

Anytime Planning in Hybrid Domains using Regression, Forward Sampling and Local Backups

Florent Teichteil-Königsbuch and Guillaume Infantes

ONERA-CERT 2, av. Edouard Belin BP 4025 - 31055 Toulouse Cedex 4, France
{florent.teichteil,guillaume.infantes}@onera.fr

Abstract

In most real-world problems of decision under uncertainty, continuous state variables (for instance: fuel level) are to be taken into account. The Hybrid Markov Decision Processes (HMDP) framework allows one to directly model both continuous and discrete components of the state space. The main issues with HMDPs are: how to compactly represent the value function (reward expectation) and how to efficiently update it (perform Bellman backup) on the hybrid state space? In the general case, closed-form computation for the update is intractable without model simplification, especially when updating the value function over the whole state space, and any compact representation of the value function has to be an approximation of the result of this update.

We present *HRTDP*, an algorithm which lifts any restriction on the transition function representation, while allowing compact representation of the value function. This is done using asynchronous backups on only some parts of the hybrid state space, approximated with sampling techniques, and using state-of-the-art regression techniques to obtain a compact (while precise) representation for the value function. It also benefits from recent advances in discrete forward stochastic planning: partial policy construction relying on domain-independent relaxation heuristic, and extends this framework to hybrid domains. We demonstrate the relevance of our approach on instances of two large hybrid planning domains with complex transition functions: search-and-rescue and airport ground traffic management.

Introduction

Despite recent advances, autonomous decision-making is still a challenge in real-world scenarios. One cause is that such scenarios involve intrinsic non-determinism in large spaces. One solution for building autonomous controllers is to compute off-line partial policies instead of plans, that is a mapping of a subset of the possible states to actions. Markov Decision Processes (MDPs) have been recently shown to be able to deal with large state spaces with probabilistic modelling of the non-determinism, particularly using forward heuristic search in order to prune the search space. But another major issue to handle real-world problems is the continuous nature of some features. Typically, resources of an agent cannot be handled natively in the MDP framework. Generally, such features of the domain are discretized, making the size of the state space of the MDP exponentially

larger, while loosing precision in the representation of these continuous features.

Hybrid MDPs (i.e. definition of the state space in form of a cross-product of discrete state variables and continuous state variables) allow a compact representation of such large MDPs with continuous features, as we will show in next section. Within this framework, real-world problems can be easily modelled, we present two examples of such modeling in the next section. As far as the authors know, there have been surprisingly few work in the domain of hybrid MDPs, as we will see. The main obstacles for solving HMDPs are: the representation of the expectation of rewards (the value function) over the hybrid state space (because of the infinite support); and the calculation of the update of the value function, the Bellman backups (because of the hybrid integration over dependent continuous and discrete variables). In order to solve such problems, one needs to have an approximation of the value function, and a way to compute the updates efficiently. Previous approaches propose to restrict possible transition functions and value function representations in order to be able to solve both these problems.

The main contribution of this work is to lift such restriction by using asynchronous updates of the value function. This is done by adapting (discrete) MDP forward heuristic sampling over trajectories, and representing compactly the value function using machine-learning techniques. We will explain what is needed for such adaptation (third section) and will show that recent advances in the domain of machine learning (particularly classification and regression) can be used in the core of the algorithm, and what kind of approximations can be used for the Bellman backup itself. It is worth noting that while using machine learning techniques for representation of some functions in our algorithm, we do not learn the policy in the sense of reinforcement learning, we prefer to use “classical” forward heuristic search and asynchronous Bellman backup. We will show the effectiveness of our approach in last section.

Solving Hybrid Markov Decision Processes

Definition 1. A Hybrid Markov Decision Process (HMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ where:

- $\mathcal{S} = \bigotimes_{i=1}^p \mathcal{V}_i^c \times \bigotimes_{i=1}^q \mathcal{V}_i^d$ is a cartesian product of p continuous state variables and q discrete state variables.

A state $s \in \mathcal{S}$ is an instantiation of all state variables: $s = (v_1^c, \dots, v_p^c, v_1^d, \dots, v_q^d)$.

- \mathcal{A} is the set of actions ; we assume all actions are enumerated and discrete. Each action $a \in \mathcal{A}$ is applicable over a set of states \mathcal{S}_a .
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function which models the probability of action outcomes ; it is a probability density function over continuous variables whose integration gives the probability of discrete variables.
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function ; $R(s, a, s')$ is the reward earned when going to state s' from state s by applying action a .

We assume all continuous variable domains are in any subsets of \mathbb{R} , and the action space is discrete. The optimization of a HMDP consists in computing the policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ which maximizes the mean over all possible state trajectories of the discounted accumulated rewards over an infinite time horizon: $\pi^*(s) = \arg \max_{\pi \in \mathcal{A}^{\mathcal{S}}} E \left[\sum_{t=0}^{+\infty} \gamma^t r_t \mid s_0, \pi \right]$ where $0 < \gamma < 1$ is the *discount factor*.

Related Work

As we will show in example domains, the main difficulty comes from the Bellman backups in hybrid domains. As the state space is composed with both discrete and hybrid components, the form of the exact summation is intractable in the general case. Another issue is the representation of the value function itself, which can be of any form, thus needs to be approximated in some way. The authors are aware of two main types of algorithms for solving hybrid Markov decision processes. The first one is the HALP framework (Kveton, Hauskrecht, and Guestrin 2006) ; while the other one is the adaptation of AO^* to hybrid domains, named HAO^* (Meuleau et al. 2009).

The HALP framework pioneered the domain of hybrid MDPs using approximate linear programming by projecting value function on a finite set of basis functions, discretizing continuous state variables and using mixture of beta transition models only. The linear programming formulation of the MDP uses an infinite set of linear constraints for each (s, a) pair. The main criticisms for this approach are that the authors use a set of “basis” functions on which they project the value function, they give severe restrictions on the form of the state transitions functions and on the basis functions in order to perform exact backups, and they truncate the infinite set of constraints in order to be able to solve the linear problem. While deteriorating the quality of the solution, these model restrictions require complex implementations. Another issue is the required use of a relevant density function that models the weight of each state in the optimization, what can be hard to define for some problems. Automated generation of a good set of basis function has been recently proposed in (Kveton and Hauskrecht 2006), as well as the use of more general transition functions and approximation techniques (Kveton, Hauskrecht, and Guestrin 2006), which shows that the authors are well-aware of these limitations, but they are still not completely lifted.

In the HAO^* algorithm, the authors extend the AO^* al-

gorithm to hybrid domains. The authors perform an exact backup on the whole continuous state space and represent the value function as a piecewise linear function, organized as kd-trees (in current implementation), and explore the state space in a “classical” tree search, grouping states that have same values. So the exact backup solution requires the use of expensive linear programming techniques in order to update the vectors of the piecewise linear value function in Bellman backups. We can see two main issues with this approach: the backup operation itself is extremely heavy and is dependent on the representation used for the value function. Furthermore, current implementations of HAO^* cannot deal with cycles (existing in our domains) because of the representation of the value function currently used, and moreover because it heavily relies on a heuristic that is domain-dependent (and that can be generalized only for domain without cycles).

We can also cite (Li and Littman 2005), in which the authors use a lazy approximation scheme for the value function but still need to go through piecewise linear constant representation, and (Marecki, Koenig, and Tambe 2007) in which it is approximated as phase-type distributions. While these representations are sound and efficient, they do not entirely solve the problem of the backups.

The curse of exact backups; proposed approach

One should notice that the need for exact backup comes from the fact that all these algorithms perform the Bellman backup on the whole (or reachable) state space at the same time. So they have to be able to have a closed-form integration, once again done with exact backup then approximation of the representation. We propose a radically different approach, based on the combination of smart forward sampling (made easy by the use of the RTDP (Barto, Bradtko, and Singh 1995) algorithm framework), and online machine learning techniques, especially of regression algorithm used to represent the value function (and confidence in this approximation) within constant (or limited) memory use.

Our *claims* are the following: the value function should be represented locally (the global value function should be decomposed into several value functions defined only over parts of the state space), and the limited support of this representation lifts the need for a complex approximation scheme along with closed-form integration. Then, thanks to asynchronous updates of the value function (like in RTDP), it is sound to update only a local representation of the value function for one subset of states S_i (we do not need to evaluate the value function for the whole state space at the same time like other approaches do by using linear programming). Using this local representation of the value function, sampling techniques can be used in order to approximate the Bellman integration, allowing to take into account only the states that are reachable from S_i (we do not need to perform the Bellman integration over the whole state space, neither the whole reachable state space). Furthermore, machine learning techniques allow to represent value functions locally efficiently (typically, only regression is needed).

This approach also has interesting theoretical *properties*: first, as we consider only local representation of the value function and approximation of the Bellman backups using

sampling, it is possible to deal with any transition function: we do not need any specific or restrictive form for the transition function, contrary to other approaches. Furthermore, RTDP has a very interesting anytime behavior, which will be reproduced by our algorithm as shown further in the *Experimental results* section. Finally, as we are performing a forward exploration (the RTDP trials), we can focus not only on the most reachable states, but also on the part of the space where the confidence on the knowledge of the value function is low, hopefully allowing a quicker convergence.

Example domains

We present two examples of stochastic planning domains with both discrete and continuous state variables, and cycles in the transition graph. The first one has a lot of discrete state variables, what makes the search in the planning graph hard because of the curse of dimensionality phenomenon (Bellman 1957). The second one contains many continuous state variables, so that it is challenging for efficient approximation of functions of continuous variables.

Search-and-rescue In the search-and-rescue mission depicted in Figure 1, the autonomous helicopter has to: find *potential* landing zones near the survivor (by using its memory and analyzing images); then explore them to decide if it is able to land in them. After landing, the survivor tries to board in, but may fail with a probability depending on his distance to the zone. In case of success, the helicopter comes back to the control center. Otherwise, it tries a different zone until the fuel is below a given threshold.

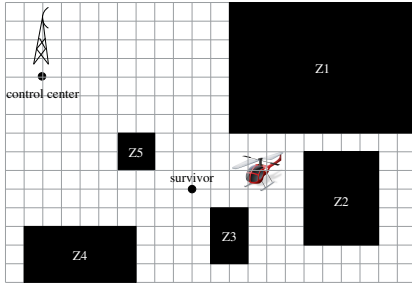


Figure 1: search and rescue domain

Let $Z = \{z_1, \dots, z_n\}$ be the set of potential landing zones, cc be the control center. Formally, discrete variables are: $at \in \{z_1, \dots, z_n\} \cup \{cc\}$: zone which the helicopter flies over; $\forall z \in Z, explored(z) \in \{0, 1\}$: boolean flags indicating for each zone if it has been explored; $\forall z \in Z, landable(z) \in \{0, 1\}$: boolean flags indicating for each zone if it is possible to land in it (after it has been explored); $com \in \{0, 1\}$: boolean flag indicating if communication with the control center is possible; $on_ground \in \{0, 1\}$: boolean flag indicating if the helicopter is on ground or if it is flying. Continuous state variables are: $fuel \in [0, FM]$: remaining fuel (FM : initial fuel level); $mem \in [0, MM]$: available memory (MM : maximum memory).

The planning domain's actions are: $\forall z \in Z \cup \{cc\}, goto(z)$: go to zone z ; $take_picture$ (and analyze

it); $download$ in order to free memory; $explore$ (the zone being flown over); $land$ (in the zone being flown over); $take_off$; $end_mission$ (if the survivor has boarded or if the remaining fuel is below a given threshold).

In order to illustrate the difficulty of Bellman integration one can see in Figure 2 that the effects of the action are defined by pieces both continuous and discrete, the success of the landing itself is probabilistic. If it fails the fuel is decreased with a Gaussian law depending on the altitude; in case of success the human is rescued with some probability depending on the distance to him. Clearly, these effects cannot be integrated in a closed-form way.

```
(and
  (probabilistic (land-success)
    (and
      (on-ground)
      (when (not (at cc))
        (forall (?z - zone)
          (when (at ?z)
            (probabilistic (/ 1 (+ 1 (* 0.03 (distance ?z hm))))
              (human-rescued))))))
    (probabilistic
      (gaussian
        (* (altitude) (land-consumption)) ;; mean
        (* 0.1 (* (altitude) (land-consumption))) ;; variance
        #rv ;; random variate
        (decrease (fuel-level) #rv) ) ) ) )
```

Figure 2: Conditional probabilistic effects of the *land* action

Airport ground traffic's management This planning domain consists in moving ground planes in an airport. Taxiways are modeled as a graph as shown on Figure 3. At each decision epoch, planes target the next planned waypoint, but the distances they cover are stochastic. Consequently, the positions of planes at each decision epoch are continuous stochastic variables defined over the graph's segments. Finally, some pilots may not target the right next waypoints optimized by the planner.

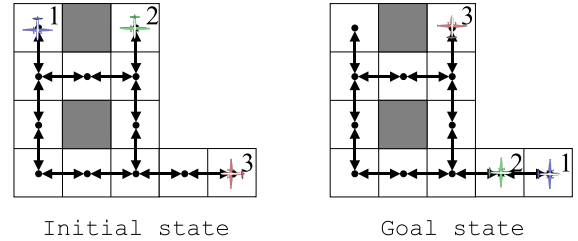


Figure 3: airport ground traffic management domain

We note $P = \{p_1, \dots, p_n\}$ the set of planes, and $W = \{w_1, \dots, w_q\}$ the set of waypoints. The function $C : W \times W \rightarrow \{0, 1\}$ indicates if 2 waypoints are connected in the graph. This planning domain's state variables are: $\forall p \in P, at(p)$: discrete variables that stand for the graph segment which frames each plane. This segment is represented by an ordered pair in order to model the orientation of the plane; $dom(at(p)) = C^{-1}(\{1\})$. $\forall p \in P, abs(p)$: continuous variables which represent the continuous position of each plane p in its segment $at(p)$, $dom(abs(p)) = [0; 1]$.

We assume all planes move at the same time, so that the actions are factored by the individual actions of each plane. The individual actions of a plane p are (we note $(w_1, w_2) = at(p)$ the segment that frames p): $move(p)$: move until reaching the end of the current segment, and

wait there for the next action if reached ; $\forall w \in \{w' \in W \setminus \{w_1\} \mid C(w_2, w') = 1\}$, *move_and_target*(p, w): move until reaching the end of the current segment, then, if it has been reached, move and target waypoint w ; *stop*(p): stop at the current (continuous) position.

Extending RTDP to hybrid domains

Real Time Dynamic Programming (RTDP) is an efficient algorithm scheme proposed initially in (Barto, Bradtke, and Singh 1995). Starting from a given state (initial state for a shortest stochastic path problem or current state if used on-line), it simulates a trajectory of the agent while greedily selecting the best action (so far) in every encountered state. A trajectory (a trial) stops whenever the goal is reached (in case of shortest stochastic path problems where a single goal is given, or when some given horizon is reached). In order to evaluate actions near unexplored states, a heuristic function is used to give an expected value to these states. One interesting feature of this algorithm is that it updates values of encountered states only, which, due to the trial mechanism, will be the most probably encountered.

A pseudo-code for the RTDP algorithm is given as Algorithm 1. The external *while* loop indefinitely repeats what is called *RTDP trials*. Practically, this loops is stopped when the value of the initial state does not decrease anymore (than a given ϵ) or when the mission is finished (since RTDP can be used on-line). It is worth being noted that this is not a sufficient condition of optimality, thinking about the case where the optimal solution has not been explored yet due to the stochastic nature of the trials. In the (discrete) MDP case,

Algorithm 1: RTDP

```

// s is initially the initial state
// all s.explored are false initially
1 while true do
2   while  $\neg GOAL(s)$  do
3      $a \leftarrow greedyAction(s)$ ;
4      $s.Value \leftarrow update(s)$ ;
5      $s \leftarrow pickNextState(s, a)$ ;
6      $s.explored \leftarrow true$ ;

```

updating the value of a state s takes the form of equation 1, where $H(s)$ denotes the value of an admissible heuristic function over S , and a is the action greedily selected.

$$s.Value = \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \quad (1)$$

$$V(s) = \begin{cases} s.Value & \text{if } s.explored = true \\ H(s) & \text{if } s.explored = false \end{cases} \quad (2)$$

Choosing the best action is done within the same calculation as updating the value of the current state (as it is computed with eq. 3). Equivalently, if the values are stored for every states, the policy can be computed as shown on equation 3.

$$\pi^*(s) = \operatorname{argmax}_{a \in A} \left(\sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V(s')] \right) \quad (3)$$

Bellman update with hybrid state variables In order to generalize the RTDP scheme to the hybrid case, we need to define several functions. The first one is about $s.explored$ ¹: we cannot give an *explored* flag to every continuous state, instead, we need to define an exploration function *explored* : $S \rightarrow \{true, false\}$, telling the algorithm if we should use a heuristic value or a computed value for the state s .

Another issue comes with $s.Value$: in the same idea, $s.Value$ cannot be stored in the usual way, as the state space is continuous. We need to define a value function *value* : $S \rightarrow \mathbb{R}$, that for any state s in the continuous space S will store the value of a state. Along with the value representation, we need a consistent way to store the policy, since it cannot be represented with classical tabular representations.

But the most complicated cases come from the update itself, equation 1, where we sum over the (discrete) state space: this may be generalized as a continuous integral sum, or more exactly, with a mixture of sum and integrations in a hybrid space with dependent summation bounds. For now, we just need to define an update function *update* : $((S \rightarrow \mathbb{R}), S) \rightarrow (S \rightarrow \mathbb{R})$, that is able to update the *value* function for one state $s \in S$ (and its neighborhood).

Hybrid planning using machine learning

Recent on-line machine learning methods like LWPR (Vijayakumar, D'souza, and Schaal 2005) and on-line Gaussian Processes (Csato and Opper 2002) can be used to efficiently represent functions of continuous variables whose values permanently change and have to be always learned again (as opposed to off-line learning methods). Such methods *learn* the values of these functions for some given states, and they also *predict* the values of these functions for the neighbour learned states. If the learned values are in a discrete set, the learning algorithm is called a *classifier*. If they are in a continuous space, it is called a *regressor*. Besides, most on-line learning methods are *sparse*, i.e. the memory used to learn the function is bounded by some constant.

Whatever the framework which they are applied to, on-line machine learning methods are state-of-the-art tools to represent functions of continuous variables very efficiently in terms of space and time. Consequently, we use on-line regressors to represent the value function of hybrid MDPs, and classifiers to store the policy and the exploration function. Although these methods come from the machine learning community, our algorithm is not related at all to reinforcement learning, since we perfectly know the planning model.

On-line learning methods are used in continuous spaces, whereas our state space contains both discrete and continuous variables. It would be possible to consider each discrete state variable as a set of fixed values of some artificial continuous variables that might be used by the learning methods. Yet, we argue that this solution would not be very efficient because it adds a lot of dimensions to regressors and classifiers, whose complexity of algorithms drastically increases with the input's dimension. Also, we use on-line learning tools only over the continuous part of the

¹in RTDP, this boolean is implicit: when a node has not been explored, the heuristic value is used

space space, thanks to the following data structure which is similar to the hybrid search graph of HAO* (Meuleau et al. 2009), except that the latter uses kd-trees to represent functions of continuous variables.

Definition 2. A *Hybrid Planning Graph (HPG)* is an oriented graph (\mathcal{S}^d, T^d) where:

- $\mathcal{S}^d \subset \bigotimes_{i=1}^q \mathcal{V}_i^d$ is the projection of the hybrid state space on the discrete state variables; each vertex (v_1^d, \dots, v_q^d) , called a *discrete state*, is a tuple $\langle \pi_c, V_r \rangle$ where:
 - $\pi_c : \bigotimes_{i=1}^p \mathcal{V}_i^c \rightarrow \mathcal{A}$ is a multi-class classifier such that $\pi_c(v_1^c, \dots, v_p^c) = \pi(v_1^c, \dots, v_p^c, v_1^d, \dots, v_q^d)$;
 - $V_r : \bigotimes_{i=1}^p \mathcal{V}_i^c \rightarrow \mathbb{R}$ is a regressor such that $V_r(v_1^c, \dots, v_p^c) = V(v_1^c, \dots, v_p^c, v_1^d, \dots, v_q^d)$.
- $T^d = \{(s_1^d, s_2^d) \in (\mathcal{S}^d)^2 \mid \exists (s_1^c, s_2^c, a) \in (\bigotimes_{i=1}^p \mathcal{V}_i^c)^2 \times \mathcal{A}, T((s_1^c, s_1^d), a, (s_2^c, s_2^d)) > 0\}$ is the set of possible transitions between the discrete states of the graph.

In this structure, one node corresponds to one instantiation of all discrete variables: the graph does not need to split nodes, it adds new one on the fly when they are encountered during exploration. In this definition, the *explored* function defined earlier is not present. But we simulate this function with the V_r functions: as the regressors toolboxes we use give not only the result but also the confidence (or the confidence bound) in the result (i.e. the learning noise), we just say that if the confidence bound is too high (or the confidence too low), then the point of the hybrid state space has not been explored enough, and we consider *explored = false* in this case: we do not need to have a separate binary classifier.

At the initialization of HRTDP, the HPG only contains 1 vertex, that corresponds to the discrete variables' instantiation of the initial state. New nodes are added to the graph during the search at each Bellman backup, as explained below. In the course of HRTDP trials, we jump from HPG nodes to HPG nodes while updating the value of the current hybrid state by using the value function regressor of each visited HPG node.

Confidence in learned information One key aspect in using modern regression and classification techniques is the notion of confidence: modern toolboxes allow not only to use the approximation resulting from the regression (or classification), but also give the confidence in this result, that can be seen as the probability that the prediction is correct given the information used so far. Some toolboxes, like the one we used, prefer the confidence bound, whose intuition is the opposite: if the bound is high, the knowledge quality is low. Indeed, it is critical to take this confidence into account while using learning techniques; if not, then it would be considered implicitly that the learning phase is perfect, particularly that the regressor gives the correct value, even with a very small training set, which cannot be true.

Furthermore, as we will see in the next section, the confidence allows to smartly guide the exploration itself within the RTDP framework. Remember that RTDP greedily chooses the best action in the deterministic case. In our case (using ML techniques), the best action is a relative criterion: if the confidence bound is low, then we know what

the best action is, but if it is high, then we should not trust that much that this action is the best one (especially if there has not been much training samples around this action). We propose an exploration versus exploitation scheme based on confidence of which action is best to guide the algorithm.

Hybrid Bellman backup Updating the value of a hybrid state requires to compute an integration over continuous and discrete state variables. As shown in Algorithm 2, we compute an approximation of this hybrid integral with a Monte-Carlo integration method where the value of the integrated function is predicted from the value function regressor of the next states' HPG nodes (line 9). New nodes are added to the HPG when random successor discrete states have not been explored (line 6). If a random successor continuous state has not yet been explored (that is the confidence bound is too high), we compute its heuristic value (see below) and we request to its value function regressor to learn it (lines 7 and 8). Finally, we request to the value function regressor (resp. the policy classifier) to learn the updated value (resp. policy) of the hybrid state (lines 14 and 15).

The *random_next* function uses the HMDP transition function T to generate random successor states. No assumptions are made regarding the form of the probability distribution in T . As a result, our algorithm works with any kind of probability distributions that can be sampled. To our knowledge, no other HMDP algorithm is able to deal with any kind of transition functions.

Algorithm 2: Hybrid state update

```

//  $s = (s^c, s^d)$  is the state to update
//  $s^d$  refers to the HPG node containing  $s$ 
//  $N$  is the number of samples to compute
// an approximation of the integral
//  $CT$  is the confidence threshold, over
// which the node is seen as unexplored
1  $best\_qvalue \leftarrow -\infty$ ;
2 for  $a \in \mathcal{A}$  do
3    $qvalue \leftarrow 0$ ;
4   for  $i \leftarrow 1$  to  $N$  do
5      $((s^c, s^d), r) \leftarrow s.random\_next(T((s^c, s^d), a, \cdot));$ 
6     if  $s^d \notin HPG.nodes()$  then  $HPG.add\_node(s^d)$ ;
7     if  $confidence(s^d.V_c.predict(s^c)) > CT$  then
8        $s^d.V_r.learn(s^c, H(s^c, s^d))$ ;
9      $qvalue \leftarrow qvalue + (\gamma \times s^d.V_r.predict(s^c)) + r$ ;
10     $qvalue \leftarrow qvalue/N$ ;
11    if  $best\_qvalue < qvalue$  then
12       $best\_qvalue \leftarrow qvalue$ ;
13       $best\_action \leftarrow a$ ;
14  $s^d.V_r.learn(s^c, best\_qvalue)$ ;
15  $s^d.\pi_c.learn(s^c, best\_action)$ ;

```

HRTDP With the update mechanism described above, we can now propose the HRTDP algorithm shown as Algorithm 3. First, we update the value for the state s and learn policy and the regressor as shown in previous section, and (in the very same loops in the real implementation) we track both the best action for this state s and the action that leads to

least known regions of the state space. Then, depending on how badly the region is known, we choose to apply either the greedy selection as in RTDP, or an exploration action that will lead to explore the less known region, in order to reduce the learning bias and to improve the knowledge as fast as possible. The $\alpha > 0$ parameter allows us to tune the exploration vs. exploitation scheme, and may be changed online, for instance to implement a simulated-annealing-like approach. In Algorithm 3, as in the experiments presented further, we randomly choose the best action with a probability $e^{-\alpha \cdot worstConf}$, what means that the less the region is known, the more exploration is performed.

Algorithm 3: HRTDP

```

1 while true do
2   // s is initially the initial state
3   while ¬GOAL(s) do
4     hybrid_state_update(s);
5     lessKnownA ← action leading to predictions with
6     highest confidence bound ;
7     worstConf ← corresponding confidence bound;
8     bestA ← action leading to states with highest value;
9     a ← { bestA with p = e-α.worstConf
10         lessknownA with p = 1 - e-α.worstConf
11     }
12     s ← pickNextState(s, a);

```

Domain-independent admissible heuristic Each time the exploration classifier of a HPG node predicts that a hybrid state s has not been explored, an admissible heuristic value $H(s)$ must be computed such that: $H(s) \geq V^{\pi^*}(s)$. Naive heuristics may require to generate all the nodes of the HPG, what is exponential in the number of discrete state variables. Also, we implemented a relaxed heuristic inspired by the deterministic FF planner (Hoffmann and Nebel 2001) to deal with the HMDP model.²

This heuristic assumes that all discrete variables are binary, what is actually not restrictive because any n -ary variable can be translated into $\lceil \log_2(n) \rceil$ binary variables. The heuristic constructs two increasing lists of all the *true* (resp. *false*) values taken by each discrete state variable during the heuristic search, without ever removing values from the list. While linear in space, these lists loose the dependence of the variables of the states explored during the heuristic search. The transition function is relaxed so that it can directly operate on these lists, without considering the continuous state variables at all. The search stops when the goal state is included in the lists, and the discounted opposite value of the distance to the goal state is an admissible heuristic for the shortest stochastic path problem. The complexity of the relaxed heuristic search is polynomial in the length of the lists, i.e. polynomial in the number of discrete state variables.

While searching on the discrete states only, the heuristic is still admissible because, intuitively, more steps would be required to reach the goal state by considering the continuous state variables during the heuristic search (there would

²This heuristic is inspired from the FSP* planner, more precisely is an implementation of the RDH heuristic found in <http://ippc-2008.loria.fr/wiki/images/c/c2/Team1-FSP.pdf>

be less applicable actions at each expansion step). Also, the relaxed distance is lower than the distance that would be computed with actual hybrid states. This heuristic has been chosen because it performed quite well in the past planning competitions, and because it is domain independent, and can be used for domains with cycles.

Experimental results

We tested our algorithm HRTDP on shortest stochastic path instances of the domains presented in the second section.³ We used an extension of the PPDDL language defined in (Teichteil 2008) to model HMDP problems with any kind of probability distribution functions. We used LWPR (Vijayakumar, D’souza, and Schaal 2005) for the implementation of HPG nodes’ regressor and classifiers. For each test, we ran HRTDP 10 times and averaged different criteria over the 10 HRTDP runs. Each criterion is presented as an average function (over HRTDP runs) of the optimization time (in seconds) observed after each HRTDP trial. In all following figures, in green is shown the average value and in red the maximum difference to the mean.

Search-and-rescue domain Based on the description of this domain, there are $2^{3+2n}(n+1)$ discrete states (maximum number of HPG nodes) and 2 continuous variables, where n is the number of zones.

Figure 4 shows that HRTDP converges extremely quickly in terms of value of the initial state (value is the expectation of rewards collected while executing the policy), meaning that our algorithm is able to give quickly good solutions for the initial state (and most probable trajectories). A “valley” can be noticed for the 20 zones problem, it seems that the regressor used underestimates the expected costs at some point. As we can see in figure 5 the number of nodes in the hybrid planning graph continues to extend after the value has stabilized, showing that our algorithm continues to explore some less reachable states (anytime behavior), while not really increasing the expected reward, exactly as expected with RTDP-like algorithms (if states are less reachable, they have lower influence on the expected reward). One could notice that the number of (explored) nodes in the HPG is very small compared with the theoretical maximum number of discrete states, this is due to the nature of the domain, because many actions have very strong preconditions and can be applied only in few states, leading to an extremely large number of states that are nearly unreachable.

On figure 6(a) is shown the percentage of explicit choice of an exploration action. This percentage does not decrease during the resolution; this is related to the worst confidence bounds during the backups shown on figure 6(b), which seems to reach an upper bound. Indeed LWPR reports the confidence bound which behaves as the opposite of the intuition of the confidence: the higher the value is, the lower we can trust the prediction. So in this case, due to the fact that we explore more and more states, the worst confidence

³We could not compare to HALP nor HAO* on the domains they can solve because we could not obtain their codes, which is hard and complex to implement, nor compiled versions. Moreover, domains used in publications are not available.

bound gets worse and worse. We think that it would improve after the whole state space has been explored enough, which we definitely do not want to do.

In order to show the influence of the explicit exploration, we ran another set of experiments where the α factor is very low, leading to few explicit exploration actions, as shown on figure 7(a). In this case we can see that the worst confidence bound during bellman backups reaches almost the same value, but takes more time to do so (on figure 7(b)). This comforts us into our first intuition of reason for the high (and non-decreasing) percentage of exploration.

Finally, both the explicit choice of an exploration action and the regression induced noise have an impact on the length of the HRTDP trials. Recall that the trials stops when reaching the goal, giving us a hint on the quality of the solution so far. On figures 8(a) we can see the (noisy) length of the trials, while on figure 8(b) we can see that when the value of the initial state gets minimal for the first time, HRTDP performs very short trials (that is the optimal solution), and after that the length augments gradually because HRTDP explores new regions, while not being able to improve the value of the initial state (as the length to reach the goal are greater).

We also tested the influence of the number of samples for the approximation of the Bellman backup, but it is straightforward: with less particles the calculation is faster, shows a lot of noise, and the more particles the less noise but the more time taken.

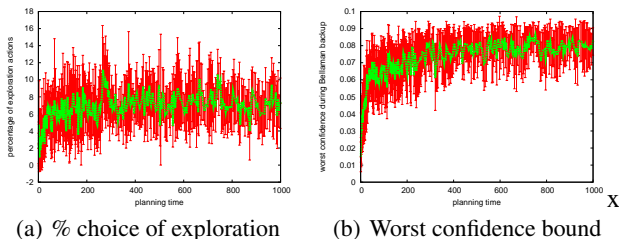


Figure 6: Expl. and conf. for 20 zones in the S&R domain

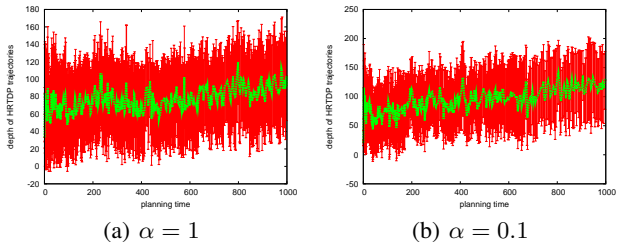


Figure 8: Length of trials (in the 10 zones S&R domain)

Airport management domain Figure 9 shows the results obtained on one instance of the airport ground traffic management domain which contains $(2N)^p$ discrete states and p continuous variables, where N is the number of connected waypoints ($N = 12$ with the tested airport) and p is the number of planes (5 in the results presented here, other are not included for space reasons). While this domain is even more difficult than the previous one, we can see that HRTDP is able to cope with it. Despite the higher number of regressors in this domain, HRTDP is still able to quickly converge to a stable policy. There are fewer points in the figures, because more computations are needed in each Bellman backup.

Comparison with a purely heuristic strategy In order to prove that HRTDP does improve the heuristic value to find good strategies, we also compared HRTDP with a purely heuristic strategy that consists in always choosing the best heuristic action in a state without learning (the heuristic value of an action in a given state is the sum of the immediate reward and the average heuristic value of its successor states). While returning informative values for HRTDP’s Bellman backups, the heuristic presented in the previous section (computed on a relaxation of the discrete state space), used in a purely heuristic strategy, was totally unable to reach any goal state in the hybrid domain over 100 runs of depth 200 each, for all the problems previously presented. Besides, the statistical value of the initial state when using this strategy was equal to $-10 = -1/(1 - \gamma)$ in all cases (with $\gamma = 0.9$), which corresponds to near-infinite trajectories that never reach a goal state. On the contrary, as shown in the previous figures, HRTDP using this heuristic was always able after very few seconds to reach a goal state in far fewer than 200 steps and the value of the initial state was never below -5 or -2 (depending on the problem): compared with the purely heuristic strategy with the same heuristic, these results do prove that HRTDP quite well optimizes the value function despite learning noise.

Conclusion and perspectives

We have shown that it is possible to solve HMDPs in an asynchronous way using forward heuristic search and hybrid representations of the core components needed to perform Bellman backups. We have proposed a formulation of such algorithm, together with the use of state-of-the-art machine learning algorithms for regression, also using on-line incremental learning for Bellman backups. We have shown the effectiveness of these techniques on two different complex domains. Our technique allows to deal with any transition function, what was not possible with any previous approach.

Future work will be to use other on-line learning methods such as Gaussian processes (Csató and Opper 2002; Lawrence, Seeger, and Herbrich 2003), in order to see the influence of the regression technique, which is crucial to ensure that our algorithm gives good policies, hopefully allowing to prove (ϵ -)optimality. We also plan to expand this approach to deal with partially observable MDPs, by using the same techniques into the (fully continuous) belief space.

References

- Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *AIJ* 72.
- Bellman, R. E. 1957. *Dynamic Programming*. Princeton University Press.
- Csató, L., and Opper, M. 2002. Sparse on-line gaussian processes. *Neural Computation* 14(3):641–668.
- Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.
- Kveton, B., and Hauskrecht, M. 2006. Learning basis functions in hybrid domains. In *AAAI*.

