# Searching for Cost-Optimized Strategies: An Agricultural Application

Christine Largouët, Yulong Zhao, Marie-Odile Cordier

▶ **To cite this version:**

Christine Largouët, Yulong Zhao, Marie-Odile Cordier. Searching for Cost-Optimized Strategies: An Agricultural Application. 2nd International Conference, ICDSST 2016, May 2016, Plymouth, United Kingdom. pp.31 - 43, ⟨10.1007/978-3-319-32877-5_3⟩. ⟨hal-01398660⟩

**HAL Id: hal-01398660**

**https://hal.science/hal-01398660v1**

Submitted on 17 Nov 2016

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC BY-NC 4.0 - Attribution - Non-commercial use - International License

# Searching for Cost-Optimized Strategies: An Agricultural Application

Christine Largouet[1] and Yulong Zhao[2] and Marie-Odile Cordier[2]

[1] AGROCAMPUS / IRISA, F-35042 Rennes Cedex, FRANCE,
christine.largouet@irisa.fr,
[2] Université de Rennes 1/ IRISA, F-35000 Rennes Cedex, France

**Abstract.** We consider a system modeled as a set of interacting components evolving along time according to explicit timing constraints. The decision making problem 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* such that the state to reach is expressed using a temporal logic. We have chosen to represent each agent using the formalism of Priced Timed Game Automata (PTGA) and a set of knowledge. PTGA is an extension of Timed Automata that allows the representation of cost on actions and the definition of a goal (to reach or to avoid). This paper describes two algorithms designed to answer the planning problem on a network of agents and proposes practical implementation using model-checking tools that shows promising results on an agricultural application: a grassland based dairy production system.

**Key words:** Decision Support System, Temporal Planning, Optimized Planning, Timed Automata, Model-checking

## 1 Introduction

When managing agro-ecosystems, one of the major decision-aid problems is to find the best management strategy to ensure the health, resilience and diversity of the ecosystem. In this paper we address the problem of finding a strategy for a class of systems that encompasses the characteristics of agro-ecosystems. We investigate the category of non-deterministic systems, defined as a group of interacting agents, each one having its proper dynamics submitted to explicit timing constraints. To improve agro-ecosystem management the model has to include the interactions between environmental, human and ecological subsystems. Time is another fundamental characteristic to express the dynamical features of these systems which are traditionally modeled as differential equations in quantitative approaches. Additionally to interaction and time, we are interested in two novel issues: the representation of unexpected events such as climatic events (hurricane, flooding, heat wave, etc.) and the expression of cost on actions to handle the environmental impact produced by human activities. Searching for a

strategy in these systems consists of choosing and organizing the actions through time in order to achieve an optimal goal, assuming that the actions of the agents have a cost.

To cope with the complexity of systems such as large ecosystems, we propose to use symbolic methods applied with success in model-checking. The paradigm of *planning with model-checking* first proposed by [1, 2] 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. Some other work on timeline-based planning has introduced explicit time on models for space applications [3, 4]. At the same time, in the multi-agent systems community, Alternating-Time Temporal Logic (ATL) has been used for reasoning about agents and their strategies [5] and has been made available in model-checkers such as MCMAS [6]. Recent work has focused on the verification against epistemic logic, or logic for knowledge [7, 8] but one of the key issues remains the state-space explosion problem.

In this paper, we propose to reformulate the planning problems in terms of *model-checking* and *controller synthesis* associated to a temporal logic to express the goal to reach or to avoid. Model-checking and controller synthesis have been widely studied for discrete event systems and more particularly for timed automata [9]. In our approach we propose to represent the system as a network of interacting agents, each agent being described as an extended timed automaton and a set of knowledge. The formalism we chose is Priced Timed Game Automata (PTGA), an extension of timed automata that allows to express timing constraints on states and transitions, costs on actions and the definition of goals [10]. This representation is relevant for multi-agent systems since the interaction can be performed through communicating and synchronizing actions.

On a network of PTGA, we propose an approach to formulate the planning strategies but also to compute the cost of each strategy. The requirement expressed in a temporal logic asks the following question: "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 propose two strategy search algorithms relying on efficient and recognized tools for the model-checking and the synthesis. The first algorithm focuses on searching for the best strategy on a multi-agent system, whereas the second combines controller synthesis and machine learning in order to generate a meta-strategy for a class of similar multi-agent systems.

This approach is applied on grassland management where reasonable practices in farming systems are essential for sustainable agriculture. Land use changes associated with intensive practices such as abusive use of fertilizer or increased number of cutting and grazing activities could have severe impacts on environmental systems. Most of the models, which are designed for agro-ecosystem management, focus on the grassland simulation but can not calculate explicit grassland management strategies. We propose a tool named PaturMata

implementing our method and dedicated to the exploration of the cost-optimal grassland management strategies in a dairy production system.

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. Section 4 describes the network of agents and describes the two algorithms that lead to the strategies. Section 5 briefly describes the application of this approach on ecological systems: grass-based dairy farming. Section 6 concludes and outlines directions for future research.

## 2 Priced Timed Game Automata (PTGA)

**Clock constraints.** Timed Automata [9] 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 $\varphi$, $\varphi_1$ and $\varphi_2$ belong to $\Phi(\mathcal{X})$; $x \in \mathcal{X}$ is a clock and $c \in \mathbb{R}^+$ a constant. Clock constraints are evaluated over clock valuations. A constraint $\varphi$ can be viewed as the set of valuations that satisfy it, hence, we say that $v$ satisfies $\varphi$, denoted $v \vDash \varphi$, if $v \in \varphi$.

**Timed automata and extended formalisms.** In timed automata [9], 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 event 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) [11] 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) [12] is an extension of timed automata where actions on edges are partitioned into controllable and uncontrollable actions. Based on these principles, Priced Timed Game Automata (PTGA) is an extension of both PTA and TGA [10].

### 2.1 PTGA Definition.

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 $(s, l, \varphi, \delta, s')$ such that $e$ connects the location $s \in \mathcal{S}$ to the location $s' \in \mathcal{S}$ on the event label $l \in \mathcal{A}ct$. 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 presents a PTGA having one clock $x$ that is used to define invariants and guards, colored in green in the figure. The goal to reach is the state $s4$. Cost-rates, *cost'*, expresses the cost per time unit while staying in the referred location whereas the cost of triggering a transition is defined by *cost*. Two edges are associated to uncontrollable actions: $a3$ and $a5$ (the edges are shown with a dotted line).

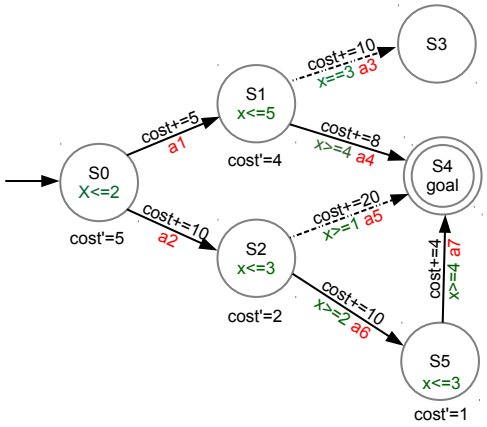


**Fig. 1.** A PTGA having one clock $x$, two uncontrollable actions ($a3$, $a5$), costs on locations and edges and a goal $s4$.

The semantics of a PTGA is 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 $v \in \mathbb{Q}$ is a clock valuation where $v$ satisfies the invariant $\mathcal{I}(s)$ of the location. We write $q \xrightarrow{l}_c q'$ the transition between two states $q$ and $q'$ labeled by the action $l$ while $c$ is the cost of this transition.

From a current state, a PTGA can evolve into a destination location through one of the outgoing edges or remains in its current location while time passes. Consequently, two kinds of transitions are distinguished: discrete and delay transitions. A *discrete transition* is enabled if the timing information of the edge is satisfied (the guard). 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* can be performed if the invariant of the active transi-

tion is satisfied. The cost of a delay transition is computed as the product of the duration and the cost rate of the current location.

A *finite run* $\varrho$ of a PTGA called A is a finite sequence of transitions, starting from the initial state: $q_0 \rightarrow^1 q_1 \rightarrow^2 q_2 \rightarrow^3 \cdots \rightarrow^n q_n$. The cost of $\varrho$, denoted $cost(\varrho)$ is the sum of all the costs along the run.

$$cost(\varrho) = \sum_{1 \leq i \leq n} \begin{cases} \delta \ .\mathcal{P}(s) & \text{if } \rightarrow^i \text{ is a delay transition} \\ \mathcal{P}(e) & \text{if } \rightarrow^i \text{ is a discrete transition} \end{cases}$$

The minimum cost of reaching a location $s$ is the minimum cost of all the finite runs from $s_0$ to $s$

Let us consider the PTGA $\mathcal{A}$ given Figure 1, a possible run in $\mathcal{A}$ is:

$$\varrho : (s_0, 0) \xrightarrow{\delta(2)} (s_0, 2) \xrightarrow{a1} (s_1, 2) \xrightarrow{\delta(1)} (s_1, 3) \xrightarrow{a4} (s_4, 3)$$

The cost of $\varrho$ is $cost(\varrho) = 5 \times 2 + 5 + 8 \times 1 + 15 = 38$

### 2.2 Best-Cost Strategy Problem for PTGA

Given a PTGA and a goal state, the best-cost strategy problem is to find a run, with the optimal cost, starting from the inital state and leading to the goal state. For the example of Figure 1, a key issue is for instance: "Does the controller activate the action $a1$ or $a2$ from the initial state $S0$ to reach in an optimal way the goal $S4$?". If the preferred run takes the edge labelled with $a2$, another question could be: "Is it possible to win whatever the behavior of the environment?".

## 3 Model-checking and Controller Synthesis

To face the large combinatorial size of the state-space, symbolic efficient data structures provide a very compact encoding for large sets of states [13]. When a system is described as a network of timed automata (extended or not), it can be treated using two successful techniques: *model-checking* [14] and *controller synthesis* [15]. These both techniques require the expression of a property to be satisfied and expressed in a temporal logic. The most popular is TCTL [16], a convenient formalism to specify properties over timed automata.

### 3.1 Timed Computation Tree Logic (TCTL)

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.

### 3.2 Model-Checking

*Model-checking* is performed using efficient tools 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 $\varphi$ to be checked, does the model $M$ satisfy $\varphi$?

### 3.3 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 $\Sigma$ 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 $\lambda$ 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).

**Definition 1 (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* $(s, \varphi, \sigma)$ *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 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 $C_f \mid\mid \mathcal{A}$, the PTGA $\mathcal{A}$ controlled by $C_f$. Controller synthesis distinguishes two kinds of control objectives depending on the property $\Phi$ to satisfy: *reachability* and *safety* that can be expressed as following:

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

Several strategies can potentially fulfill the property $\Phi$ to satisfy. We call $C^* = \{C_f \mid (C_f \mid\mid \mathcal{A}) \models \Phi\}$ 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 to 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 [10]).

### 3.4 Tools

Uppaal [17] 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 [18] 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 [19] 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.

## 4 Strategy Search Methods

We propose two methods to compute the planning strategies. The first one called *Best Strategy Search* is looking for the optimal strategy on a model, whereas the second, *Meta-Strategy Search*, provides a meta-strategy for a class of models.

### 4.1 Network of Agents

The *strategy search* methods are applied on multi-agent systems represented by a set of interacting agents, each agent being described by a PTGA and complementary knowledge.

**Knowledge Descriptors:** Agents are enriched with a set of variables called *knowledge descriptors*. The knowledge descriptors define global information on the agent and its behavior. Let $KD$ be a set of knowledge descriptors. Each agent $A_i$ defines a valuation over $KD_i \subseteq KD$ by assigning an integer value to each knowledge descriptor.

**Model:** A model consists of a network of $n$ interacting agents $A_i$. Each agent is composed of two parts: a PTGA, $PTGA_i$ and a set of knowledge descriptors $KD_i$. A model $\mathcal{M}$ is defined as follows: $\mathcal{M} = \{A_i :< PTGA_i, KD_i >\}$ such that:

- $PTGA_i$ is a PTGA defining one agent $A_i$.
- $KD_i$ define the set of valuations for each knowledge descriptor such that $KD_i \subseteq KD$.

The interaction between the agents is realized by the PTGA according to the CSS parallel composition operator [20] that allows interleaving of actions and handshake synchronization (on specified actions through communication channels).

### 4.2 Best Strategy Search Algorithm

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 2 (PTA derived from a PTGA).** *A Priced Timed Automaton (PTA) is derived from a PTGA if all the actions, controllable and uncontrollable, that are labelling the edges are replaced by classical event labels $e \in \mathcal{A}ct$, with $\mathcal{A}ct$ now defined as a set of classical event labels. Events usually label the edges of a timed automaton when any notion of controllability (or uncontrollability) is required.*

**Definition 3 (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.*
*Nota: We denote by $\oplus$ the operation that add discrete variables, as authorized in Uppaal, to a classical TGA. A variable called varCost can be added to a derived TGA, A, such that $A \oplus varCost$ enables the definition of the costs on A from the original PTGA.*

The Algorithm 1 *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 is only based on the PTGA part of each agent $A_i$ of the model $\mathcal{M}$. The principle of the Algorithm 1 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 in input, the synchronized product of the model and a TCTL formula meaning "Is there is path leading to the state $g$?". If this path exists, the model-checker returns the optimal

**Algorithm 1** BestStratSearch

**Require:**
- $\mathcal{M} = \{A_i =< PTGA_i, KD_i >\}$, /* $i \in [1, n]$ 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 */
$Strategy \leftarrow UppaalTiga(Prod2, Query2)$

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: $\{(s, \varphi, \sigma)\}$.

### 4.3 Meta-Strategy Search Algorithm

**Similar Models:** Two models $\mathcal{M}_1$ and $\mathcal{M}_2$ are *similar* if they only differ by the values of their knowledge descriptors $KD_i$ of each agent $A_i$ of the model. It means that $\mathcal{M}_1$ and $\mathcal{M}_2$ are composed of the same $PTGA_i$ and their $KD_i$ define the same variables, only their values differ.

The previous method provides the best and complete strategy for one model $\mathcal{M}$. However, the stakeholder is sometimes more interested by having general rules that work on similar models. This is the case in agro-ecosystem management where systems can not be defined very precisely because parameters are unknown or data unavailable. In such situations, working on a class of similar models is less restrictive and allows the computation of more general decision rules. These generalized rules can be viewed as meta-strategies and are extracted from a set of similar models. However, the generalization of the rules may lead to a minor loss of the optimality. It means that the application of the general-

ized rules do not always give the optimal cost. For agro-ecosystem management, stakeholders are interested by more interpretable rules than too specific ones that are difficult to apply.

Given a goal to achieve, Algorithm 2, *MetaStratSearch*, explains how to compute the meta-strategy on a set of similar models $\mathcal{M}_j$ with $j \leq m$ and $m$ the number of models. Each model is composed of a set of interacting agents $A_i$ with $i \leq n$ and $n$ the number of agents. We propose to compute the optimal strategy for each model $\mathcal{M}_j$ using the previous algorithm *BestStratSearch* for the goal to reach $g$. Each optimal strategy $Strat_j$ provided by this algorithm is associated to the global set of knowledge descriptor values $MKD_j$ such that $MKD_j = \cup_{i\in[1,n]} KD_i$ with $n$ the number of agents of each model $\mathcal{M}_j$. Each couple $(Strat_j, MKD_j)$ is added in a strategy base called $StrategyBase$. Hence, this base of strategies is exploited by a machine learning algorithm to provide the meta-strategy. We chose to apply a classical rule learner algorithm, RIPPER (Repeated Incremental Pruning to Produce Error Reduction), designed to generate rule sets for datasets with many features [21].

**Algorithm 2** MetaStratSearch

**Require:**
- $\{\mathcal{M}_j\}$ /* Similar models */
- $g \leftarrow state\_to\_reach$

**for all** $\mathcal{M}_j$ **do**
  $Strat_j \leftarrow BestStratSearch(\mathcal{M}_j, g)$
  $MKD_j \leftarrow \cup_{i\in[1,n]} KD_i$, /* $n$ the number of agents of $\mathcal{M}_j$ */
  $StrategyBase \xleftarrow{+} (Strat_j, MKD_j)$
**end for**
$MetaStrategy \leftarrow Learner(StrategyBase)$

The meta-strategy provided by this algorithm is expressed as a set of generic rules, valid for the models.

## 5 Agricultural Application

We applied our approach on a grassland based dairy production system using a prototype software named PaturMata. PaturMata comes from the combination of the two words *paturage* (which is a french word meaning pasture) and *automata*. In this context the main issues are to maintain the dairy production at a desired level while limiting the nitrogen fertilization used to increase the grass growth which is known for its environmental damages. PaturMata[1] is a decision support system that models the grass growth under different climate condition, grass consumption by the herd and some agricultural activities like

[1] PaturMata can be freely downloaded from the website: http://people.irisa.fr/Christine.Largouet/paturmata.html

grass cutting and soil fertilization. The model in PaturMata is composed by several agents which are organized in a hierarchy according to their functions (cf. Figure 5):

- Grassland layer: This is the biological model which simulates the grass growth and consumption in each paddock.
- Execution layer: This is the activity model which represents all the agricultural activities including herds' movement, grass cutting and soil fertilization. Each activity is defined by a PTGA. A grazing PTGA represents one herd and simulate its movement from one paddock to another. A grass cutting PTGA and a fertilization PTGA simulate the cutting and the fertilization activity on one specific paddock.
- Controller layer: The different models of this layer simulate the strategies management. One PTGA of the controller layer is associated to a PTGA of the execution layer. To model human decisions, these automata are activated once a day.
- Time layer: One PTGA is in charge of the scheduling.

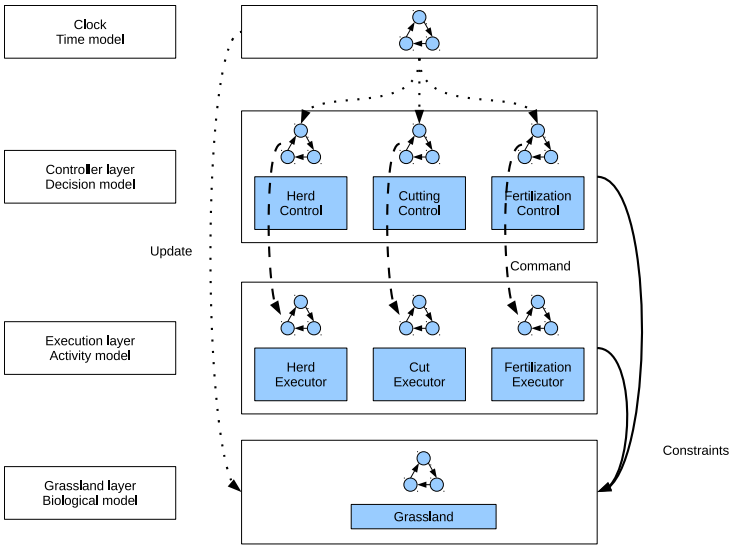


**Fig. 2.** PaturMata architecture in 4 layers composed of PTGA

We carried out one experiment on 100 similar paddock configurations. 50 of them are used to generate fertilization strategy and the other 50 are used to test the strategy. The knowledge descriptor associated to each configuration are: surface of paddocks, distance between paddock and the milking building. We present here some of the decision rules obtained from the *MetaStratSearch* algorithm.

- Fertilize a paddock

  - if the date is between the $15^{th}$ Mars and the $1^{st}$ April, and
  - the distance between the paddock and the milking building is between 400 and 800 meters, and
  - the surface of the paddock is larger than 2.5 ha.
- Fertilize a paddock
  - if the date is between the $15^{th}$ Mars and the $1^{st}$ April, and
  - the surface of the paddock is between 2.0 and 2.2 ha.
- Fertilize a paddock
  - if the date is between the $15^{th}$ Mars and the $1^{st}$ April, and
  - the surface of the paddock is between 1.5 and 1.8 ha, and
  - the distance between the paddock and the milking building is less than 400 meters.

To test the performance, we applied the obtained strategy on the 50 paddock configurations of the test set and noted the cost at the end of simulation. We then launch *BestStratSearch* algorithm on each configuration in order to calculate the optimal cost. A simple comparison showed that the cost obtained by applying the meta-strategy on the model doesn't exceed 20% of the optimal cost on 39 configurations out of 50.

## 6 Concluding Remarks

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. However, the strategy provided by the analysis of one particular multi-agent system is sometimes too specific when regarding the agro-ecosystem management problem. Consequently, we propose a second algorithm to generate a meta-strategy from a class of models. This meta-strategy is more easily interpretable. To our knowledge, this is the first approach combining model-checking, controller synthesis and machine learning. To complete the results presented in this paper, we can report on an experimental evaluation on a realistic agro-ecosystem: a grassland based dairy production system.

One limitation of this work is that the expression of the non-determinism is only effective for controllable actions. This is a consequence of the derivation of PTGA to PTA where all the transitions are considered as controllable. A first perspective to consider for future work would be to study how to avoid this limitation in order to exploit the full potential of the expressiveness of this formalism. To improve the algorithm performance, an idea would be to consider the reduction of the memory used by the search algorithm. This could be realized by using, in the PTGA, clocks dealing with integer values instead of real values.

This granularity of time is sufficient for the management of agro-ecosystems. Finally, a further perspective would be to validate PaturMata through long-term real-life agricultural practices.

## References


1. Giunchiglia, F., Traverso, P.: Planning as model checking. In: ECP'99. (1999) 1–20
2. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artificial Intelligence **147**(1-2) (2003) 35–84
3. Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E.: Analyzing Flexible Timeline-Based Plans. In: ECAI-2010. (2010) 471–476
4. Orlandini, A., Finzi, A., Cesta, A., Fratini, S.: Tga-based controllers for flexible plan execution. In: KI 2011. (2011) 233–245
5. Lomuscio, A., Raimondi, F.: Model checking knowledge, strategies, and games in multi-agent systems. In: AAMAS-2006. (2006) 161–168
6. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A model checker for the verification of multi-agent systems. In: CAV 2009. (2009) 682–688
7. Lomuscio, A., Michaliszyn, J.: Decidability of model checking multi-agent systems against a class of EHS specifications. In: ECAI-2014. (2014) 543–548
8. Huang, X., Van der Meyden, R.: Symbolic model checking epistemic strategy logic. In: AAAI-14, Qubec, Canada (2014)
9. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
10. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.: Optimal strategies in priced timed game automata. In: FSTTCS-2004, LNCS 3328 (2004) 148–160
11. Alur, R., La Torre, S., Pappas, G.: Optimal paths in weighted timed automata. In: HSCC-2001, Springer (2001) 49–62
12. Asarin, E., Maler, O., Pnueli, A.: Symbolic controller synthesis for discrete and timed systems. In: Hybrid Systems II, LNCS 999, Springer (1995) 1–20
13. Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Information and Computation **111**(2) (1994) 193–244
14. Clarke, E., Grumberg, O., Peled, D.: Model-Checking. MIT Press (2002)
15. Ramadge, P., Wonham, W.: The control of discrete event systems. IEEE **77** (1994) 81–98
16. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking in dense real-time. Information and Computation **104**(1) (1993) 2–34
17. Larsen, K., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. Journal on Software Tools for Technology Transfer **1** (1997) 134–152
18. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: Uppaal-tiga: Time for playing games! In: CAV-2007. (2007) 121–125
19. Behrmann, G., Larsen, K., Rasmussen, J.: Priced timed automata: Algorithms and applications. In: FMCO-2004, Springer-Verlag (2004) 162–182
20. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
21. Cohen, W.: Fast effective rule induction. In: In Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann (1995) 115–123