Problem Solving Using Classical Planners
AAAI Technical Report WS-12-12

Solving Goal Hybrid Markov Decision Processes Using Numeric Classical Planners

Florent Teichteil-Königsbuch
florent.teichteil@onera.fr
Onera — The French Aerospace Lab
F-31055, Toulouse, France

Abstract

We present the domain-independent HRFF algorithm, which solves goal-oriented HMDPs by incrementally aggregating plans generated by the Metric-FF planner into a policy defined over discrete and continuous state variables. HRFF takes into account non-monotonic state variables, and complex combinations of many discrete and continuous probability distributions. We introduce new data structures and algorithmic paradigms to deal with continuous state spaces: hybrid hierarchical hash tables, domain determinization based on dynamic domain sampling or on static computation of probability distributions' modes, optimization settings under Metric-FF based on plan probability and length. We compare with HAO* on the Rover domain and show that HRFF outperforms HAO* by many order of magnitudes in terms of computation time and memory usage. We also experiment challenging and combinatorial HMDP versions of benchmarks from numeric classical planning, with continuous dead-ends and non-monotonic continuous state variables.

Introduction

Hybrid Markov Decision Processes (HMDPs) with discrete and continuous state variables (Kveton, Hauskrecht, and Guestrin 2006; Marecki, Koenig, and Tambe 2007; Meuleau et al. 2009) offer a rich model for planning in probabilistic domains. Recent advances in solving HMDPs allow practitioners to solve complex real problems, like irrigation networks (Kveton, Hauskrecht, and Guestrin 2006) or Mars rover navigation (Meuleau et al. 2009). Yet, state-of-the-art algorithms usually consume a lot of time and memory, thus hardly scaling to larger problems. One of the first papers about using Hybrid MDPs for solving realistic applications (Bresina et al. 2002), mentions that such complex problems could be certainly only tackled by "new and dramatically different approaches". They propose an appealing heuristic approach – radically different from existing methods –, which would consist in building an initial plan that would be progressively improved by "augmenting it with contingent branches". However, they only give principles, but not practical algorithmic means, of such a method, mentioning that it would be non-trivial at all, even in the case of their particular problem.

This idea has been actually recently implemented with success in the field of goal-oriented MDPs with only discrete state variables, where many approaches propose to construct a policy by calling many times a deterministic planner on a determinized domain (Yoon et al. 2008; Kolobov, Mausam, and Weld 2010; Teichteil-Königsbuch, Kuter, and Infantes 2010). Such methods have often proven to scale better than traditional optimal MDP algorithms, without compromising optimality too much. However, extending such algorithms like RFF to domains with continuous variables, as envisioned by (Bresina et al. 2002), is not straightforward because these algorithms rely on assumptions that are only valid for discrete state spaces: e.g. countable states, notions of explored or expanded graph nodes, finite number of actions' effects. Encoding and updating action policies in efficient data structures over mixed discrete and continuous state variables is still an opening issue.

In this paper, we present a heuristic algorithm named HRFF, for *Hybrid* RFF, which approximately solve goal-oriented HMDPs using some numeric classical planner. It actually implements and extends intuitive principles described by (Bresina et al. 2002) in a domain-independent manner: it incrementally adds contingent branches to an initial plan in order to construct a compact and adaptive contingent plan, i.e. policy, defined over discrete and continuous state variables. HRFF is an extension of RFF (Teichteil-Königsbuch, Kuter, and Infantes 2010) to hybrid state spaces and continuous probabilistic changes of actions' effects. As RFF, HRFF incrementally aggregates plans into a policy, but using the Metric-FF (Hoffmann 2003) hybrid deterministic planner from many states that are possibly reachable by executing the current policy from the initial state. Our algorithm introduces new data structures and algorithmic paradigms to deal with continuous state spaces: hybrid hierarchical hash tables, different domain determinization techniques based on dynamic domain sampling or on static computation of probability distributions' modes, optimization settings under Metric-FF based on plan probability and length. HRFF does not only "simply" merge plans with the current policy: it also takes into account probabilities of actions' effects, in order to select the most helpful actions from the plans to merge into the current policy, without decreasing

its quality. We analyze HRFF and its underlying deterministic planner on probabilistically-interesting HMDP benchmarks, and show that HRFF outperforms HAO* by many order of magnitudes on the rover domain.

Goal Hybrid Markov Decision Processes

A goal-oriented Hybrid Markov Decision Process (goal-HMDP) is a tuple $\langle S, A, T, I, G \rangle$ such that: $S = \bigotimes_{i=1}^{n} V_i^c \times \bigotimes_{i=1}^{m} V_i^d$ is a cartesian product of n continuous and m discrete state variables; A is the set of enumerated and discrete actions, each action $a \in A$ is applicable over a set of states S_a; $T : S \times A \times S \rightarrow [0; 1]$ is a transition function, such that for all $(s, a, s') \in S \times A \times S$, $T(s, a, s') = dP(s' \mid a, s)$ is the hybrid probability distribution of arriving in state s' when starting in state s and applying action a; I is the initial state of the decision process; G is the set of goal states.

We assume that the hybrid probability distribution dP of the transition function can be algorithmically sampled and that we can compute its *mode*, i.e. the values of its random variables that maximize it. For an action a whose transition's probability distribution is discrete, the mode can be seen as the most probable effect of a. In our implementation, we use the Gnu Scientific Library (Free Software Foundation 2011), which offers a wide set of distributions used in many engineering or physics applications. Contrary to many models of HMDPs or continuous MDPs proposed in the literature (Kveton, Hauskrecht, and Guestrin 2006; Marecki, Koenig, and Tambe 2007; Meuleau et al. 2009), we do not assume continuous state variables to be in a closed interval of $\mathbb{R}$. We also handle non-monotonic continuous state variables, which can increase or decrease over time.

Finally, we define a convenient function $succ : S \times A \rightarrow 2^S$ such that for all state s and action a, $succ(s, a)$ is the set of states that are directly reachable with a positive probability density by applying a in s. Because of continuous state variables, $succ(s, a)$ may be an infinite subset of S.

Solving goal-oriented HMDPs. We aim at computing a *policy function* $\pi : S \rightarrow A$ that, ideally, maximizes the probability to reach the goal, while minimizing the average number of steps required to reach the goal from the starting state. In particular, we are interested in problems where there is a positive probability to reach some states, named *dead-ends*, from which there is no path leading to the goal. As in (Meuleau et al. 2009), we do not need to compute a policy defined over all states, but a partial and closed one: $\pi : \mathcal{X} \subseteq S \rightarrow A$ such that $I \in \mathcal{X}$ and for all $s \in \mathcal{X}$, $succ(s, \pi(s)) \subseteq \mathcal{X}$. In other terms, executing π from the initial state I will always lead to a state where the policy is defined. However, some algorithms like HRFF presented in the next, are based on Monte-Carlo sampling, which means in theory that states reachable by applying the current policy from the initial state cannot be all explored in finite time. Therefore, we define p-closed policies $\pi : \mathcal{X} \subseteq S \rightarrow A$ such that for all $s \in \mathcal{X}$, $Pr(succ(s, \pi(s)) \subseteq \mathcal{X}) \geqslant p$.

PPDDL-based modeling of HMDPs. We consider domain-independent planning, where the goal-oriented HMDP is modeled in an extension of PPDDL (Younes and Littman 2004). Our extension to the grammar handles various discrete and continuous probability distributions, and so probabilistic continuous state changes (Teichteil-Königsbuch 2008). It introduces random continuous variable terms, whose stochastic values impact their underlying effects. For instance, in the following example, the continuous variable fuel is assigned a stochastic value that follows a lognormal probability distribution:

```
(probabilistic (lognormal
  (capacity ?a) (* 0.001 (capacity ?a)) #rv)
    (assign (fuel ?a) #rv))
```

As described in the next section, PPDDL-based modeling of HMDPs allows us to automatically derive a deterministic planning problem in the PDDL-2.1 language (Fox and Long 2003), which can be solved by a numeric deterministic planner like Metric-FF (Hoffmann 2003).

Hybrid probabilistic domain determinization

Like many successful algorithms for solving discrete-state MDPs (Kolobov, Mausam, and Weld 2010; Yoon et al. 2008; Teichteil-Königsbuch, Kuter, and Infantes 2010), our algorithm relies on automatic domain-independent determinization of the probabilistic domain. In the deterministic case, two determinization strategies have been particularly studied: "most probable outcome determinization" and "all outcome determinization". The first one consists in translating each probabilistic action into a deterministic action whose effect is obtained by recursively keeping the most probable effect of each probabilistic rule that appears in the effect of the probabilistic action. The second one translates each probabilistic action into as many deterministic actions as the number of possible effects of the probabilistic action. The first strategy leads to less deterministic actions, reducing the makespan of the deterministic planner, but it may end up with empty plans if the goal of the probabilistic problem is not reachable from the initial state by following only the most probable trajectories of all actions in the model. On the contrary, the second strategy is complete, but it drastically increases the makespan of the deterministic planner.

Mode-based determinization. The "most probable outcome determinization" can be easily generalized to HMDPs, even if the probability of a continuous effect has no sense a priori (continuous random variables have no probability mass). The solution resides in the mode of probability distributions, which is the point in the space of random variables of the distribution, which maximizes the density of the distribution. Concerning the transition function of actions in HMDPs, the mode is the outcome hybrid state s^* such that $dP(s^*)$ is maximum. For discrete-state MDPs, the mode of the transition function $T(s, a, s')$ reduces to the most probable outcome of action a, i.e. the state with the highest chance to be reached in one step when applying action a in state s. This interpretation is not directly transposable to continuous state spaces, since a single state has theoretically no chance to be reached from a previous one. However, if we sample many states s' from s by applying a, most of them will be

distributed around the mode of the transition function, provided it is continuous and smooth enough in the vicinity of the mode. For this reason, we can often interpret the mode of the transition function as the state that attracts at most outcomes of action a applied in state s.

Sampling-based determinization. As in discrete-state MDPs, mode-based determinization does not guarantee to find a policy that reaches the goal with a positive probability, if none of the trajectories generated by following most probable outcomes of actions from the initial state lead to the goal. Indeed, the deterministic planner run on the determinized domain would always return empty plans from all possible initial states. In the discrete-state case, it has been proposed to translate each probabilistic effect of each action into a deterministic action, but this strategy is impossible in continuous domains, because the number of outcomes of actions is potentially infinite. Instead, we propose to dynamically sample the effects of probabilistic actions to create deterministic effects, at each iteration of the HMDP planner. Thus, the entire probabilistic domain is sampled and translated into a deterministic domain before each call to the deterministic planner. Implementation assumptions of the mode-based determinization are valid, and the multinomial distribution can be handled by this strategy.

Compact and adaptive representation of functions defined over continuous variables

A key issue when constructing policies over continuous subspaces is to represent functions of continuous variables, which are (i) compact, (ii) as precise as possible (if not exact) at some points of the continuous subspace, and (iii) which can be updated with a cheap computation cost without decreasing the precision at the points previously updated. Our solution is actually a rewriting from scratch of hierarchical spatial hash tables, recently used with success in spatial indexing and graphics computation (Pouchol et al. 2009), but adapted to probabilistic planning operations, and especially to `HRFF`. Another motivation is to bring the efficiency of standard hash tables, used in many successful discrete-state MDP planners for encoding the search graph over the discrete state variables, to continuous state variables. The mathematical tool behind our planning-specific implementation is a hierarchical equivalence relation defined over the continuous state variables.

Spatial hierarchical equivalence relation. In order to avoid a blowup due to memorizing too many points in the continuous subspace, we aim at representing all points that look similar (in terms of value, policy, etc.) by only one of them. Like many approaches of the literature (e.g. (Lee and Lau 2004)), we specifically search for an adaptive state space partitioning mechanism, whose level of detail is higher in "important" areas of the subspace. To this end, we define the $\sim^{\delta}$ equivalence relation over $\mathbb{R}^n$, which represents the continuous state variable subspace, such that, for two points $(v_1^c, \cdots, v_n^c)$ and $(w_1^c, \cdots, w_n^c)$ in the continuous subspace $V_1^c \times \cdots \times V_n^c$:[1]

$$(v_1^c, \cdots, v_n^c) \sim^{\delta} (w_1^c, \cdots, w_n^c) \Leftrightarrow \left\lfloor \frac{v_i^c}{\delta} \right\rfloor = \left\lfloor \frac{w_i^c}{\delta} \right\rfloor, 1 \leqslant i \leqslant n$$

Intuitively, if we imagine a virtual grid discretization of the continuous subspace whose step is δ, two points v and w in the continuous subspace are equivalent if they belong to the same (hypercube) cell centered at:

$$r_\delta(w) = r_\delta(v) = \left(\left(\left\lfloor \frac{v_1^c}{\delta} \right\rfloor + \frac{1}{2} \right) \cdot \delta, \cdots, \left(\left\lfloor \frac{v_n^c}{\delta} \right\rfloor + \frac{1}{2} \right) \cdot \delta \right)$$

Since this point is uniquely defined for all points equivalent to it, it represents their equivalence class. We name it the δ-*reference* point of the cell that contains it. We have now a way to locally aggregate states by substituting them for their δ-reference point, at a fixed level of detail defined by δ. Yet, if we need to refine the aggregation inside a given cell, we use a more detailed aggregation defined by the $\sim^{\frac{\delta}{2}}$ equivalence relation. Successive refinements of this equivalence relation leads to an adaptive hierarchical partitioning of the continuous subspace. Two important properties can be highlighted for the convergence of HMDP algorithms. *P1: given two points v and w in the continuous subspace, there exists an integer k such that $v \not\sim^{\frac{\delta}{2^k}} w$* ; it means that we can always achieve the most level of precision desired if we want. *P2: given two different integers k and q, all $\frac{\delta}{2^k}$-reference points are different from all $\frac{\delta}{2^q}$-reference points*, meaning that associating points with their reference points in different partitioning levels does not lead to redundant information (refinement and point referencing increases information).

Spatial hierarchical hash tables. We need an efficient algorithmic implementation of the previously defined equivalence relations, so that we can: (i) implicitly represent grid cells ; (ii) locally create these cells on-the-fly ; (iii) quickly access and refine them. Our solution is a spatial hierarchical hash table $\mathcal{H}_\delta^c(T)$ whose first level of (implicit) discretization is $\delta > 0$ and T is the type of elements stored in the hash table. Elements in $\mathcal{H}_\delta^c(T)$ are tuples $(point, data, htPtr)$, where $point$ is a δ-reference point (i.e. represents a cell at the δ-step discretization), $data$ is the information of type T stored in the hash table, and $htPtr$ is a pointer to a $\mathcal{H}_{\delta/2}^c(T)$ refined hash table whose all elements are $\sim^{\delta}$-equivalent to $point$ (i.e. the cells they represent are all included in the cell represented by the parent $point$). For each terminal element, $htPtr$ is `NULL`, meaning that it is not refined. Coordinates of $point$ are used to compute the hash value of each element, because reference points are all unique and different. We call δ-*cells* the elements of $\mathcal{H}_\delta^c(T)$, since they represent cells (with attached data) of size δ in the continuous subspace.

Three operations on spatial hierarchical hash tables are sufficient for our needs: (1) `find` or (2) `insert` data

[1] For a real number x, $\lfloor x \rfloor$ is its integer part.

points, and (3) `refine` δ-cells. The `insert` operation inserts a data point in the hierarchical hash table and returns the inserted cell, or returns an already existing cell if there is a matching in the highest-level hash table. The `find` operation keeps track of the parent cell c of the current visited hash table (i.e. that matched in the parent hash table) and returns it if there is no matching with this hash table. Otherwise, it returns the cell matched in the highest-level hash table. The `refine` operation takes a δ-cell c as input, and a point with its associated data, and inserts it in a refined $\frac{\delta}{2}$-cell included in c. For this purpose, it creates a new hash table, attaches it to cell c, and inserts the input data point in it using cell size $\frac{\delta}{2}$ for hash values and collision tests.

Hybrid hierarchical hash table. A single hierarchical hash table can be used to store data defined over the entire discrete and continuous state space, by pushing a standard hash table, whose keys are sets of discrete state variables, on top of our spatial hierarchical hash table. We note $\mathcal{H}_\delta(T)$ such a hybrid hierarchical hash table, whose δ is the top-level size of cells, i.e. the size of elements included in the top spatial hash table (at level 2). The `refine` operation is only available from level 2 ; the `find` and `insert` operations are valid at all levels, but the equality test and hash value are computed by using the values of discrete state variables at the first level. The δ-reference point of a hybrid state $s = (s^c, s^d)$ is defined as: $r_\delta(s) = (r_\delta(s^c), s^d)$. The first level of our hybrid hierarchical data structure can be seen as a hash table implementation of the Hybrid Planning Graph (HPG) used in `HAO*` (Meuleau et al. 2009). Other levels are obviously different from KD-trees used in nodes of the HPG.

The `HRFF` algorithm

`HRFF` is an extension of `RFF` (Teichteil-Königsbuch, Kuter, and Infantes 2010) to goal-oriented hybrid MDPs, which relies on three new features specific to hybrid domains: (1) hybrid hierarchical hash tables, (2) state equivalence relation $\sim^\delta$ and δ-reference states, (3) plan aggregation based on actions performance statistics computed over hybrid states. Like `RFF`, `HRFF` is a heuristic algorithm, which uses a deterministic planner as a guide to generate helpful state trajectories, i.e. trajectories reaching the goal, that are incrementally merged with the current policy. It is not optimal regarding standard goal-oriented HMDPs criteria like minimal average accumulated cost to the goal, but it aims at quickly obtaining sufficiently good policies in practice.

`HRFF` incrementally aggregates plans computed by `Metric-FF` on a determinization of the probabilistic domain into a policy, until the latter is $(1-\epsilon)$-closed from the initial state I, where $\epsilon > 0$ is the computation precision. It alternates two phases: the first one computes the reference states that are reachable from the initial state by following the current policy until reaching a cell where it is not yet defined (such reference states are similar to reachable unexpanded graph nodes in `RFF`) ; the second one expands the policy on these reference states by calling `Metric-FF` from them on a determinized problem. Convergence on the hybrid state space is guaranteed thanks to properties P1 and P2 of hybrid hierarchical hash tables highlighted in the previous section. Before going into details, we first explain how we transfer deterministic actions from plans to the current policy and update it, since this operation, which is relatively trivial in discrete-state settings, is in fact quite challenging in continuous subspaces.

Policy update using sampled plans. As discussed before, we propose to use sampling-based domain-independent determinization in hybrid domains to replace the "all outcome determinization" employed in discrete-state MDPs, which is not possible in hybrid domains because of the potentially infinite number of actions' outcomes. Yet, on-the-fly domain sampling is theoretically challenging, because two successive calls to the deterministic planner on the same state but with different sampled effects (of all actions in the domain) will likely provide very different plans: some samplings will result in empty plans, some others with plans of different lengths to the goal or with different probabilities to reach the goal. Thus, unlike the discrete-state version `RFF`, it is no longer possible to simply replace the action of an already updated state. Moreover, in theory, the probability of visiting the same state multiple times during the search (from different domain samplings) is zero in hybrid domains. Our solution is to compute some statistical performance metrics about actions included in the plans computed by `Metric-FF`. It is worth noting that the statistics presented below also boost the mode-based determinization approach, by selecting actions that are more likely to lead to the goal in the probabilistic domain.

Let s be some hybrid state and $\varpi = (a^d_{i_1}, \cdots, a^d_{i_k})$ be a plan of length k computed by `Metric-FF` from s on a sampled determinization of the probabilistic domain. When we compute a sampled effect e_φ of an effect e of a given probabilistic action a, we also compute the density $dP_e(e_\varphi)$ of the probability distribution of e at the sampled effect e_φ. The sampled effect gives rise to a deterministic action a^d whose density is: $dP(a) = dP_e(e_\varphi)$. It allows us to compute the density of plan ϖ: $dP(\varpi) = \prod_{1 \leqslant j \leqslant k} dP\left(a^d_{i_j}\right)$, which roughly represents the probability of reaching the goal from state s by executing plan ϖ with the current sampling of the domain. We can also define the density of any subplan $\varpi(a^d_{i_j})$ of ϖ starting at the j^{th} reachable state, which gives an idea of the probability to reach the goal from this state using action $a^d_{i_j}$ in the current sampled domain. The length $k - j + 1$ of this subplan is another good performance criterion, which indicates the length of a solution trajectory to the goal, starting in the j^{th} reachable state from s with the current sampled domain. Finally, performance statistics of each action a are compiled in a hybrid hierarchical hash table $\mathcal{H}^a_\delta$, such that for each hybrid state s, $\mathcal{H}^a_\delta(s)$ is a pair (cd, as) where: cd is the sum of the densities of all subplans of prefix a starting in the highest-level cell containing s, and as is the average length of these subplans.

We use the previously defined statistical metrics of actions to rank plans generated from a given hybrid state, in such a way to stabilize the current policy. Each time an action a is

found in a given subplan of Metric-FF, we first compute its corresponding state s in the plan (by executing the plan from its head up to this action in the determinized domain), then we update its statistics hierarchical hash table at state s and compute its metrics performance in this state using the updated statistics (cd, as), defined as: $m^a(s) = -log(cd) \times as$. We update the current policy in state s if s has not been yet visited, or if $m^a(s) < V^\pi(s)$, where $V^\pi(s)$ is the current best value of plans generated from state s. We encode V^π in a hybrid hierarchical hash table, like action performance statistics and the current policy.

Moreover, using Metric-FF allows us to optimize some metric criterion during plan generation, which was not possible with deterministic planners used in determinization-based approaches to solving discrete-state MDPs. Thus, in order to consolidate the convergence of the value function in the probabilistic domain, we can ask Metric-FF to find the plan that minimizes the value function in the deterministic domain. To this end, we add two additional fluents to the deterministic domain: one representing the sum of the opposite logarithms $(-log(\cdot))$ of probability densities of the actions in the plan (seen as a cost), the other representing the length of the plan. Unfortunately, Metric-FF is not able to minimize fluent products, so we instead minimize their sum. However, this strategy can significantly improve HRFF performances in some domains. In others, it takes far too long for Metric-FF to optimize plan metrics.

Putting it all together. Algorithm 1 presents a detailed pseudo-code of HRFF. The main procedure (Lines 1 to 15) is a loop, which alternates a phase of computation of reference states where no policy is defined and that are reachable from the initial state by following the current policy (procedure compute_reachability, see Lines 16 to 31), and a phase of policy expansion by merging plans computed by Metric-FF from these reachable reference states with the current policy (procedure generate_trajectory, see Lines 32 to 43). Iterations stop when the policy is $(1-\epsilon)$-closed from the initial state (see Line 15). There are numerous differences with the original RFF due to hybrid settings. First, the sampling-based determinization strategy requires to generate many sampled trajectories from the initial state before entering the main loop (Lines 3 to 5), as well as from each reachable reference state inside the loop (Lines 10 to 13). Indeed, many plans must be generated from different sampled domains to ensure a sufficient coverage of the long-term effects of actions in the plans. Second, contrary to RFF, HRFF can not search for single reachable states where no policy is defined, because there are infinite but, above all, uncountable. Instead, it tracks reachable reference points (Line 27) since they are countable: if two reachable single states are in the same $\tilde{\delta}$-cell of the continuous subspace, where there is no policy attached (policy query at Line 25 fails), then they will be merged in the same reference point, which represents their equivalence class for $\sim^{\tilde{\delta}}$. Third, HRFF computes and updates statistics about Metric-FF plans' density and average length to the goal, optionally asking Metric-FF to directly optimize them in the solution

Algorithm 1: HRFF

input : I: initial state, $\mathcal{M}$: PPDDL-based HMDP, N: number of Monte-Carlo samples, δ: size of highest-level cells (initial discretization)
output: $\mathcal{H}_\delta^\pi$: hybrid hierarchical hash table encoding the solution policy

1 Procedure main()
2 $policyProb \leftarrow 0$; $referenceStates \leftarrow$ Empty set of states;
3 **if** *sampling-based determinization* **then**
4 $\quad \mathcal{P} \leftarrow$ sampling-based determinization of $\mathcal{M}$;
5 $\quad$ **for** $1 \leqslant i \leqslant N$ **do** generate_trajectory$(\mathcal{P}, I)$;
6 **else** $\mathcal{P} \leftarrow$ mode-based determinization of $\mathcal{M}$;
7 **repeat**
8 $\quad$ compute_reachability();
9 $\quad$ **for** $s \in referenceStates$ **do**
10 $\quad\quad$ **if** *sampling-based determinization* **then**
11 $\quad\quad\quad \mathcal{P} \leftarrow$ sampling-based determinization of $\mathcal{M}$;
12 $\quad\quad\quad$ **for** $1 \leqslant i \leqslant N$ **do**
13 $\quad\quad\quad\quad$ generate_trajectory$(\mathcal{P}, s)$;
14 $\quad\quad$ **else** generate_trajectory$(\mathcal{P}, s)$;
15 **until** $(1 - policyProb) < \epsilon$;
16 Procedure compute_reachability()
17 $referenceStates.clear()$; $policyProb \leftarrow 0$;
18 **for** $1 \leqslant i \leqslant N$ **do**
19 $\quad s \leftarrow I$;
20 $\quad$ **while** true **do**
21 $\quad\quad$ **if** $s \in G$ **then break** ; // goal state
22 $\quad\quad (v, \tilde{\delta}, b) \leftarrow \mathcal{H}_\delta^V$.find$(s)$;
23 $\quad\quad$ **if** $b =$ true **and** $v.data = +\infty$ **then**
24 $\quad\quad\quad$ **break** ; // dead-end
25 $\quad\quad (a, \tilde{\delta}, b) \leftarrow \mathcal{H}_\delta^\pi$.find$(s)$;
26 $\quad\quad$ **if** $b =$ false **or** $s \notin S_a$ **then**
27 $\quad\quad\quad referenceStates.insert\,(r_{\tilde{\delta}}(s))$;
28 $\quad\quad\quad policyProb \leftarrow policyProb + \frac{1}{N}$;
29 $\quad\quad\quad$ **break**;
30 $\quad\quad$ **else** $s \leftarrow$ sample next state from a;
31 $policyProb \leftarrow 1 - policyProb$;
32 Procedure generate_trajectory$(\mathcal{P}, s)$
33 $\varpi = (a^d_{i_1}, \cdots, a^d_{i_k}) \leftarrow$ solve $\mathcal{P}$ with Metric-FF ;
34 **if** ϖ *is empty* **then** update_hashtable$(\mathcal{H}_\delta^V, s, +\infty)$;
35 **else**
36 $\quad s' \leftarrow s$;
37 $\quad$ **for** $1 \leqslant j \leqslant k$ **do**
38 $\quad\quad m^{a^d_{i_j}}(s') \leftarrow$ update statistics and compute action value ;
39 $\quad\quad (v, \tilde{\delta}, b) \leftarrow \mathcal{H}_\delta^V$.find$(s')$;
40 $\quad\quad$ **if** $b =$ false **or** $v.data > m^{a^d_{i_j}}(s')$ **then**
41 $\quad\quad\quad$ update_hashtable$(\mathcal{H}_\delta^V, s', m^{a^d_{i_j}}(s'))$;
42 $\quad\quad\quad$ update_hashtable$(\mathcal{H}_\delta^\pi, s', a^d_{i_j})$;
43 $\quad\quad s' \leftarrow$ successor state of s' with action $a^d_{i_j}$ in $\mathcal{P}$;
44 Procedure update_hashtable$(\mathcal{H}_\delta, s, data)$
45 $(c, \tilde{\delta}, b) \leftarrow \mathcal{H}_\delta$.insert$(s)$;
46 **if** $b =$ false **then**
47 $\quad$ **if** $|data - c.data| > \epsilon$ **then** $\mathcal{H}_\delta$.refine$(c, \tilde{\delta}, s, data)$;
48 $\quad$ **else** $c.data \leftarrow data$;

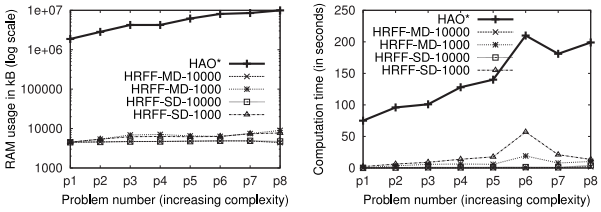


Figure 1: Rover domain (RAM is resident set size)

plan via additional fluents (Line 38). It then uses these statistics to decide if it replaces an action of the policy in a given cell by an action from the plan (Lines 40 to 42), which is required by hybrid settings and was not present at all in the original RFF. Finally, HRFF extensively relies on hybrid hierarchical hash tables to access or update the value or the policy over hybrid states (Lines 44 to 48) in a compact and efficient way.

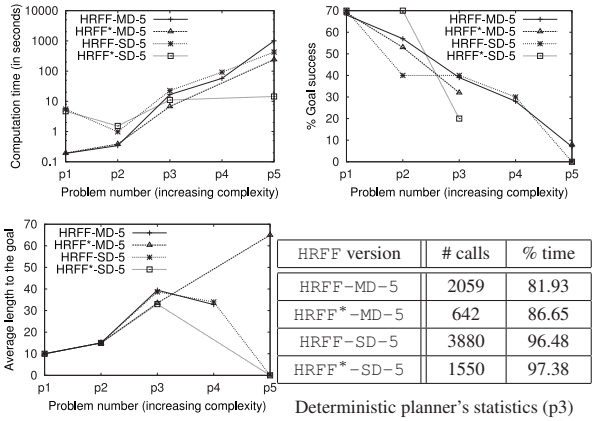
Experimental evaluation

We now present several experimentations conducted with HRFF. We note: HRFF−MD−δ (resp. HRFF−SD−δ) the version using mode-based (resp. sampling-based) determinization with an initial (implicit) cell discretization of δ ; HRFF*−[M,S]D−δ denotes the same variants, but using plan density and length optimization inside Metric−FF. For all tests, mode-based (resp. sampling-based) determinization was used with $N = 100$ (resp. 10) Monte-Carlo samples at each iteration.

Comparison with HAO* on the Rover domain. This domain was designed by NASA (Bresina et al. 2002; Meuleau et al. 2009). A rover has to take some pictures of rocks scattered in an outdoor environment, while navigating along predefined paths. It has been solved with success by the HAO* algorithm (Meuleau et al. 2009), which we could gracefully use for comparison purposes. HAO* is an optimal heuristic search algorithm, which performs dynamic programming updates on a subset of states, and uses a heuristic estimate of the optimal value function to decide which new states to explore during the search. In our settings, HAO* minimizes the average length of paths to the goal.

Figure 1 shows that HRFF uses nearly 3-order of magnitude lower RAM than HAO* on all problems. Moreover, HRFF's memory usage increases at a far lower rate than HAO*. We come to the same conclusion regarding CPU time consumption. We see that HRFF with $\delta = 1000$ takes more time and consumes more CPU than HRFF with $\delta = 10000$, which was expected because lower values of δ increase the chance to discover unexplored cells during the search, thus generating more Metric−FF trajectories. We also obtained (not included in the paper) the same average length to the goal with HRFF and with HAO* for all tested problems, although only HAO* is proven to be optimal.

Impact of the deterministic planner. We now compare different versions of HRFF on the Depot domain from the



HRFF version	# calls	% time
HRFF−MD−5	2059	81.93
HRFF*−MD−5	642	86.65
HRFF−SD−5	3880	96.48
HRFF*−SD−5	1550	97.38

Deterministic planner's statistics (p3)

Figure 2: Depot domain

numeric part of the International Planning Competition (see Figure 2). We added hybrid probabilistic effects (uniform, gaussian, discrete distributions) such that all policies must reach some dead-end states with a positive probability.

Computation times (top left plot) are globally the same for all versions of HRFF, except for small problems where mode-based determinization versions find solutions in less time. The top right and bottom left plots show that using plan density and length optimization inside Metric−FF (versions HRFF*−[M,S]D−5) does not need to improve the quality of policies in terms of percentage of goal success and average length to the goal. One reason might be that Metric−FF is not able to optimize fluent products, which would allow us to heuristically optimize these two metrics (see previous section). It can only optimize their sums, which does not seem to help in this domain. Finally, the bottom right table highlights 2 main different impacts of varying HRFF options: 1/ the portion of time used by all calls to the deterministic planner, as well as the number of calls to it, are higher with the sampling-based determinization strategy, because more hybrid states are visited than with the mode-based determinization strategy; 2/ for both determinization strategies, using plan density and length optimization inside Metric−FF results in far less calls to the deterministic planner, yet with the same global portion of solving time used by the deterministic planner. It shows that optimization settings in Metric−FF bring better actions in the policy (it is more focused towards the goal), but at a higher computation cost for the deterministic planner.

Results for navigation problems where using plan density and length optimization inside Metric−FF is helpful are discussed in (Teichteil-Königsbuch 2012). This paper also presents results for larger problems that were successfully solved by HRFF, with more than 700 binary state variables and 10 continuous state variables for the biggest ones.

Conclusion

We have presented the HRFF algorithm for solving large goal-oriented Hybrid Markov Decision Processes. HRFF determinizes on-the-fly the input probabilistic domain,

solves it from many different reachable states by using `Metric-FF`, and incrementally merges plans produced by the latter with the policy. Some action statistics based on plans' probabilities and lengths are updated during the search to improve the convergence of `HRFF`. The policy and the action statistics are encoded in hybrid hierarchical hash tables, which are novel, compact and efficient data structures to reason over hybrid state spaces. Experimental results show that `HRFF` outperforms `HAO*` by many order of magnitudes on the rover domain. It can also solve problems, whose size, complexity, and expressivity, were not yet tackled by any existing domain-independent HMDP algorithm, to the best of our knowledge.

References

Bresina, J.; Dearden, R.; Meuleau, N.; Ramkrishnan, S.; Smith, D.; and Washington, R. 2002. Planning under Continuous Time and Resource Uncertainty: A Challenge for AI. In *Proceedings of the Eighteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, 77–84. San Francisco, CA: Morgan Kaufmann.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Free Software Foundation. 2011. GNU Scientific Library. `http://www.gnu.org/software/gsl/`.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res. (JAIR)* 20:291–341.

Kolobov, A.; Mausam; and Weld, D. S. 2010. Classical planning in MDP heuristics: with a little help from generalization. In *ICAPS*, 97–104.

Kveton, B.; Hauskrecht, M.; and Guestrin, C. 2006. Solving factored MDPs with hybrid state and action variables. *J. Artif. Int. Res.* 27:153–201.

Lee, I. S., and Lau, H. Y. 2004. Adaptive state space partitioning for reinforcement learning. *Engineering Applications of Artificial Intelligence* 17(6):577 – 588.

Marecki, J.; Koenig, S.; and Tambe, M. 2007. A fast analytical algorithm for solving markov decision processes with real-valued resources. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, 2536–2541. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam. 2009. A heuristic search approach to planning with continuous resources in stochastic domains. *J. Artif. Int. Res.* 34:27–59.

Pouchol, M.; Ahmad, A.; Crespin, B.; and Terraz, O. 2009. A hierarchical hashing scheme for nearest neighbor search and broad-phase collision detection. *J. Graphics, GPU, & Game Tools* 14(2):45–59.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental plan aggregation for generating policies in MDPs. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, 1231–1238. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Teichteil-Königsbuch, F. 2008. Extending PPDDL1.0 to Model Hybrid Markov Decision Processes. In *Proceedings of the ICAPS 2008 workshop on A Reality Check for Planning and Scheduling Under Uncertainty*.

Teichteil-Königsbuch, F. 2012. Fast Incremental Policy Compilation from Plans in Hybrid Probabilistic Domains. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, 1010–1016. AAAI Press.

Younes, H. L. S., and Littman, M. L. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.