Adapting an MDP planner to time-dependency: case study on a UAV coordination problem

Emmanuel Rachelson
Dept. of Electronic and Computer Engineering
Technical University of Crete
Chania, Crete, Greece
rachelson@intelligence.tuc.gr

Patrick Fabiani
ONERA-DCSD
2, avenue Edouard Belin
F-31055 Toulouse, FRANCE
patrick.fabiani@onera.fr

Frédérick Garcia
INRA-BIA
Chemin de Borde Rouge
F-31326 Castanet-Tolosan, FRANCE
fgarcia@toulouse.inra.fr

Abstract

In order to allow the temporal coordination of two independent communicating agents, one needs to be able to plan in a time-dependent environment. This paper deals with the modeling and solving of such problems through the use of Time-dependent Markov Decision Processes (TiMDPs). We provide an analysis of the TiMDP model and exploit its properties to introduce an improved asynchronous value iteration method. Our approach is evaluated on a UAV temporal coordination problem and on the well-known Mars rover domain.

Introduction

Recent advances in planning under uncertainty are moving the field one step closer to real-world applications. One of the remaining obstacles with the modeling of planning tasks as Markov Decision Processes (MDPs) deals with the integration of temporal extensions of actions and domains. These temporal extensions can consist in action durations (stochastic or deterministic) but also in the modeling of time-dependency in the planning problem. In this paper, we start from a practical problem of mono-agent planning in a time-dependent environment and explore the advantages and drawbacks of using Time-dependent MDPs (TiMDPs, (Boyan and Littman 2001)) to model and solve it.

(Rachelson, Garcia, and Fabiani 2008) has shown that introducing an observable time variable in an MDP model implied using an extended optimality equation which differs from the classical Bellman equation only by a γ^t factor. This allows to derive algorithms which are similar to the ones for standard MDPs. This also allows to unify in a single framework previous algorithms for dealing with continuous variables in MDPs, from (Boyan and Littman 2001), (Feng et al. 2004) and (Li and Littman 2005).

This investigation comes from a practical UAV (unmanned air vehicle) coordination problem. We first present this problem and its implications for single-agent planners. Then, we recall the TiMDP model of Boyan and Littman and show why it is relevant to our problem. The main part of the paper introduces the mono-agent planning algorithm which extends the one of Boyan and Littman and meets with previous results from Feng et al. and Li and Littman. We finally present some optimisation results on the UAV coordination problem and an application of our algorithm on an adapted version of the well-known Mars rover planning domain (Bresina).

The UAV coordination problem

We consider the problem where two independent agents need to coordinate their plans of action without delegating their decision process to each other. This is the case, for instance, when two heterogeneous agents such as an unmanned air vehicle (UAV) and a ground rover need to coordinate their actions online, for a common set of goal tasks. More specifically, suppose the UAV has its own individual mission of patroling over a burning forest in order to observe some areas at certain times; while the rover tries to cross the forest and relies on the UAV's informations about roads for that. In the setup we consider, none of the agents has the opportunity to decide what the other agent will do. However, they are able to communicate when they are close enough, and can exchange goal-related information about their intentions and plans. This information is timed and consists in messages concerning the time-dependent availability of common goals or features of the environment. For instance, the UAV can declare that it will fulfill the task of patroling over an area at a certain time, or it can communicate informations about the evolution of the fire to the rover through a predefined communication protocol.

We consider this problem from the single-agent planning point of view and suppose that there exists a communication protocol which allows the exchange of information between agents. The effect of this communication protocol is to introduce time-dependent goals, rewards and transition dynamics in the planning domain. The problem each agent has to solve is a problem of decision under uncertainty within a time-dependent environment. The uncertainty in the problem comes from the model of each agent's dynamics, from the fire's evolution and from the other agent's actions. On top of that, the exchange of timed-messages between agents introduces time-dependent rewards and goals.

Hence, we focus on the question of generating plans for each agent separately, given the uncertainty in the environment and its time-dependency. We cast this problem as a Time-dependent Markov Decision Process (TiMDP) planning problem for each agent.

Time-dependency and uncertainty in planning

We first recall the definition and common results about Markov Decision Processes. Then we compare different approaches to planning under uncertainty with time-dependency and present the TiMDP model of (Boyan and Littman 2001).

Markov Decision Processes

A Markov Decision Process is given by the tuple $\langle S, A, P, r\rangle$ where S is the set of possible states for the agent, A is a set of available actions among which the agent chooses at each decision epoch, $P(s'|s,a)$ is a Markovian transition model providing the probability of reaching state s' after undertaking action a in s and $r(s,a)$ (or $r(s,a,s')$) is a reward model describing the reward obtained during transition (s,a) (or (s,a,s')).

In order to control the MDP, one usually defines a Markovian *control policy* π, mapping states to actions. Optimizing such a policy then corresponds to defining a criterion measuring the policy's quality and searching for the policy which maximizes the criterion. Common criteria are the discounted or total reward criteria which provide the expected γ-discounted cumulative reward of applying policy π over an infinite horizon, starting in a given state s.

$$V^{\pi}(s) = E\left(\sum_{\delta=0}^{\infty}\gamma^{\delta} r(s_{\delta}, \pi(s_{\delta}))|s_0 = s\right), \quad \gamma \in [0;1]$$

An important issue illustrated by this criterion's definition is that a policy is optimized on the basis of a unit duration for all actions. However, in the problems we wish to consider, the uncertainty often affects both the action outcomes and the sojourn times in successive states.

Time-dependent MDPs

Several extensions to discrete-event dynamic systems exist to take durative actions and time-dependency into account for decision optimization. For instance, one could refer to the SMDP (Howard 1963) or STDN (Wellman, Ford, and Larson 1995) frameworks. Recent deterministic or stochastic modelling efforts have been devoted to such systems (Younes and Simmons 2004; Cushing et al. 2007; Pralet and Verfaillie 2008; Rachelson 2009). In this paper, we will focus on a straightforward way of including time in the state space of an MDP. This approach, introduced by (Boyan and Littman 2001), is known as TiMDPs.

A TiMDP is described by a set of discrete states S and a set of actions A as in a standard MDP. However, whenever one performs action a in s and *at time* t, an *outcome* μ, among the set M of outcomes, is triggered with probability $L(\mu|s,t,a)$. Each outcome is described by a destination state s_{μ} and a duration model P_{μ} characterizing the sojourn time before the transition to s_{μ} triggers. This duration model can either be *relative* — it then provides the probability density function (pdf) on the sojourn time (or transition duration) — or *absolute* — giving the pdf on the transition date. Figure 1 illustrates the previous definition of TiMDPs.

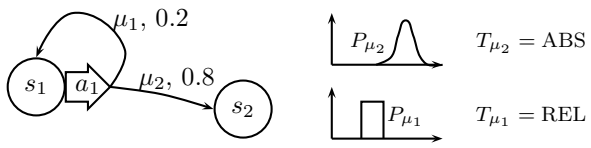

Figure 1: Time-dependent MDP

Optimality equation

(Rachelson, Garcia, and Fabiani 2008) provide insight on the mathematical foundations of the optimality equations for TiMDPs given in (Boyan and Littman 2001) which we recall and comment here:

$$U(\mu,t) = \begin{cases} \int_{-\infty}^{\infty} P_{\mu}(t')[R(\mu,t,t') + V(s'_{\mu},t')]dt' \\ \int_{-\infty}^{\infty} P_{\mu}(t'-t)[R(\mu,t,t') + V(s'_{\mu},t')]dt' \end{cases} \tag{1}$$

$$Q(s,t,a) = \sum_{\mu\in M} L(\mu|s,t,a)\cdot U(\mu,t) \tag{2}$$

$$\overline{V}(s,t) = \max_{a\in A} Q(s,t,a) \tag{3}$$

$$V(s,t) = \sup_{t'\geq t}\left(\int_t^{t'} K(s,\theta)d\theta + \overline{V}(s,t')\right) \tag{4}$$

U is the expected value of outcome μ while Q is the expected value of undertaking a in (s,t). Note that with TiMDPs, policy values have to be manipulated as functions of t in each state s, instead of simple scalar values as in the MDP case. $\overline{V}(s,t)$ is the value of the best action in (s,t) given all the Q functions and, finally, $V(s,t)$ is the expected value function in s if one allows for a specific $wait$ action which leaves the discrete state unchanged and deterministically moves forward in time with a time-dependent reward rate $K(s,t)$. This $wait$ action is a second specificity of TiMDPs: on top of having a state space composed of discrete and continuous components, it also leaves room for the duration parameter τ of a continuous $wait$ action, on top of all the other discrete actions.

The UAV as a TiMDP

We can cast a simple version of the UAV planning problem as a navigation TiMDP. The UAV has to move over a grid of geographical positions and its goals are given as reward rates, corresponding to the importance of patroling over the corresponding cells at specific times of day. These reward rates correspond to the priority of the patrolling task, as defined by the communication protocol. The UAV's available actions are the four movement actions and the $wait$ action, which is the only one providing rewards. The transition model is also made time-dependent based on the weather and wind forecast, affecting the transition probabilities from one cell to the other. Hence, optimizing the movement policy for the agent corresponds to deciding which cell to monitor, given the non-stationary reward and transition models and the possible compromise to make between goals (communication can result in conflicting goals by defining non-zero reward rates on overlapping intervals, hence forcing the

agent to compute the optimal compromise). Figure 2 illustrates this planning problem.

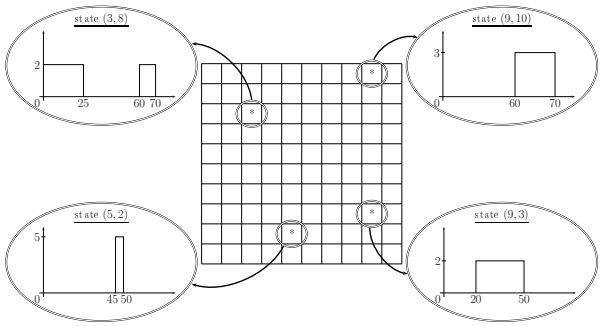

Figure 2: UAV patrol grid

Planning algorithm

(Boyan and Littman 2001) illustrate that with piecewise constant (PWC) L functions, piecewise linear (PWL) reward models and discrete P_μ distributions, one could perform the value iteration Bellman backups inspired by equations 1 to 4 exactly. (Feng et al. 2004) builds on this idea to compute solutions to continuous state MDPs and (Li and Littman 2005) explore the practical resolution of value iteration using PWC functions with the Lazy Approximation algorithm. The approach we present in this paper for TiMDPs relates to the Lazy Approximation scheme. Our results complement and extend (Li and Littman 2005) in several ways. More specifically, in this section, we first present a new general result about planning horizon in TiMDPs, then we focus on the computation of Bellman backups and restate a previous result from the litterature in a more general fashion. This leads to a better understanding of approximation for TiMDP value functions. Finally, we focus on asynchronous value function updates to speed up the resolution. The union of these contributions provides our $TiMDP_{poly}$ algorithm.

Planning horizon vs. temporal horizon

In (Feng et al. 2004) and (Li and Littman 2005), whenever time is included as a state variable, the optimization process is presented as a finite horizon MDP where the value function is optimized for a limited number of consecutive decision epochs. However, this restriction can be avoided by distinguishing between *planning* horizon and *temporal* horizon. The planning horizon, as usually defined, is the number of sequential decision epochs of the agent. Deciding with a finite planning horizon restricts the number of steps an agent can perform. MDPs are usually optimized for an *unbounded* horizon. On the other hand, the *temporal* horizon T corresponds to the initial value of a non-replenishable time resource. Whenever this resource becomes depleted, the process enters an absorbing state providing zero reward and representing the end of the episode. Hence, every episode is defined between times 0 and T and we are mostly interested in the policy and value function between these two times.

For real-life, observable time processes such as TiMDPs, the time-dependency of the problem is only known until a given temporal horizon T. This implies that in most reasonable cases[1], only a finite — but initially unknown — number of actions can be performed before reaching this temporal horizon. This distinction allows us to make the following conjecture:

Conjecture 1. *For observable time problems, such as TiMDPs, with a bounded temporal horizon, there exists a finite planning horizon N such that the N step look-ahead optimized value function is equal to the unbounded-horizon optimal value function on the temporal interval $[0, T]$. One can rephrase this statement: for a TiMDP with temporal horizon T, there exists a value N such that the value function on the $[0, T]$ interval of the best N-step policy is equal to the value function of the best $(N + k)$-step policy. Hence, with π^*_N the optimal N-step value function, we write:*

$$\forall t \in [0, T],\ V^{\pi^*_N}(s, t) = V^{\pi^*_\infty}(s, t)$$

This intuitive conjecture can be proven in a more general framework with specific modelling hypothesis but this result is beyond the scope of this paper. We will simply settle for the proof's intuition which relies on the fact that actions are durative and can be grouped into sequences having a minimal, strictly positive duration.

Consequently, instead of searching for a N-step planning horizon solution, one can search for the stationary value function of an infinite *planning* horizon programming method, on $[0, T]$. This function is a fixed point of the Bellman operator for observable-time MDPs. This brings us to performing value iteration-like Bellman backups on the continuous $V(s, t)$ value functions, where t is the time resource.

Closed-form Bellman backups

We draw inspiration from the value iteration scheme of (Boyan and Littman 2001) which updates a function $V_s(t) = V(s, t)$ every time the discrete state s is updated, using equations 1 to 4. Boyan and Littman show that each of these Bellman backups can be performed analytically, computing $V_{n+1}(s, t)$ in closed-form, based on $V_n(s, t)$, under the four following assumptions:

1. $P_\mu(t')$ and $P_\mu(\tau)$ are discrete distributions.
2. $L(\mu|s, t, a)$ is a PWC function of t.
3. $R(\mu, t, t') = r_t(t) + r_{t'}(t') + r_\tau(t' - t)$.
4. r_t, $r_{t'}$ and r_τ are PWL functions.

Feng et al. adapt this idea to the general case of continuous state MDPs: they keep the same hypothesis on the shape of the functions but extend them to rectangular partitions in a continuous state space. These partitions are described and modified using kd-trees. Li and Littman define a similar algorithm called Lazy Approximation which allows the use of

[1] See (Rachelson, Garcia, and Fabiani 2008; Rachelson 2009) for a complete discussion on the mathematical assumptions necessary for a sound inclusion of time as a state variable in MDPs.

PWC distributions for P_μ and iteratively approximates all successive value functions obtained through Bellman backups by PWC functions. Our contribution analyzes why such PWC, PWL or discrete distributions are well-suited for such problems with continuous variables and builds on this analysis to improve the resolution mechanism.

In order to generalize on the previous hypothesis, we consider the case of piecewise polynomial (PWP) functions and distributions. We write $\mathcal{P}_m$ the set of PWP functions of maximum degree m and:

1. $P_\mu \in \mathcal{P}_A$
2. $r_t, r_{t'}, r_\tau \in \mathcal{P}_B$
3. $L \in \mathcal{P}_C$

Then, we can write:

Result 1 (Value function degree)**.** *The sequence of value functions issued by the application of the Bellman backups corresponding to equations 1 to 4 has the degree:*

$$d^\circ(V_n) = B + n(A + C + 1) \tag{5}$$

where n is the iteration number.

Proof. This follows from establishing that the convolution of two PWP functions of degree m and n yields a PWP of degree $m+n+1$. Then, taking equations 1 to 4 step by step and observing the functions' degrees provides the result. □

This leads to the immediate consequence:

Corollary. *In order to have a closed-form solution of the Bellman equation throughout the value iterations, one needs to insure $A + C = -1$.*

While this is not possible for purely PWP functions, one needs to remember that A is indeed the degree of a PWP distribution (and not a PWP function). Analyzing equations 1 to 4 shows that if P_μ is a discrete distribution, then it behaves as a "$\mathcal{P}_{-1}$" distribution with respect to the convolution operation of equation 1. Then, one can reach the $A + C = -1$ condition with $A = -1$ and $C = 0$. Hence, this result shows that Boyan and Littman exact closed-form resolution of TiMDPs cannot be directly extended to PWP distributions.

However, this investigation also provides some insight on how to perform the analytical operations of the Bellman backups. One can still implement these operations, they just do not result in a closed-form solution anymore. Consequently, any projection scheme on a lower PWP degree function space which provides error bounds on the approximation error could fit an approximate value iteration method for TiMDPs (and more generally for continuous state MDPs).

Another side-consequence which becomes almost immediate through the observation of equations 1 to 4 is that the exact resolution conditions of (Boyan and Littman 2001) can only be slightly extended. The previous result imposed conditions on the degrees A and C, the next one bounds the value of B:

Result 2 (Exact closed-form resolution conditions)**.** *In order to analytically perform all operations for Bellman backups, one also needs to exactly compute the roots of PWP functions (more specifically the intersection of PWP functions in equation 3) and thus, the exact resolution conditions for PWP representations are:*

$$A = -1,\ C = 0,\ B \leq 4$$

Similarly to the previous remarks, having a reward model's degree larger than 4 does not imply that one cannot perform the Bellman backups. However, these backups will become approximate because of the approximate root finding procedure necessary to solve equation 3.

Finally, in practice, the convolution, multiplication, summation and intersections operations of equations 1 to 4 imply subdividing the definition intervals of the different PWP functions to obtain the next V_{n+1} value function:

Conjecture 2. *Bellman backups on PWP representations result in a linear increase in the number of pieces necessary to describe the value function, even in the exact resolution case.*

Consequently, in practice, to avoid numerical inconsistencies such as intervals of length tending to zero, one necessarily needs to make use of approximation at some point. Moreover, experience shows that these very small intervals often have very close values and, hence, can be easily merged into larger intervals if one allows for an L_∞-bounded approximation scheme.

This initial analysis of the Bellman backup operator for TiMDPs provides us with a good generalization of the results presented in (Feng et al. 2004) and (Li and Littman 2005). Based on these results, we are still able to perform analytical Bellman backups on the extended PWP representations and need a projection scheme in order to keep a practical shape to the obtained value functions. This generalizes what the Lazy Approximation method does with PWC functions and opens the door to richer representations such as spline approximation (Ahlberg, Nielson, and Walsh 1967).

An approximation method for value functions

We implemented a specific case of approximation method in order to interpolate any PWP of degree k by a PWP of fixed degree l with a low number of definition intervals. In order to be able to use the standard error bounds for value iteration, one needs to bound the L_∞ approximation error of the projection method.

Finding an optimal interpolation of a continuous function by a PWP in terms of number of intervals and degree is a difficult problem to solve. Thus, our method implements the sub-optimal (but efficient) following ϵ-approximation scheme. This methods proceeds in two steps: first it considers a single "piece" of the input function p_{in}, *ie.* an interval over which p_{in} has a continuous polynomial definition. Over this interval, which we write $\mathcal{I}$, it calls an interpolation method $\texttt{interpolate}(p_{in}, \mathcal{I}, l, \epsilon)$ where l is the maximum degree allowed and ϵ the L_∞ approximation tolerance. This method computes an interpolation polynomial of degree at most l over $\mathcal{I}$ and outputs it along with the largest approximation error e_{max} and the abscissa t_{max} where e_{max} is reached. If e_{max} is smaller than ϵ, then the output PWP p_{out} is set to the interpolation polynomial over

$\mathcal{I}$ and the algorithm moves on to the second phase. Else, $\mathcal{I}$'s upper bound is shifted to t_{max} and the process restarts. Once a suitable interpolation has been found on the reduced $\mathcal{I}$, then a new $\mathcal{I}$ is defined by taking the uncovered part or the initial $\mathcal{I}$ and the same method is applied until the algorithm reaches the upper bound of the initial $\mathcal{I}$. This approximation scheme is illustrated with maximum degree $l = 1$ on fig. 3.

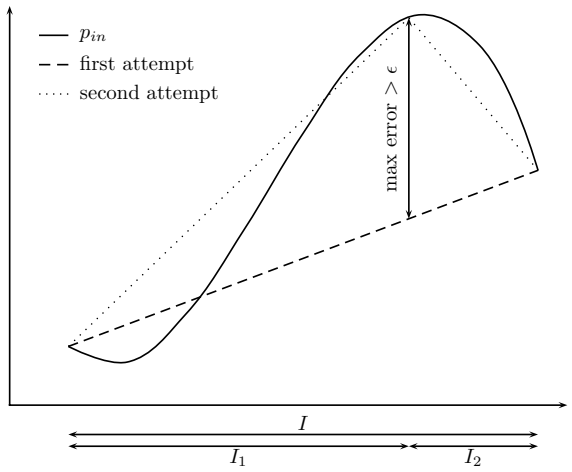

Figure 3: Refinement phase of the projection method

Then the second phase of the approximation allows to keep the number of intervals low by trying to merge any consecutive intervals after the end of $\mathcal{I}$ into a larger interval using an interpolating PWP of degree l and an approximation error of ϵ at most. If it fails, it returns to the first phase.

This procedure is repeated until T is reached and an interpolation PWP p_{out} is output. Our method finally has the following nice properties:

1. Since the `interpolate` method is free, it is easy to preserve the continuity of the function for $l \geq 1$ (and eventually its smoothness if l is large enough).
2. It always outputs a PWP function $p_{out} \in \mathcal{P}_l$.
3. The output function has a suboptimal (but good) number of intervals.
4. The approximation error ϵ is controlable and one has the guarantee that: $\|p_{in} - p_{out}\|_\infty \leq \epsilon$.
5. Experience showed that the output function was close to an optimal approximation in terms of intervals number with a significant reduction in computational effort.

Ordering Bellman backups

With the analytical computation of Bellman backups and the previous approximation method, one has a straightforward way of performing value iteration on TiMDPs. A third contribution of this paper on solving TiMDPs builds on the properties implied by conjecture 1. Since the value function we are looking for corresponds to the fixed point of an infinite horizon dynamic programming operator, we can avoid updating each state sequentially as in simple value iteration. Instead, we can try to reduce the computational effort induced by PWP operations by ordering the states in which we perform the Bellman backups such that convergence to an ϵ-optimal value function is quick.

There are many ways of implementing such an ordering method, all of them relate to the general conditions of asynchronous value iteration (Bertsekas and Tsitsiklis 1996). As long as every state is chosen infinitely often for update (as the number of state updates tends to ∞) one can show that the value function converges to the optimal value function. Hence, the state ordering for Bellman backups does only impact the convergence speed to the optimal value function. A well-known and efficient method for ordering Bellman backups in standard MDPs is *Prioritized Sweeping* (Moore and Atkeson 1993).

Prioritized Sweeping maintains a priority list on states. The first state in the list (the highest priority) is the next state to update. Every time a state's value is updated, the amplitude of the value change ΔV is computed. Parent states of s are states from which there exists an action allowing to reach s in one step. When s is updated, a priority of $P(s|s_-, a_-)\Delta V$ is applied to each parent state s_- (with a_- the action allowing to reach s with highest priority). If this priority is higher than another previously defined priority, it replaces it. Else, this operation has no effect. Thus, every time a state's value is updated, the amplitude of this update is propagated to neighbour states and the priority list is updated too. This method results in propagating the largest value function changes in priority throughout the state space.

We adapted the prioritized sweeping scheme to TiMDPs in order to minimize the number of Bellman backups necessary before reaching an ϵ-optimal value function. However, prioritized sweeping is defined for discrete state spaces MDPs and we are dealing here with a hybrid state space and with functions $V_s(t)$ in every discrete state instead of values. Hence, the prioritized sweeping scheme needs to be adapted to these hybrid state spaces. On top of that, in the algorithm of (Moore and Atkeson 1993), the ΔV quantity is computed once and propagated by multiplying it with the corresponding parent transition's probability. For TiMDPs, because of equations 1 and 2, this priority computation is not as simple and we have to turn back to calculating separately a ΔQ for each transition.

The adapted prioritized sweeping algorithm for TiMDPs is presented in algorithm 1 and commented below.

Our version of prioritized sweeping for TiMDPs maintains both a record of the value functions $V_s(t)$ and of the Q-functions $Q_{s,a}(t)$. When the first state in the priority queue is selected for update, it is removed from the queue. We call it s'. The `BellmanBackup()` procedure applies equations 3 and 4 in order to update $\overline{V}_{s'}(t)$ and $V_{s'}(t)$ with respect to their child transitions' $Q_{s',a}(t)$ functions. Then, as soon as s' is updated, all parent Q-functions are also updated, hence insuring consistency between the Q and their destination state's V functions. This is done through the `BellmanUpdate()` procedure which computes the result of equations 1 and 2. Note that if memory is not an issue, one can also keep track of the U-functions to increase the calculation's efficiency. Finally, the priority affected to the par-

Algorithm 1: Prioritized Sweeping for TMDPs

```
Init: V ← 0
Init: priority_queue ← UnprioritizedVI()
Init: continue = true
while continue = true do
    while priority_queue ≠ ∅ do
        Remove the top state s' from priority_queue.
        V_{s'}(t).BellmanBackup()
        foreach (s, a) ∈ predecessors(s') do
            Q_{s,a}(t).BellmanUpdate()
            Prio(s, a) = ‖Q_{s,a}(t) − Q^{old}_{s,a}(t)‖_∞
            if Prio(s, a) > ε and
            Prio(s, a) > Prio(s) then
                Insert s in priority_queue with
                Prio(s) = Prio(s, a)
    priority_queue ← UnprioritizedVI()
    if max_priority(priority_queue) < ε then
        Either take a smaller ε or set continue = false.
```

ent states correspond to the largest amplitude of Q-function change and the priority queue is updated.

This process is repeated until no priority is larger than a given ϵ. In algorithm 1, only states with priorities larger than ϵ are inserted in the priority queue and the procedure stops whenever this queue becomes empty.

This priority queue can be initialized by hand if one has some prior knowledge about the problem's structure, or it can be built from a single pass of unprioritized value iteration through the state space (the `UnprioritizedVI()` procedure). In order to insure that no states are left out during the optimization process, whenever the priority queue becomes empty, a new pass of unprioritized value iteration is performed. If this pass only generates priorities lower than ϵ, the algorithm terminates. Upon termination of the algorithm, one has the guarantee that the global value function $V(s,t)$ found is at least ϵ-optimal for the TiMDP problem.

The $TiMDP_{poly}$ algorithm

By putting together the three separate improvements presented above, one obtains a more general algorithm which we call $TiMDP_{poly}$. This algorithm makes use of:

1. Analytical (eventually approximate) Bellman Backup calculations on PWP representations.
2. PWP degree reduction and interval simplification.
3. Adapted prioritized sweeping for hybrid variables as presented on algorithm 1.

$TiMDP_{poly}$ generalizes on common features with Lazy Approximation and the algorithm of (Feng et al. 2004). Like these methods, it makes use of analytical operations for performing Bellman backups; however, it extends the class of functions these methods were able to deal with to the general case of PWP representations. The simplification phase was already present in the previous methods but was separate from the optimization and it could only take PWC functions into account: $TiMDP_{poly}$ generalizes it to general PWP functions and merges the operations of degree reduction and interval simplification. Finally, $TiMDP_{poly}$ introduces and justifies the use of the adapted prioritized sweeping method for the hybrid state space at hand, which reduces the computational effort required for value function update and improves convergence speed to the optimal value function.

Moreover, $TiMDP_{poly}$ takes into account the specific $wait$ action of TiMDPs which was left out by the previous methods. It optimizes the continuous parameter of this action, the same way it computes the results of value function backups. Hence, it is important to recall that TiMDPs are a specific class of MDPs with both a hybrid state *and* action space. This shades a different light upon $TiMDP_{poly}$ as a solving algorithm for a very specific class of hybrid, parametric actions and hybrid state problems.

Experiments

The $TiMDP_{poly}$ algorithm was implemented as a general purpose C++ library for TiMDP problem specification and solving. This implementation relies on a specific PWP function C++ library called *POLYTOOLS* which we developed in order to handle the PWP functions and operations necessary to the analytical computation of Bellman backups. We tested $TiMDP_{poly}$ on two benchmarks: the first is the UAV problem presented earlier, the second is an adapted version of the Mars rover domain (Bresina et al. 2002).

Prioritizing is useful: we compared a run of $TiMDP_{poly}$ with an optimization scheme using the same Bellman backups but with unprioritized value iteration, as in (Boyan and Littman 2001). The temporal horizon T was 100. The precision on t imposed for approximate PWP operations was 10^{-3}, the approximation tolerance for PWP degree reduction and simplification was $5 \cdot 10^{-2}$ and the algorithm was stopped when all priorities became smaller than 10^{-1} which corresponds to $\sim 10^{-3}$ times the highest priority encountered during optimization. The value function was initialized to zero in all cases. The results appear in table 1. Prioritizing the states upon which we perform Bellman backups retains its property of convergence acceleration. Hence, this provides experimental validation to the way we compute the priorities on discrete states and confirms the interest of prioritizing in $TiMDP_{poly}$. An interesting feature to note is that the UAV problem is defined on a 10×10 grid, so the number of discrete states is 100; since $TiMDP_{poly}$ finishes the optimization after performing 531 Bellman backups in individual states, it means it only needs visiting each state 5.31 times on average to decrease the update maximum priority by a factor 10^{-3}. This confirms the fact that asynchronously performing the Bellman backups in order to structure the dynamic programming optimization saves a lot on computational resources.

Algorithm	number of Bellman backups
$TiMDP_{poly}$	531
Value Iteration	33000

Table 1: Number of state Bellman backups

Impact of the PWP functions' maximum degree: we tried defining different shapes of PWP for the transition and duration models of the problem to analyze how the degree of the model's functions affects the computational resources needed for resolution. Preliminary results show that while the number of required Bellman backups remained low, the computation became more complex with higher degrees. However, this is not a final conclusion and it relies more on the current *POLYTOOLS* implementation than on $TiMDP_{poly}$ itself. To illustrate this, we can compare the resolution of the UAV problem with discrete P_μ and with piecewise linear P_μ. In the first case, the resolution required 531 Bellman backups and 13.90 seconds while in the second case, the 824 Bellman backups required 740.17 seconds [2].

Approximation is necessary: we tried to run $TiMDP_{poly}$ in the exact resolution case, without the interval simplification scheme. On the simple but non-trivial example of the UAV patrol problem, the number of intervals in value function definition increased steadily and eventually caused numerical problems before the value function converged to the optimal value function. This leads us to conclude — from experience — that even within the exact resolution conditions, for non-trivial problems, approximation through interval simplification is necessary, thus confirming conjecture 2.

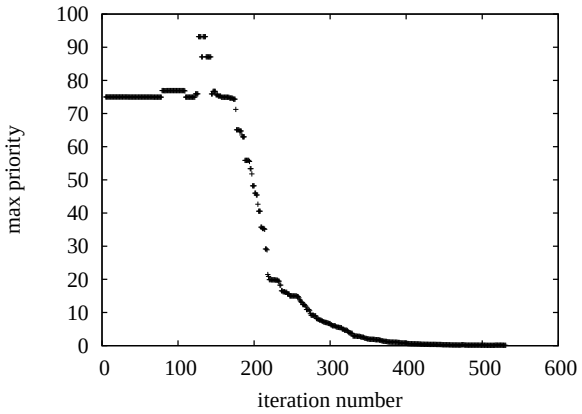

Figure 4: Maximum priority vs. iteration number

Policy quality evolution: figure 4 shows the evolution of the maximum priority with the Bellman backup number. One can use this maximum priority as an *asymptotic, approximate* measure of the current policy's quality since the priorities are related to the Bellman error. This figure illustrates that priorities follow a global decreasing trend until an ϵ-optimal value function is reached. However, it also shows that this evolution is not necessarily monotonous. A detailed discussion concerning the monotonicity of the maximum priority decrease can be found in (Rachelson 2009). Nevertheless, one can distinguish three phases on this graph: first, the large value changes are propagated to the whole state space as fast as possible (backups 1 to $\sim$170) then the global shape of the value function is found by making the compromise between Q functions (iteration $\sim$171 to $\sim$300) and finally, the value function is refined in order to get close to the optimal value function. Figure 5 illustrates the shape of the different functions found, using the example of the 158^{th} Bellman backup, occurring in state $(5, 4)$.

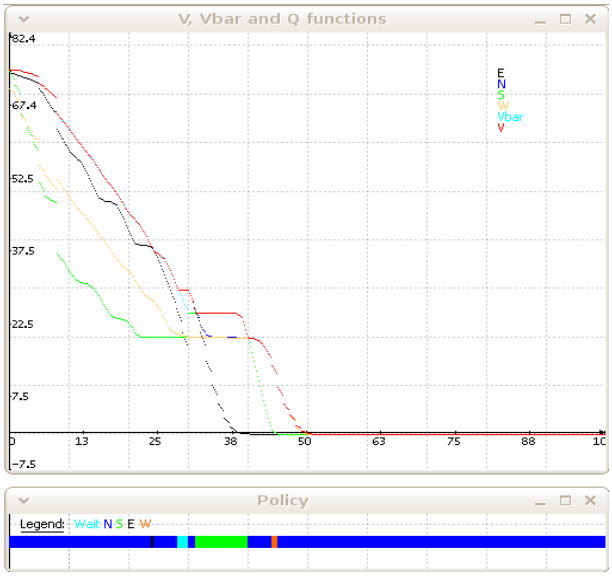

Figure 5: Illustrating V, $\overline{V}$ and Q, backup #158, state $(5, 4)$

In the second domain, an autonomous rover needs to plan its course of action in order to navigate, collect ground samples and take pictures on Mars. Actions are durative and uncertain and the model is time-dependent. The rover can navigate between nodes of a navigation graph, take pictures, recharge its batteries, etc. Figure 6 illustrates the value function and optimal policy found in a given location ($p = 3$), when none of the mission goals have been completed yet, as a function of the current time and energy level.

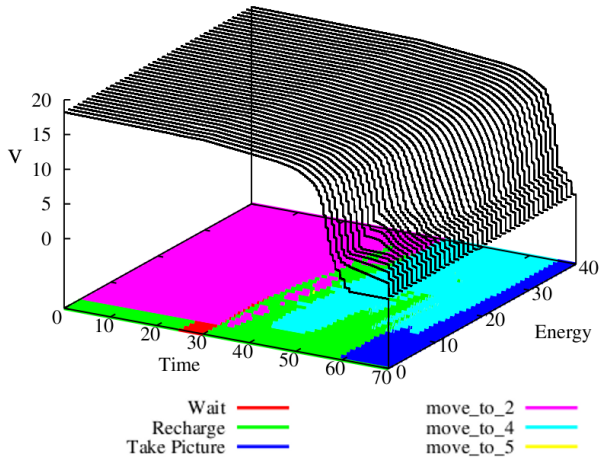

Figure 6: Rover's value function and policy

This adapted version of the Mars rover problem has 1968

[2] experiments were ran on a 1.8 Ghz PC with 1GB of RAM

discrete states and one continuous variable (time) ranging from 0 to 70. It also has between one and six discrete actions available per state, plus the continuous $wait$ action. $TiMDP_{poly}$ required about 35000 Bellman backups on functions to find the ϵ-optimal value function and took 1690 seconds. These numbers can be related to the number of state updates necessary if one discretized the time variable on a unit-duration basis and solved the approximate problem using finite-horizon dynamic programming. Such a process would require 139728 Bellman backups on state values. On the other hand, with 35000 Bellman backups on functions, we obtain the continuous-time solution, which can be a good tradeoff if the resolution needed for the time variable is even smaller than one.

From figure 6, one can remark that, even though there is a gain in considering a continuous time, it might be desirable to obtain a similar gain on the reasoning concerning the energy variable. In particular, it would allow for a more compact representation and reasoning about the plateau observed at the left of the time axis. However, hypercube representations might not be the most adapted here since the edge of the plateau is not parallel to the energy axis: it seems to follow a straight line, so a triangular representation, for example, might be more flexible and more compact in order to describe the different parts of the value function. This takes the problem to the more general case of continuous variable partitionning in MDPs and constitutes an important issue for future research.

Conclusion

In this paper, we introduced a new algorithm called $TiMDP_{poly}$ designed to find control policies for the observable time, hybrid state and action MDPs called TiMDPs. This algorithm builds on the TiMDP framework introduced in (Boyan and Littman 2001) and is related to previous work from (Feng et al. 2004; Li and Littman 2005). $TiMDP_{poly}$ introduces three distinct features:

1. It generalizes the possibility of performing analytically the Bellman backup operations on piecewise polynomial representations.
2. It introduces an approximation scheme based on degree reduction and interval simplification, guaranteeing L_∞ bounds on the approximation error. This approximation scheme makes methods as value iteration possible and improve their efficiency by keeping the number of definition intervals low.
3. It performs an asynchronous value iteration optimization by assigning priorities to discrete state updates, in order to let the value function converge faster to the optimal V^*.

Future work implies writing complexity bounds for the approximation method and relating them to the general algorithm's complexity. This might help finding good compromises between PWP degree and number of pieces. Generalization of this work implies application to spline functions in general. Experimental work on $TiMDP_{poly}$ also implies testing on a full version of the Mars rover domain closer to the one tested in (Li and Littman 2005) for instance.

Going from one continuous variable to several can be done as in (Feng et al. 2004) or (Li and Littman 2005). Then, more general questions arise such as how to define partitions in the state space. (Feng et al. 2004) used kd-trees to define hypercubes but other partitioning methods, such as the Kuhn triangulations of (Munos and Moore 2002), might be more relevant and flexible. Another question is to determine how these representations scale to the curse of dimensionality, when the number of variables in the problem increases. Finally, adapting the asynchronous prioritization scheme for value iteration is almost straightforward and might help in the general case of continuous state spaces.

References

[1] Ahlberg, J. H.; Nielson, E. N.; and Walsh, J. L. 1967. *The Theory of Spline Functions and Their Applications*. Academic Press, New York.

[2] Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

[3] Boyan, J. A., and Littman, M. L. 2001. Exact Solutions to Time Dependent MDPs. *Advances in Neural Information Processing Systems* 13:1026–1032.

[4] Bresina, J.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; and Washington, R. 2002. Planning under Continuous Time and Resource Uncertainty: a Challenge for AI. In *UAI*.

[5] Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is Temporal Planning Really Temporal? In *International Conference on Automated Planning and Scheduling*.

[6] Feng, Z.; Dearden, R.; Meuleau, N.; and Washington, R. 2004. Dynamic Programming for Structured Continuous Markov Decision Problems. In *UAI*.

[7] Howard, R. A. 1963. Semi-Markovian Decision Processes. In *34th Session of the International Statistical Institute*.

[8] Li, L., and Littman, M. L. 2005. Lazy Approximation for Solving Continuous Finite-Horizon MDPs. In *National Conference on Artificial Intelligence*.

[9] Moore, A. W., and Atkeson, C. G. 1993. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning Journal* 13(1):103–105.

[10] Munos, R., and Moore, A. W. 2002. Variable Resolution Discretization in Optimal Control. *Machine Learning Journal* 49(2-3):291–323.

[11] Pralet, C., and Verfaillie, G. 2008. Using Constraint Networks on Timelines to Model and Solve Planning and Scheduling Problems. In *ICAPS*.

[12] Rachelson, E.; Garcia, F.; and Fabiani, P. 2008. Extending the Bellman Equation for MDP to Continuous Actions and Continuous Time in the Discounted Case. In *ISAIM*.

[13] Rachelson, E. 2009. *Temporal Markov Decision Problems — Formalization and Resolution*. Ph.D. Dissertation, University of Toulouse, France.

[14] Wellman, M.; Ford, M.; and Larson, K. 1995. Path Planning under Time-Dependent Uncertainty. In *Conference on Uncertainty in Artificial Intelligence*, 532–539.

[15] Younes, H. L. S., and Simmons, R. G. 2004. Solving Generalized Semi-Markov Decision Processes using Continuous Phase-Type Distributions. In *AAAI*.