current location. Four actions are possible for a truck: loading or unloading packages, driving between two locations and refueling. The packages are characterized by their size and their current location. A truck can move directly from a location to another one if a road exists between these two locations. The classical description of IPC benchmarks is given using PDDL 2.1 [1]. The temporal aspect of this domain is expressed by durative actions that describe the truck behavior.

Transport Domain with PTGA

We translate the Transport Domain into a network of agents where each *truck* and *package* is defined as an agent formalized using a PTGA. We add a third kind of agent: the *mobile repair truck*.

Transport Domain Extension. Compared to the PDDL approach and in order to illustrate the benefit of the PTGA formalism, we enriched the transport domain by adding: explicit timed constraints, cost and uncontrollable actions on agents. In this way, we add a time limit of delivery to each package such that an overrun delay is associated to an extra cost. Cost is updated with the fuel consumption required between two locations. An uncontrollable action has been added on the truck agent in order to simulate an engine breakdown or a run out of gas. We then define a new agent called *Mobile Repair Truck* to complete the model.

The planning problem is then defined as the following: *Given an initial state where the trucks and packages are in their initial locations, what is the strategy to achieve a goal state where the packages are at their final locations and the cost of the trucks fuel consumption is optimal?*.

PTGA Model. The model $\mathcal{M}$ is composed of t *truck* PTGA, n *package* PTGA and one *mobile repair truck* PTGA. The synchronization between trucks and packages agents is performed by the $delivery$ action. The uncontrollable action $breakdown$ realizes a synchronization between the truck and the mobile repair truck PTGA.

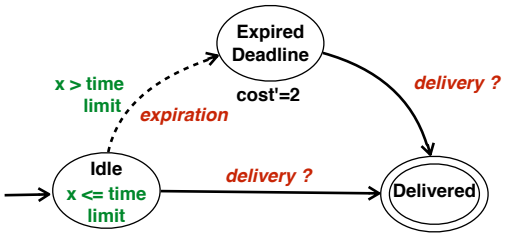

Figure 2. PTGA of a *package* agent.

- **Package agent.** The PTGA that defines each package is presented Figure 2. The synchronized actions between the various agents of the model is expressed using the UPPAAL formalism: *a?* for a received action and *a!* for an emitted action. The package agent is composed of three locations: $Idle$, $Delivered$ and $ExpiredDeadline$. From the initial location $Idle$, the system can move to the $Delivered$ location, if a truck has performed the $delivery!$ action. However each package is associated to a time limit to express the maximum expected time of delivery. The time limit is expressed using a global variable using the UPPAAL formalism. Once in the $ExpiredDeadline$ location, the cost increases of two units per time unit until the system receives the $delivery?$ action from one of the trucks.

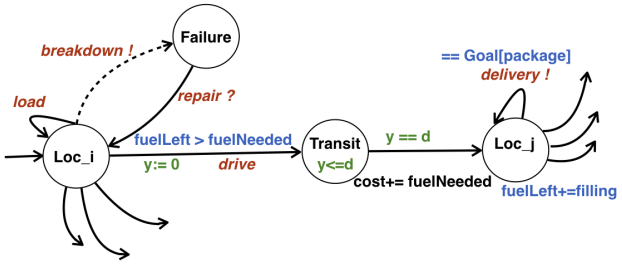

Figure 3. PTGA of a *truck* agent.

- **Truck agent.** The truck agent is defined by a PTGA shown Figure 3. The state space of the automaton is a map of all the reachable locations of the truck. The variable d expresses the travel duration which depends on the distance between a location Loc_i and a destination location Loc_j. Once a package is loaded by the truck ($load$ action), the agent moves to the location $Transit$ if it has enough fuel left to get to the destination Loc_j. The clock y is set to zero when entering to $Transit$. When the local clock y is equal to the travel duration d, the truck moves to the location Loc_j. The cost is incremented by a value related to the fuel consumption needed to join the two locations. When the truck reaches the destination location, the synchronized action $delivery!$ is emitted (and received by the corresponding package agent). In any location Loc_i of the domain, a breakdown or a run out of gas can occur on the truck and the agent moves to the state $Failure$ with a emitted uncontrollable action $breakdown!$.
- **Mobile Repair Truck agent.** The PTGA of the mobile repair truck models the repair and refuel actions of the deficient trucks after the emission of the uncontrollable action $breakdown$. When the repair is performed on the truck, a synchronized action $repair!$ is emitted and the truck can go back to the normal state Loc_i.

The predicates and functions defined in PDDL 2.1 can be expressed using the global or local variables proposed by the UPPAAL tools to complete the theoretical formalism of PTGA. For instance, the fuel quantity between two locations is defined by an array $fuelNeeded[l][l]$ with l the number of locations. The distance between two goals is saved in a variable used to compute the duration d between two locations and exploited in the truck PTGA. The required goals (final locations of the packages) are expressed using a global array variable $Goal[i]$.

Planning Goal

Our planning algorithm expresses the goal using a TCTL formula. Given n, the number of packages, we denote each package agent $Package(i)$ with $i \leq n$. The location $Delivered$ of a package agent $Package(i)$ is then expressed by: $Package(i).Delivered$. For the first part of the algorithm, *Search of the best cost*, the TCTL query is the following:

$Query1 = \exists\Diamond\, Package(1).Delivered == Goal[1]$
$\cdots \wedge Package(n).Delivered == Goal[n]$

Given the optimal cost $optCost$, the second part of the algorithm, *Search for controller*, is computed using the TCTL formula:

$Query2 = Query1 \wedge cost == optCost.$

A global clock can been added to express a time limit for the delivery of all the packages. In that case, for a global clock $timer$ and a time limit t, it is possible to add $\wedge timer == t$ on $Query1$ (and $Query2$) to check whether or not the packages are delivered in time.

EXPERIMENTAL RESULTS

We applied the planning algorithm $BestStratSearch$ on a series of experiments. The various configurations follow the IPC benchmarks and depend on:

- the size of the states space: the number of trucks, the number of packages and the number of locations;
- the goal: the final package locations.

Setup. The experiments were executed on a Linux 64bit, Intel Xeon at 3.20GHz with 5.8GB RAM. The current UPPAAL TIGA version used 8GB. A JAVA program builds automatically the PTGA model presented in the section from the PDDL of the IPC Benchmark.

Results. Table I presents the time response of the planning algorithm based on PTGA. For each experiment we give the number of locations, the number of packages and the number of trucks. The size of the global PTGA (number of locations) is presented and the last two columns give the time response for the two parts of the algorithm: the cost search using the model-checking tool UPPAAL CORA and the complete strategy computation using UPPAAL TIGA.

nb loc.	nb pack.	nb truks	PTGA size (nb loc)	Time Part 1 (Cora)	Time Part 2 (Tiga)
5	2	1	400	3s	1m47s
5	2	2	10000	2m6s	5m38s
10	2	2	73984	5,25s	2m25s
10	3	1	4352	0,94s	4m51s
10	3	2	295936	1m1s	out of mem
10	4	2	1183744	2m33s	out of mem

Table I
RESULTS FOR THE TRANSPORT DOMAIN USING PTGA

The non linearity of the performance results can be explained by the differences in input of the IPC benchmarks. They differ in the mapping of the locations (the road existence) and in the definition of initial and goal package locations. The current version of UPPAAL TIGA uses very limited memory size that explains the memory explosion when the space state size is large.

Differences in goals of multiagent synthesis tools make the direct comparison between studies difficult. However, we compared our approach with MCMAS-SLK, the model-checker dedicated to the synthesis of strategies for multiagent systems. MCMAS-SLK does not manage explicit time and cost as PTGA. To compare the more similar models as possible, we add local variables defining time and cost on the MCMAS-SLK agents definition. Experimentations show that we retrieve the strategy in similar time computation for the first configuration presented Table 1: 2.9s for the search of the cost and 64s for the search of the strategy. However, for the other configurations MCMAS-SLK reaches the memory explosion and no strategy can be provided for a domain having more than five locations, two trucks and two packages. Compared to our approach, MCMAS-SLK provides as a result, one strategy for each agent while the synthesis obtained by our method is a global controller.

CONCLUSION

When considering the planning problem of the growing size and complexity systems, recent studies demonstrated the advantages of using model-checking techniques. A brief survey of work combining model-checking and planning can be found in [26]. In most of these pioneered approaches, temporal properties and temporally flexible plans have not been addressed. More recently studies have proposed temporal models such as timed automata in timeline-based planning [6], [8], [9]. These approaches comes closest to our method since they propose modeling the system as TGA and exploiting the efficiency of an existing tool for model-checking and synthesizing (UPPAAL). However, when considering strategies, the stakeholders are interested in getting the best plan assuming that a cost is associated to each controllable action of the system. Compared to this previous work we propose to use an expressive formalism such as PTGA that allows both the representation of cost and controllable/uncontrollable actions over a simple timed automata. In the multi-agent systems community, the verification of systems against ATL specifications has been widely and theoretically explored. Practical model-checkers, the more popular being JMOCHA [27] and MCMAS [28], have to face the state-space explosion problem. Other work discussed applying epistemic goals [29] to planning but, to our knowledge, they have not yet been applied to any real world planning applications.

In this paper we propose to express interacting agents in the convenient formalism of PTGA which gathers explicit timing constraints and cost on actions. PTGA allows to model systems having non-determinism on controllable actions and offers a manageable description to define the interactions between the agents. If the main benefit of this formalization is its expressiveness, these models are not easy to tackle for realistic planning problems. Our method proposes using recognized and efficient model-checking tools to produce the optimal strategy. We propose a planning algorithm that explains how to derive the unified scheme of PTGA into PTA and TGA and use the recognized tools of the UPPAAL suite. The multiagent temporal planning problem is then solved by a combination of model-checker and controller synthesis. To our knowledge, this is the first approach relying on PTGA, model-checking and controller synthesis, for solving temporal multiagent planning.

We translated the Transport Domain traditionally modeled using PDD2L.1 into the PTGA formalism. We extended the classical example, to become more realistic, by adding timing constraints on delivery and costs to label the potential source of delays. PTGA offer an easy scheme to formalize

the domain and to define a planning algorithm. We would have expected better results when dealing with extended state spaces. However, we can handle larger configurations than MCMAS-SLK. To improve the algorithm's performance, an idea would be to consider the reduction of the memory used by the planning algorithm. A first direction would be to compute, as realized in MIPS[5], the global product using a heuristic approach in order to reduce the blow-up of locations.

REFERENCES

[1] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 61–124, 2003.

[2] J. Kvarnström, "Planning for loosely coupled agents using partial order forward-chaining," in *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*, 2011.

[3] F. Giunchiglia and P. Traverso, "Planning as model checking," in *ECP'99*, 1999, pp. 1–20.

[4] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "Weak, strong, and strong cyclic planning via symbolic model checking," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 35–84, 2003.

[5] S. Edelkamp, "Taming numbers and durations in the model checking integrated planning system," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 195–238, 2003.

[6] L. Khatib, N. Muscettola, and K. Havelund, "Verification of plan models using UPPAAL," in *FAABS 2000*, 2000, pp. 114–122.

[7] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci, "Flexible Timeline-Based Plan Verification," in *KI 2009. German Conference on Artificial Intelligence*, 2009, pp. 49–56.

[8] ——, "Analyzing flexible timeline-based plans," in *ECAI 2010*, 2010, pp. 471–476.

[9] A. Orlandini, A. Finzi, A. Cesta, and S. Fratini, "Tga-based controllers for flexible plan execution," in *KI 2011*, 2011, pp. 233–245.

[10] L. Jezequel and E. Fabre, "Networks of automata with read arcs: a tool for distributed planning," in *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*, 2011, pp. 7012–7017.

[11] J. Tozicka, J. Jakubuv, and A. Komenda, "Generating multi-agent plans by distributed intersection of finite state machines," in *ECAI 2014*, 2014, pp. 1111–1112.

[12] P. Bouyer, F. Cassez, E. Fleury, and K. Larsen, "Optimal strategies in priced timed game automata," in *FSTTCS-2004*. LNCS 3328, 2004, pp. 148–160.

[13] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[14] R. Alur, S. La Torre, and G. Pappas, "Optimal paths in weighted timed automata," in *HSCC-2001*. Springer, 2001, pp. 49–62.

[15] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *Hybrid Systems II, LNCS 999*. Springer, 1995, pp. 1–20.

[16] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," *Information and Computation*, vol. 111, no. 2, pp. 193–244, 1994.

[17] E. Clarke, O. Grumberg, and D. Peled, *Model-Checking*. MIT Press, 2002.

[18] P. Ramadge and W. Wonham, "The control of discrete event systems," *IEEE*, vol. 77, pp. 81–98, 1994.

[19] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller synthesis for timed automata," in *IFAC Symposium on System Structure and Control*. Elsevier Science, 1998, pp. 469–474.

[20] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-checking in dense real-time," *Information and Computation*, vol. 104, no. 1, pp. 2–34, 1993.

[21] H. Gimbert and W. Zielonka, "Games where you can play optimally without any memory," in *CONCUR 2005*. Springer, 2005, pp. 428–442.

[22] K. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *Int. Journal on Software Tools for Technology Transfer*, vol. 1, pp. 134–152, 1997.

[23] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime, "Uppaal-tiga: Time for playing games!" in *CAV-2007*, 2007, pp. 121–125.

[24] G. Behrmann, K. Larsen, and J. Rasmussen, "Priced timed automata: Algorithms and applications," in *FMCO-2004*. Springer-Verlag, 2004, pp. 162–182.

[25] R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[26] S. Bensalem, K. Havelund, , and A. Orlandini, "Verification and validation meet planning and scheduling," *STTT*, vol. 16, no. 1, pp. 1–12, 2014.

[27] R. Alur, L. de Alfaro, R. Grosu, T. Henzinger, M. Kang, C. Kirsch, R. Majumdar, F. Mang, and B.-Y. Wang, "JMOCHA: A model checking tool that exploits design structure," in *ICSE 2001*, 2001, pp. 835–836.

[28] A. Lomuscio, H. Qu, and F. Raimondi, "MCMAS: A model checker for the verification of multi-agent systems," in *CAV 2009*, 2009, pp. 682–688.

[29] G. Aucher and T. Bolander, "Undecidability in epistemic planning," in *IJCAI-2013*, Aug 2013. [Online]. Available: https://hal.inria.fr/hal-00856469