Noname manuscript No.
(will be inserted by the editor)

Planning in Hybrid Relational MDPs

Davide Nitti · Vaishak Belle ·
Tinne De Laet · Luc De Raedt

Received: date / Accepted: date

Abstract We study planning in relational Markov Decision Processes involving discrete and continuous states and actions, and an unknown number of objects. This combination of hybrid relational domains has so far not received a lot of attention. While both relational and hybrid approaches have been studied separately, planning in such domains is still challenging and often requires restrictive assumptions and approximations. We propose HYPE: a sample-based planner for hybrid relational domains that combines model-based approaches with state abstraction. HYPE samples episodes and uses the previous episodes as well as the model to approximate the Q-function. In addition, abstraction is performed for each sampled episode, this removes the complexity of symbolic approaches for hybrid relational domains. In our empirical evaluations, we show that HYPE is a general and widely applicable planner as it domains ranging from strictly discrete to strictly continuous to hybrid ones, handles intricacies such as unknown objects and relational models. Moreover, empirical results showed that abstraction provides significant improvements.

1 Introduction

Markov Decision Processes (MDPs) [47] are a natural and general framework for modeling probabilistic planning problems. Since the world is inherently relational, an important extension is that of relational MDPs [54], where the state is represented in terms of first-order logic, that is objects and relations between them. However, while significant progress has been made in developing robust planning algorithms for discrete, relational and continuous MDPs separately, the more intricate combination of those (hybrid relational) and settings with an unknown number of objects have received less attention.

The recent advances of probabilistic programming languages (e.g., BLOG [29], Church [11], ProbLog [21], Anglican [55], distributional clauses [12]) has

Address(es) of author(s) should be given

significantly improved the expressive power of formal representations for probabilistic models.

While it is known that these languages can express decision problems [46, 49], including hybrid relational MDPs, it is less clear if the inbuilt general-purpose inference system can cope with the challenges (e.g., scale, time constraints) posed by actual planning problems, and compete with existing state-of-the-art planners.

In this paper, we consider the problem of effectively planning in propositional and relational domains where reasoning and handling unknowns may be needed in addition to coping with mixtures of discrete and continuous variables. In particular, we adopt *dynamic distributional clauses* (DDC) [35,36] (an extension of distributional clauses [12] and based on distribution semantics [42]) to describe the MDP and perform inference. In such general settings, exact solutions may be intractable, and so approximate solutions are the best we can hope for. Popular approximate solutions include Monte Carlo methods to estimate the expected reward of a policy (i.e., policy evaluation).

Monte Carlo methods provide state-of-the-art results in probabilistic planners [22,18]. Monte Carlo planners have been mainly applied in discrete domains (with some notable exceptions, such as [26,5], for continuous domains). Typically, for continuous states, function approximation (e.g., linear regression) is applied. In that sense, one of the few Monte Carlo planners that works in arbitrary MDPs with no particular assumptions is Sparse Sampling (SST) [17]; but as we demonstrate later, it is often slow in practice. We remark that most, if not all, Monte Carlo methods require only a way to sample from the model of interest. While this property seems desirable, it prevents us from exploiting the actual probabilities of the model, as discussed (but unaddressed) in [18]. In this paper we address this issue proposing a planner that exploits the knowledge of the model via importance sampling to perform policy evaluation.

The first contribution of this paper is HYPE: a conceptually simple but powerful planning algorithm for a given (hybrid relational) MDP in DDC. However, HYPE can be adapted for other languages, such as RDDL [40]. The proposed planner exploits the knowledge of the model via importance sampling to perform policy evaluation, and thus, policy improvement. Importance sampling has been used in off-policy Monte Carlo methods [38,43,44], where policy evaluation is performed using trajectories sampled from another policy. We remark that standard off-policy Monte Carlo methods have been used in reinforcement learning, which are essentially model-free settings. In our setting, given a planning domain, the proposed planner introduces a new off-policy method that exploits the model and works, under weak assumptions, in discrete, relational, continuous, hybrid domains as well as those with an unknown number of objects.

The second contribution of this paper is a sample-based abstraction algorithm for HYPE. In particular, using individual samples of trajectories, it removes irrelevant facts from the sampled states with an approach based on logical regression. There exists several exact methods that perform symbolic dynamic programming, that is dynamic programming at the level of abstract

states (set of states). Those methods has been successfully used in relational domains [19,53]. However, abstraction is more challenging in hybrid relational domains, even though some attempts have been made [41,57] in propositional domains, under expressivity restrictions. To overcome the complexity of logical regression in general hybrid relational domains we perform abstraction at the level of sampled episodes. Such an approach carries over the benefits of symbolic methods to sampling approaches. We provide detailed derivations behind this abstraction, and show that it comes with a significant performance improvement.

The first contribution is based on the previous paper [34] and the second contribution on the workshop paper [33]. This paper extends previous works with an algorithm for logical regression to abstract samples, a more detailed theoretical justification for abstraction and additional experiments.

2 Background

2.1 Markov Decision Processes

In a MDP, a putative agent is assumed to interact with its environment, described using a set S of *states*, a set A of *actions* that the agent can perform, a *transition function* $p : S \times A \times S \to [0, 1]$, and a *reward function* $R : S \times A \to \mathbb{R}$. That is, when in state s and on doing a, the probability of reaching s' is given by $p(s'|s, a)$, for which the agent receives the reward $R(s, a)$. The agent is taken to operate over a finite number of time steps $t = 0, 1, \ldots, T$, with the goal of maximizing the expected reward: $\mathbb{E}[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)] = \mathbb{E}[G(E)]$, where $\gamma \in [0, 1]$ is a discount factor, $E =< s_0, a_0, s_1, a_1, ..., s_T, a_T >$ is the state and action sequence called episode and $G(E) = \sum_{t=0}^{T} \gamma^t R(s_t, a_t)$ is the total discounted reward of E.

This paper focuses on maximizing the reward in a finite horizon MDP; however the same ideas are extendable for infinite horizons. This is achieved by computing a (deterministic) policy $\pi : S \times D \to A$ that determines the agent's action at state s and remaining steps d (horizon). The expected reward starting from state s_t and following a policy π is called the *value function* (V-function):

$$V_d^\pi(s_t) = \mathbb{E}[G(E_t)|s_t, \pi] = \mathbb{E}\left[\sum_{k=t}^{t+d} \gamma^{k-t} R(s_k, a_k) \mid s_t, \pi\right]. \quad (1)$$

Where $E_t =< s_t, a_t, s_{t+1}, a_{t+1}, ..., s_T, a_T >$ is the subset of E from time t. Furthermore, the expected reward starting from state s_t while executing action a_t and following a policy π is called the *action-value function* (Q-function):

$$Q_d^\pi(s_t, a_t) = \mathbb{E}[G(E_t)|s_t, a_t, \pi] = \mathbb{E}\left[\sum_{k=t}^{t+d} \gamma^{k-t} R(s_k, a_k) \mid s_t, a_t, \pi\right]. \quad (2)$$

Since $T = t + d$, in the following formulas we will use T for compactness. An *optimal policy* π^* is a policy that maximizes the V-function for all states. The optimal policy satisfies the Bellman equation:

$$V_d^*(s_t) = max_a\Big(R(s_t, a_t) + \gamma \int_{s_{t+1}} p(s_{t+1}|s_t, a_t)V_{d-1}^*(s_{t+1})ds_{t+1}\Big). \quad (3)$$

This formula is used to solve the MDP in value iteration methods.

Alternatively, sample-based planners use Monte Carlo methods to solve an MDP and find a (near) optimal policy. Such planners simulate (by sampling) interaction with the environment in episodes $E^m = <s_0^m, a_0^m, s_1^m, a_1^m, ..., s_T^m, a_T^m>$, following some policy π. Each episode is a trajectory of T time steps, and we let s_t^m denote the state visited at time t during episode m. (So, after M episodes, $M \times T$ states would be explored). After or during episode generation, the sample-based planner updates $Q_d(s_t^m, a_t^m)$ for each t according to a backup rule, for example, averaging the total rewards obtained starting from (s_t^m, a_t^m) till the end. The policy is improved using a strategy that trades off exploitation and exploration, e.g., the ϵ-greedy strategy. In this case the policy used to sample the episodes is not deterministic; we indicate with $\pi(a_t|s_t)$ the probability to select action a_t in state s_t under the policy π. Under certain conditions, after a sufficiently large number of episodes, the policy converges to a (near) optimal policy, and the planner can execute the greedy policy $argmax_a Q_d(s_t, a)$.

2.2 Logic programming

In this section we briefly introduce logic programming concepts. See [32, 2, 25] for an extensive introduction.

An atomic formula (atom) is a predicate applied to a list of terms that represents objects. A term is a constant symbol, a logical variable or a function (functor) applied to terms. For example, $\texttt{inside}(\texttt{1}, \texttt{2})$ is an atomic formula, where `inside` is a predicate, sometimes called relation, and $\texttt{1}, \texttt{2}$ are terms, in particular constant symbols that refer to objects. A literal is an atomic formula or a negated atomic formula. A *clause*, in logic programming, is a first-order formula with a head (atom), and a body (a list of literals). For example, the clause

$$\texttt{inside}(\texttt{A}, \texttt{B}) \leftarrow \texttt{inside}(\texttt{A}, \texttt{C}), \texttt{inside}(\texttt{C}, \texttt{B}).$$

states that for all $\texttt{A}, \texttt{B}$ and $\texttt{C}$, $\texttt{A}$ is inside $\texttt{B}$ if $\texttt{A}$ is inside $\texttt{C}$ and $\texttt{C}$ is inside $\texttt{B}$ (transitivity property). $\texttt{A}, \texttt{B}$ and $\texttt{C}$ are logical variables, that informally refer to an arbitrary object or value. A clause usually contains non-ground literals, that is, literals with logical variables (e.g., $\texttt{inside}(\texttt{A}, \texttt{B})$). A clause with logical variables is assumed to be preceded by universal quantifiers for each logical variable, e.g., in the above clause: $\forall \texttt{A}, \forall \texttt{B}, \forall \texttt{C}$. A substitution θ replaces the variables with other terms. For example, for $\theta = \{\texttt{A} = \texttt{1}, \texttt{B} = \texttt{2}, \texttt{C} = \texttt{3}\}$ the above clause becomes:

$$\texttt{inside}(\texttt{1}, \texttt{2}) \leftarrow \texttt{inside}(\texttt{1}, \texttt{3}), \texttt{inside}(\texttt{3}, \texttt{1})$$

and states that if $\texttt{inside}(1,3)$ and $\texttt{inside}(3,1)$ are true, then $\texttt{inside}(1,2)$ is true. We indicate with $\theta = \texttt{mgu}(\texttt{A},\texttt{B})$ the most general unifier, i.e. the most general substitution θ that makes $\texttt{A}\theta = \texttt{B}\theta$.

An Herbrand interpretation I is a set of ground atomic formulas that are assumed to be true. The facts not in I are assumed to be false. In this paper, Herbrand interpretations represent states. For example, $I = \{\texttt{inside}(1,3), \texttt{inside}(2,3)\}$ represents a world state where 1 and 2 are inside 3, any other fact is false.

In this paper we refer to complete (full) states and partial (abstract) states. In a full state any fact has an assignment true/false, and any other variable has a value. In a partial (abstract) state some of the facts or variables have an assignment. The remaining facts and variables are left undefined. Formally, an abstract state is a conjunctive formula $\mathcal{F}$ that represents the set of complete states that satisfies $\mathcal{F}$, that is, $\mathcal{F} = l_1 \wedge ... \wedge l_n$ where all variables are existentially quantified and each literal l_i is either an atom or a negated atom. We will extend the usual notions of substitution, unification and subsumption to these expressions. In addition, an abstract state $\mathcal{F}$ subsumes a state s (notation $s \preceq \mathcal{F}$) if and only if there exists a substitution θ such that $\mathcal{F}\theta \subseteq s$. For example, the abstract state $\texttt{on}_\texttt{t}(1,2), \texttt{not}(\texttt{on}_\texttt{t}(2,\texttt{table}))$ represents all the states where object 1 is on top of object 2 and 2 is not on a table. An abstract state might contain logical variables, e.g., $\texttt{on}_\texttt{t}(1,\texttt{A})$ represents the set of all the states where 1 is on top of an arbitrary object. An example of such state is $\texttt{on}_\texttt{t}(1,2), \texttt{on}_\texttt{t}(2,3)$, subsumed by $\texttt{on}_\texttt{t}(1,\texttt{A})$: $\texttt{on}_\texttt{t}(1,2), \texttt{on}_\texttt{t}(2,3) \preceq \texttt{on}_\texttt{t}(1,\texttt{A})$. In this paper we consider only grounded abstract states, that is, without logical variables.

2.3 Relational MDPs

In first-order (relational) MDPs, the state is represented in terms of logic formulas. In particular, in relational MDPs based on logic programming, a state is a Herbrand interpretation and the actions are described as facts. The state transition model and the reward function are compactly defined in terms of probabilistic rules exploiting first-order logic. For example, in a blocksworld we can say that if $\texttt{on}(\texttt{A},\texttt{C}), \texttt{clean}(\texttt{B})$ holds then action $\texttt{move}(\texttt{A},\texttt{B})$ will add $\texttt{on}(\texttt{A},\texttt{B})$ with probability 0.9 to the state and remove $\texttt{on}(\texttt{A},\texttt{C}), \texttt{clean}(\texttt{B})$, otherwise with probability 0.1 the state will remain unchanged. In addition, it is often convenient to define when an action is applicable in a given state. This can be specified again in terms of rules (clauses). The conditions that make an action applicable are often called preconditions.

A relational MDP can be solved using the Bellman equation applied to abstract states with logical regression, instead of single states individually. This method is called Symbolic Dynamic Programming (SDP), and it has been successfully used to solve big MDPs efficiently [19,53,16,13]. Similar principles have been applied in (propositional) continuous and hybrid domains [41,57]. Despite the effectiveness of such approaches, they make restrictive assumptions (e.g., deterministic transition model for continuous variables) to keep exact in-

ference tractable. For more general domains approximations are needed [56]. Another issue of SDP is keeping the structures that represent the V-function compact. Despite the recent progress, and the availability of regression methods for inference in hybrid domains [3], SDP remains a challenging approach in general hybrid relational domains, including MDPs where the number of variables can change over time.

In Section 5 we will show how to simplify abstraction by performing regression on the sampled episodes.

3 Dynamic Distributional Clauses

Standard relational MDPs cannot handle continuous variables. To overcome this limitation we consider hybrid relational MDPs formulated using probabilistic logic programming [20,12,35]. In particular, we adopt (Dynamic) Distributional Clauses [35,12], an expressive probabilistic language that supports discrete and continuous variables and an unknown number of objects, in the spirit of BLOG [29].

A *distributional clause* (DC) is of the form $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n}$, where the $\mathtt{b_i}$ are literals and $\sim$ is a binary predicate written in infix notation. The intended meaning of a distributional clause is that each ground instance of the clause $(\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n})\theta$ defines the random variable $\mathtt{h}\theta$ as being distributed according to $\mathcal{D}\theta$ whenever all the $\mathtt{b_i}\theta$ hold, where θ is a substitution. Furthermore, a term $\simeq(d)$ constructed from the reserved functor $\simeq/1$ represents the value of the random variable d.

Example 1 Consider the following clauses:

$$\mathtt{n} \sim \mathtt{poisson}(6). \tag{4}$$

$$\mathtt{pos}(\mathtt{P}) \sim \mathtt{uniform}(1, 10) \leftarrow \mathtt{between}(1, \simeq(\mathtt{n}), \mathtt{P}). \tag{5}$$

$$\mathtt{left}(\mathtt{A}, \mathtt{B}) \leftarrow \simeq(\mathtt{pos}(\mathtt{A})) > \simeq(\mathtt{pos}(\mathtt{B})). \tag{6}$$

Capitalized terms such as $\mathtt{P}, \mathtt{A}$ and $\mathtt{B}$ are logical variables, which can be substituted with any constant. Clause (4) states that the number of people $\mathtt{n}$ is governed by a Poisson distribution with mean 6; clause (5) models the position $\mathtt{pos}(\mathtt{P})$ as a random variable uniformly distributed from 1 to 10, for each person $\mathtt{P}$ such that $\mathtt{P}$ is between 1 and $\simeq(\mathtt{n})$. Thus, if the outcome of $\mathtt{n}$ is two (i.e., $\simeq(n) = 2$) there are 2 independent random variables $\mathtt{pos}(1)$ and $\mathtt{pos}(2)$. Finally, clause (6) shows how to define the predicate $\mathtt{left}(\mathtt{A}, \mathtt{B})$ from the positions of any $\mathtt{A}$ and $\mathtt{B}$. Ground atoms such as $\mathtt{left}(1, 2)$ are binary random variables that can be true or false, while terms such as $\mathtt{pos}(1)$ represent random variables that can take concrete values from the domain of their distribution.

A *distributional program* is a set of distributional clauses (some of which may be deterministic) that defines a distribution over possible worlds, which in turn defines the underlying semantics. A possible world is generated starting

from the empty set $S = \emptyset$; for each distributional clause $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b_1}, ..., \mathtt{b_n}$, whenever the body $\{\mathtt{b_1}\theta, ..., \mathtt{b_n}\theta\}$ is true in the set S for the substitution θ, a value v for the random variable $\mathtt{h}\theta$ is sampled from the distribution $\mathcal{D}\theta$ and $\simeq(h\theta) = v$ is added to S. This is repeated until a fixpoint is reached, i.e., no further variables can be sampled. *Dynamic distributional clauses* (DDC) extend distributional clauses in admitting temporally-extended domains by associating a time index to each random variable.

Example 2 Let us consider an object search scenario (*objsearch*) used in the experiments, in which a robot looks for a specific object in a shelf. Some of the objects are visible, others are occluded. The robot needs to decide which object to remove to find the object of interest. Every time the robot removes an object, the objects behind it become visible. This happens recursively, i.e., each new uncovered object might occlude other objects. The number and the types of occluded objects depend on the object covering them. For example, a box might cover several objects because it is big. This scenario involves an unknown number of objects and can be written as a partially observable MDP. However, it can be also described as a MDP in DDC where the state is the type of visible objects; in this case the state grows over time when new objects are observed or shrink when objects are removed without uncovering new objects. The probability of observing new objects is encoded in the state transition model, for example:

$$\mathtt{type(X)}_{\mathtt{t+1}} \sim \mathtt{val(T)} \leftarrow \simeq(\mathtt{type(X)}_{\mathtt{t}}) = \mathtt{T},\ \mathtt{not(removeObj(X))}. \tag{7}$$

$$\mathtt{numObjBehind(X)}_{\mathtt{t+1}} \sim \mathtt{poisson(1)} \ \leftarrow \simeq(\mathtt{type(X)}_{\mathtt{t}}) = \mathtt{box}, \mathtt{removeObj(X)}. \tag{8}$$

$$\begin{aligned}\mathtt{type(ID)}_{\mathtt{t+1}} \sim \mathtt{finite}([0.2 : \mathtt{glass}, 0.3 : \mathtt{cup}, 0.4 : \mathtt{box}, 0.1 : \mathtt{can}]) \leftarrow \\ \simeq(\mathtt{type(X)}_{\mathtt{t}}) = \mathtt{box}, \mathtt{removeobj(X)}, \simeq(\mathtt{numObjBehind(X)}_{\mathtt{t+1}}) = \mathtt{N}, \mathtt{getLastID(Last)}_{\mathtt{t}}, \\ \mathtt{NewID\ is\ Last} + 1, \mathtt{EndNewID\ is\ NewID} + \mathtt{N}, \mathtt{between(NewID, EndNewID, ID)}.\end{aligned} \tag{9}$$

Clause (7) states that the type of each object remains unchanged when we do not perform a remove action. Otherwise, if we remove the object, its type is removed from the state at time $t+1$ because it is not needed anymore. Clauses (8) and (9) define the number and the type of objects behind a box `X`, added to the state when we perform a remove action on `X`. Similar clauses are defined for other types. The predicate $\mathtt{getLastID(Last)}_{\mathtt{t}}$ returns the highest object `ID` in the state and is needed to make sure that any new object has a different `ID`.

DDC can be easily extended to define MDPs. In the previous example we showed how to define a state transition model. To complete the MDP specification we need to define a reward function $R(s_t, a_t)$, the terminal states that indicate when the episode terminates, and the applicability of an action a_t is

a state s_t as in PDDL. For *objsearch* we have:

$$\texttt{stop}_\texttt{t} \leftarrow \simeq(\texttt{type}(\texttt{X})_\texttt{t}) = \texttt{can}.$$
$$\texttt{reward}(20)_\texttt{t} \leftarrow \texttt{stop}_\texttt{t}.$$
$$\texttt{reward}(-1)_\texttt{t} \leftarrow \texttt{not}(\texttt{stop}_\texttt{t}).$$

That is, a state is terminal when we observe the object of interest (e.g., a can), for which a reward of 20 is obtained. The remaining states are nonterminal with reward -1. To define action applicability we use a set of clauses of the form

$$\texttt{applicable}(\texttt{action})_\texttt{t} \leftarrow \texttt{preconditions}_\texttt{t}.$$

For example, action `removeobj` is applicable for each object in the state, that is when its type is defined with an arbitrary value `Type`:

$$\texttt{applicable}(\texttt{removeobj}(\texttt{X}))_\texttt{t} \leftarrow \simeq(\texttt{type}(\texttt{X})_\texttt{t}) = \texttt{Type}.$$

In DDC a (complete) state contains facts as in standard relational MDPs and statements $\simeq(\texttt{variable}) = \texttt{v}$ (the value of `variable` is `v`) for continuous or categorical random variables, e.g.: $\texttt{on}_\texttt{t}(1, 2), \texttt{clean}_\texttt{t}(1), \texttt{on}_\texttt{t}(1, \texttt{table}), \simeq (\texttt{energy}_\texttt{t}) = 1.3$. All the facts not in the state are assumed false and all variables not in the state are assumed not existent.

4 HYPE: Planning by Importance Sampling

In this section we introduce HYPE (= *hybrid episodic planner*), a planner for hybrid relational MDPs described in DDC. HYPE is a planner that adopts an *off-policy strategy* [47] based on importance sampling and *derived* from the transition model. Related work is discussed more comprehensively in Section 6, but as we note later, sample-based planners typically only require a generative model (a way to generate samples) and do not exploit the model of the MDP (i.e., the actual probabilities) [18]. In our case, this knowledge leads to an effective planning algorithm that works in discrete, continuous, and hybrid domains, and/or domains with an unknown number of objects under weak assumptions. Moreover, HYPE performs abstraction of sampled episodes. In this section we introduce HYPE without abstraction; the latter will be introduced in Section 5.

4.1 Basic algorithm

In a nutshell, HYPE samples episodes E^m and stores for each visited state s_t^m an estimation of the V-function (e.g., the total reward obtained from that state). The action selection follows an given strategy (e.g., ϵ-greedy), where the Q-function is estimated as the immediate reward plus the weighted average of the previously stored V-function points at time $t + 1$. This is justified by

Algorithm 1 HYPE without abstraction

1: **function** SAMPLEEPISODE(d, s_t^m, m) ▷ Horizon d, state s_t^m in episode m
2: **if** $d = 0$ **then**
3: **return** 0
4: **end if**
5: **for** each applicable action a in s_t^m **do** ▷ Q-function estimation
6: $\tilde{Q}_d^m(s_t^m, a) \leftarrow R(s_t^m, a) + \gamma \frac{\sum_{i=0}^{m-1} w^i \tilde{V}_{d-1}^i(s_{t+1}^i)}{\sum_{i=0}^{m-1} w^i}$
7: **end for**
8: $a_t^m \leftarrow policy(\{\tilde{Q}_d^m(s_t^m, a)\})$ ▷ action policy, e.g., ϵ-greedy
9: sample $s_{t+1}^m \sim p(s_{t+1} \mid s_t^m, a_t^m)$ ▷ sample next state
10: $G(E_t^m) \leftarrow R(s_t^m, a_t^m) + \gamma \cdot$SAMPLEEPISODE $(d-1, s_{t+1}^m, m)$ ▷ recursive call
11: $\tilde{V}_d^m(s_t^m) \leftarrow G(E_t^m)$
12: store $(s_t^m, \tilde{V}_d^m(s_t^m), d)$
13: **return** $\tilde{V}_d^m(s_t^m)$ ▷ V-function estimation for s_t^m at horizon d
14: **end function**

means of importance sampling as explained later. The essential steps of our planning system HYPE are given in Algorithm 1.

The algorithm realizes the following key ideas:

- $\tilde{Q}$ and $\tilde{V}$ denote approximations of the Q and V-function respectively.
- Lines 8 select an action according to a given strategy.
- Lines 9-12 sample the next step and recursively the remaining episode of total length T, then stores the total discounted reward $G(E_t^m)$ starting from the current state s_t^m. This quantity can be interpreted as a sample of the expectation in formula (1), thus an estimator of the V-function. For this and other reasons explained later, $G(E_t^m)$ is stored as $\tilde{V}_d^m(s_t^m)$.
- Most significantly, line 6 approximates the Q-function using the *weighted average* of the stored $\tilde{V}_{d-1}^i(s_{t+1}^i)$ points:

$$\tilde{Q}_d^m(s_t^m, a) \leftarrow R(s_t^m, a) + \gamma \frac{\sum_{i=0}^{m-1} w^i \tilde{V}_{d-1}^i(s_{t+1}^i)}{\sum_{i=0}^{m-1} w^i}, \tag{10}$$

  where w^i is a *weight function* for episode i at state s_{t+1}^i. The weight exploits the transition model and is defined as:

$$w^i = \frac{p(s_{t+1}^i \mid s_t^m, a)}{q(s_{t+1}^i)} \alpha^{(m-i)}. \tag{11}$$

Here, for evaluating an action a at the current state s_t, we let w^i be the ratio of the transition probability of reaching a stored state s_{t+1}^i and the probability used to sample s_{t+1}^i, denoted using q. Recent episodes are considered more significant than previous ones, and so α is a parameter for realizing this. We provide a detailed justification for line 6 below.

We note that line 6 requires us to go over a finite set of actions, and so in the presence of continuous action spaces (e.g., real-valued parameter for a move action), we can discretize the action space or sample from it. More sophisticated approaches are possible [9,45].

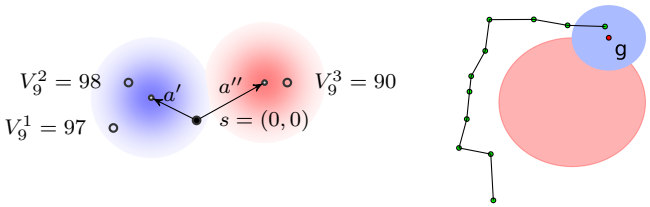


Fig. 1 Left: weight computation for the objpush domain. Right: a sampled episode that reaches the goal (blue), and avoids the undesired region (red).

Example 3 As a simple illustration, consider the following example called *objpush*. We have an object on a table and an arm that can push the object in a set of directions; the goal is to move the object close to a point g, avoiding an undesired region (Fig. 1). The state consists of the object position (x, y), with push actions parameterized by the displacement $(\texttt{DX}, \texttt{DY})$. The state transition model is a Gaussian around the previous position plus the displacement:

$$\texttt{pos(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{\texttt{t}}) + (\texttt{DX}, \texttt{DY}), \texttt{cov}) \leftarrow \texttt{push}(\texttt{ID}, (\texttt{DX}, \texttt{DY})). \tag{12}$$

The clause is valid for any object `ID`; nonetheless, for simplicity, we will consider a scenario with a single object. The terminal states and rewards in DDC are:

$$\begin{aligned} \texttt{stop}_{\texttt{t}} &\leftarrow \texttt{dist}(\simeq(\texttt{pos(A)}_{\texttt{t}}), (0.6, 1.0)) < 0.1. \\ \texttt{reward}(100)_{\texttt{t}} &\leftarrow \texttt{stop}_{\texttt{t}}. \\ \texttt{reward}(-1)_{\texttt{t}} &\leftarrow \texttt{not}(\texttt{stop}_{\texttt{t}}), \texttt{dist}(\simeq(\texttt{pos(A)}_{\texttt{t}}), (0.5, 0.8)) >= 0.2. \\ \texttt{reward}(-10)_{\texttt{t}} &\leftarrow \texttt{not}(\texttt{stop}_{\texttt{t}}), \texttt{dist}(\simeq(\texttt{pos(A)}_{\texttt{t}}), (0.5, 0.8)) < 0.2. \end{aligned} \tag{13}$$

That is, a state is terminal when there is an object close to the goal point $(0.6, 1.0)$ (i.e., distance lower than 0.1), and so, a reward of 100 is obtained. The nonterminal states have reward -10 whether inside an undesired region centered in $(0.5, 0.8)$ with radius 0.2, and $R(s_t, a_t) = -1$ otherwise.

Let us assume we previously sampled some episodes of length $T = 10$, and we want to sample the $m = 4$-th episode starting from $s_0 = (0, 0)$. We compute $\tilde{Q}_{10}^m((0, 0), a)$ for each action a (line 6). Thus, we compute the weights w^i using (11) for each stored sample $\tilde{V}_9^i(s_1^i)$. For example, Figure 1 shows the computation of $\tilde{Q}_{10}^m((0, 0), a)$ for action $a' = (-0.4, 0.3)$ and $a'' = (0.9, 0.5)$, where we have three previous samples $i = \{1, 2, 3\}$ at depth 9. A shadow represents the likelihood $p(s_1^i | s_0 = (0, 0), a)$ (left for a' and right for a''). The weight w^i (11) for each sample s_1^i is obtained by dividing this likelihood by $q(s_1^i)$ (with $\alpha = 1$). If $q^i(s_1^i)$ is uniform over the three samples, sample $i = 2$ with total reward $\tilde{V}_9^2(s_1^2) = 98$ will have higher weight than samples $i = 1$ and $i = 3$. The situation is reversed for a''. Note that we can estimate

$\tilde{Q}_d^m(s_t^m, a)$ using episodes i that may never encounter s_t^m, a_t provided that $p(s_{t+1}^i | s_t^m, a_t) > 0$.

4.2 Computing the (Approximate) Q-Function

The purpose of this section is to motivate our approximation to the Q-function using the weighted average of the V-function points in line 6. Let us begin by expanding the definition of the Q-function from (2) as follows:

$$Q_d^\pi(s_t, a_t) = R(s_t, a_t) + \gamma \int_{s_{t+1:T}, a_{t+1:T}} G(E_{t+1}) p(s_{t+1:T}, a_{t+1:T} | s_t, a_t, \pi) ds_{t+1:T}, a_{t+1:T}, \tag{14}$$

where $G(E_{t+1})$ is the total (discounted) reward from time $t+1$ to T: $G(E_{t+1}) = \sum_{k=t+1}^{T} \gamma^{k-t-1} R(s_k, a_k)$. Given that we sample trajectories from the target distribution $p(s_{t+1:T}, a_{t+1:T} | s_t, a_t, \pi)$, we obtain the following approximation to the Q-function equaling the true value in the sampling limit:

$$Q_d^\pi(s_t, a_t) \approx R(s_t, a_t) + \frac{1}{N} \gamma \sum_i G(E_{t+1}^i). \tag{15}$$

Policy evaluation can be performed sampling trajectories using another policy, this is called *off-policy* Monte Carlo [47]. For example, we can evaluate the greedy policy while the data is generated from a randomized one to enable exploration. This is generally performed using (normalized) *importance sampling* [43]. We let w^i be the ratio of the target and proposal distributions to restate the sampling limit as follows:

$$Q_d^\pi(s_t, a_t) \approx R(s_t, a_t) + \frac{1}{\sum w^i} \gamma \sum_i w^i G(E_{t+1}^i). \tag{16}$$

In standard off-policy Monte Carlo the proposal distribution is of the form:

$$p(s_{t+1:T}, a_{t+1:T} | s_t, a_t, \pi') = \prod_{k=t}^{T-1} \pi'(a_{k+1} | s_{k+1}) p(s_{k+1} | s_k, a_k)$$

The target distribution has the same form, the only difference is that the policy is π instead of π'. In this case the weight becomes equal to the policy ratio because the transition model cancels out. This is desirable when the model is not available, for example in model-free Reinforcement Learning. The question is whether the availability of the transition model can be used to improve off-policy methods. This paper shows that the answer to that question is positive.

We will now describe the proposed solution. Instead of considering only trajectories that start from s_t, a_t as samples, we consider all sampled trajectories from time $t+1$ to T. Since we are ignoring steps before $t+1$, the proposal distribution for sample i is the marginal

$$p(s_{t+1:T}, a_{t+1:T} | s_0, \pi^i) = q^i(s_{t+1}) \pi^i(a_{t+1} | s_{t+1}) \prod_{k=t+1}^{T-1} \pi^i(a_{k+1} | s_{k+1}) p(s_{k+1} | s_k, a_k),$$

where q^i is the marginal probability $p(s_{t+1}|s_0, \pi^i)$. To compute $\tilde{Q}_d^m(s_t^m, a)$ we use (16), where the weight w^i (for $0 \leq i \leq m-1$) becomes the following:

$$\frac{p(s_{t+1}^i|s_t^m, a)\pi(a_{t+1}^i|s_{t+1}^i)\prod_{k=t+1}^{T-1}\pi(a_{k+1}^i|s_{k+1}^i)p(s_{k+1}^i|s_k^i, a_k^i)}{q^i(s_{t+1}^i)\pi^i(a_{t+1}^i|s_{t+1}^i)\prod_{k=t+1}^{T-1}\pi^i(a_{k+1}^i|s_{k+1}^i)p(s_{k+1}^i|s_k^i, a_k^i)}$$
$$= \frac{p(s_{t+1}^i|s_t^m, a)}{q^i(s_{t+1}^i)}\frac{\prod_{k=t}^{T-1}\pi(a_{k+1}^i|s_{k+1}^i)}{\prod_{k=t}^{T-1}\pi^i(a_{k+1}^i|s_{k+1}^i)} \tag{17}$$
$$\approx \frac{p(s_{t+1}^i|s_t^m, a)}{q^i(s_{t+1}^i)}\alpha^{(m-i)}. \tag{18}$$

Thus, we obtain line 6 in the algorithm given that $\tilde{V}_{d-1}^i(s_t^i) = G(E_{t+1}^i)$. In our algorithm the target (greedy) policy π is not explicitly defined, therefore the policy ratio is hard to compute. We replace the unknown policy ratio with a quantity proportional to $\alpha^{(m-i)}$ where $0 < \alpha \leq 1$; thus, formula (17) is replaced with (18). The quantity $\alpha^{(m-i)}$ becomes smaller for an increasing difference between the current episode index m and the i-th episode. Therefore, the recent episodes are weighted (on average) more than the previous ones, as in *recently-weighted average* applied in on-policy Monte Carlo [47]. This is justified because the policy is improved over time, thus recent episodes should have higher weight.

In general, $q^i(s_{t+1}^i)$ is not available in closed form; we adopt the following approximation:

$$q^i(s_{t+1}^i) = p(s_{t+1}^i|s_0, \pi_i) = \int_{s_t, a_t} p(s_{t+1}^i|s_t, a_t)p(s_t, a_t|s_0, \pi_i)$$
$$\approx \frac{1}{2D+1}\sum_{j=i-D}^{i+D} p(s_{t+1}^i|s_t^j, a_t^j). \tag{19}$$

Where we are assuming that $\pi^j \approx \pi^i$ for $i - D < j < i + D$, and the samples s_t^j, a_t^j refer to episode E^j. Each episode E^j is sampled from $p(s_{0:T}, a_{0:T}|s_0, \pi_j)$, thus samples (s_t^j, a_t^j) are distributed as $p(s_t, a_t|s_0, \pi_j)$ and are used in the estimation of the integral.

The likelihood $p(s_{t+1}^i|s_t^m, a)$ is required to compute the weight. This probability can be decomposed using the chain rule, e.g., for a state with 3 variables we have:

$$p(s_{t+1}^i|s_t^m, a) = p(v_3|v_2, v_1, s_t^m, a)p(v_2|v_1, s_t^m, a)p(v_1|s_t^m, a),$$

where $s_{t+1}^i = \{v_1, v_2, v_3\}$. In DDC this is performed evaluating the likelihood of each variable in v_i following the topological order defined in the DDC program. The target and the proposal distributions might be mixed distributions of discrete and continuous random variables; importance sampling can be applied in such distributions as discussed in [37, Chapter 9.8].

To summarize, for each state s_t^m, $Q(s_t^m, a_t)$ is evaluated as the immediate reward plus the weighted average of stored $G(E_{t+1}^i)$ points. In addition, for

each state s_t^m the total discounted reward $G(E_t^m)$ is stored. We would like to remark that we can estimate the Q-function also for states and actions that have never been visited, as shown in example 1. This is possible without using function approximations (beyond importance sampling).

4.3 Extensions

Our derivation follows a Monte Carlo perspective, where each stored point is the total discounted reward of a given trajectory: $\tilde{V}_d^m(s_t^m) \leftarrow G(E_t^m)$. However, following the Bellman equation, $\tilde{V}_d^m(s_t^m) \leftarrow max_a \tilde{Q}_d^m(s_t^m, a)$ can be used instead (replacing line 11 in Algorithm 1). The Q-function estimation formula in line 6 is not affected; indeed we can repeat the same derivation using the Bellman equation and approximate it with importance sampling:

$$\begin{aligned} Q_d^\pi(s_t, a_t) &= R(s_t, a_t) + \gamma \int_{s_{t+1}} V_{d-1}^\pi(s_{t+1}) p(s_{t+1}|s_t, a_t) ds_{t+1} \\ &\approx R(s_t, a) + \gamma \sum \frac{w^i}{\sum w^i} \tilde{V}_{d-1}^i(s_{t+1}^i) = \tilde{Q}_d^m(s_t, a_t), \end{aligned} \tag{20}$$

with $w^i = \frac{p(s_{t+1}^i|s_t, a_t)}{q^i(s_{t+1}^i)}$ and s_{t+1}^i the state sampled in episode i for which we have an estimation of $\tilde{V}_{d-1}^i(s_{t+1}^i)$, while $q^i(s_{t+1}^i)$ is the probability with which s_{t+1}^i has been sampled. This derivation is valid for a fixed policy π; for a changing policy we can make similar considerations to the previous approach and add the term $\alpha^{(m-i)}$. If we choose $\tilde{V}_{d-1}^i(s_{t+1}^i) \leftarrow G(E_{t+1}^i)$, we obtain the same result as in (10) and (18) for the Monte Carlo approach.

Instead of choosing between the two approaches we can use a linear combination, i.e., we replace line 11 with $\tilde{V}_d^m(s_t^m) \leftarrow \lambda G(E_t^m) + (1-\lambda) max_a \tilde{Q}_d^m(s_t^m, a)$. The analysis from earlier applies by letting $\lambda = 1$. However, for $\lambda = 0$, we obtain a local value iteration step, where the stored $\tilde{V}$ is obtained maximizing the estimated $\tilde{Q}$ values. Any intermediate value balances the two approaches (this is similar to, and inspired by, TD(λ) [47]). Another strategy consists in storing the maximum of the two: $\tilde{V}_d^m(s_t^m) \leftarrow max(G(E_t^m), max_a \tilde{Q}_d^m(s_t^m, a))$. In other words, we alternate Monte Carlo and Bellman backup according to which one has the highest value. This strategy works often well in practice; indeed it avoids a typical issue in (on-policy) Monte Carlo methods: bad policies or exploration lead to low rewards, averaged in the estimated Q/V-function.

4.4 Practical improvements

In this section we briefly discuss some practical improvements of HYPE. To evaluate the Q-function the algorithm needs to query all the stored examples, making the algorithm potentially slow. This issue can be mitigated with solutions used in instance-based learning, such as hashing and indexing. For example, in discrete domains we avoid multiple computations of the likelihood

and the proposal distribution for samples of the same state. In addition, assuming policy improvement over time, only the N_{store} most recent episodes are kept, since older episodes are generally sampled with a worse policy.

HYPE's algorithm relies on importance sampling to estimate the Q-function, thus we should guarantee that $p > 0 \Rightarrow q > 0$, where p is the target and q is the proposal distribution. This is not always the case, like when we sample the first episode. Nonetheless we can have an indication of the estimation reliability. In our algorithm we use $\sum w^i$ with expectation equal to the number of samples: $\mathbb{E}[\sum w^i] = m$. If $\sum w^i < thres$ the samples available are considered insufficient to compute $Q_d^m(s_t^m, a)$, thus action a can be selected according to an exploration policy. It is also possible to add a fictitious weighted point in line 6, that represents the initial $Q_d^m(s_t^m, a)$ guess. This can be used to exploit heuristics during sampling.

A more problematic situation is when, for some action a_t in some state s_t, we always obtain null weights, that is, $p(s_{t+1}^i | s_t, a_t) = 0$ for each of the previous episodes i, no matter how many episodes are generated. This issue is solved by adding noise to the state transition model, e.g., Gaussian noise for continuous random variables. This effectively 'smoothes' the V-function. Indeed the Q-function is a weighted sum of V-function points, where the weights are proportional to a noisy version of the state transition likelihood.

5 Abstraction

By exploiting the (relational) model, we can improve the algorithm by using abstract states, because often, only some parts of the state determine the total reward. The idea is to generalize the specific states into abstract states by removing the irrelevant facts (for the outcome of the episode). This resembles symbolic methods to exactly solve MDPs in propositional and relational domains [54]. The main idea in symbolic methods is to apply the Bellman equation on abstract states, using logical regression (backward reasoning). As described in Section 2.3 and 6, symbolic methods are more challenging in hybrid relational MDPs. The main issues are the intractability of the integral in the Bellman equation (3), and the complexity of symbolic manipulation in complex hybrid relational domains.

To overcome these difficulties, we propose to perform abstraction at the level of samples. The modified algorithm with abstraction is sketched in Algorithm 2. The main differences with Alg. 1 are:

- Q-function estimation from abstracted states (line 6)
- regression of the current state (line 11)
- the procedure returns the abstract state and its V-function, instead of the latter only (line 15). This is required for recursive regression.

Algorithm 2 HYPE with abstraction

function SAMPLEEPISODE(d, s_t^m, m) ▷ Horizon d, state s_t^m, episode m
  if $d = 0$ **then**
    return $(\emptyset, 0)$
  end if
  for each applicable action a in s_t^m **do** ▷ Q-function estimation
    $\tilde{Q}_d^m(s_t^m, a) \leftarrow R(s_t^m, a) + \gamma \frac{\sum_{i=0}^{m-1} w^i \tilde{V}_{d-1}^i(\hat{s}_{t+1}^i)}{\sum_{i=0}^{m-1} w^i}$
  end for
  $a_t^m \leftarrow policy(\{\tilde{Q}_d^m(s_t^m, a)\})$ ▷ action policy, e.g., ϵ-greedy
  sample $s_{t+1}^m \sim p(s_{t+1}|s_t^m, a_t^m)$
  $(\hat{s}_{t+1}^m, v) \leftarrow$SAMPLEEPISODE$(d-1, s_{t+1}^m, m)$
  $\hat{s}_t^m \leftarrow$ REGRESS$(\{R(s_t^m, a_t^m) \wedge \hat{s}_{t+1}^m\}, \{s_t^m, a_t^m, \hat{s}_{t+1}^m\})$ ▷ abstraction
  $G(E_t^m) \leftarrow R(s_t^m, a_t^m) + \gamma v$
  $\tilde{V}_d^m(s_t^m) \leftarrow G(E_t^m)$
  store $(\hat{s}_t^m, \tilde{V}_d^m(s_t^m), d)$
  return $(\hat{s}_t^m, \tilde{V}_d^m(s_t^m))$
end function

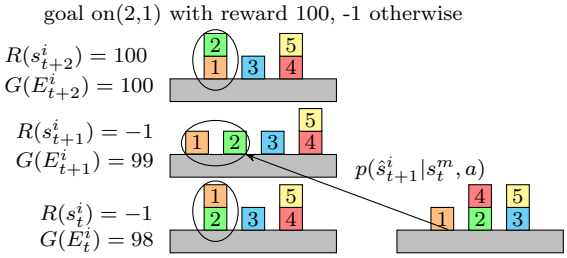


Fig. 2 Blocksworld with abstraction. Current full state on the right, and a sampled episode on the left. The abstracted states are circled.

5.1 Basic principles of abstraction

Before describing abstraction formally, let us consider the blocksworld example to give an intuition. Fig. 2 shows a sampled episode from the first state (bottom left) to the last state (top left) that ends in the goal state $\texttt{on}(2,1)$. Informally, the relevant part of the episode is the set of facts that are responsible for reaching the goal, or more generally responsible for obtaining a given total reward. This relevant part is called the abstracted episode. Fig. 2 shows the abstract states (circled) that together define the abstract episode. Intuitively, objects $3, 4, 5$ and their relations are irrelevant to reach the goal $\texttt{on}(2,1)$, and thus do not belong to the abstracted episode.

The abstraction helps to exploit the previous episodes in more cases, speeding up the convergence. For example, Fig. 2 shows the computation of a weight w^i (using (11)) to compute the Q-function of the (full) state s_t^m depicted on the right, exploiting the abstract state $\hat{s}_{t+1}^i$ indicated by the arrow (from episode i). If the action is moving 4 on top of 5 we have $p(\hat{s}_{t+1}^i|s_t^m, a) > 0 \Rightarrow w^i > 0$. Thus, the Q-function estimate $\tilde{Q}_d^m(s_t, a)$ will include $w^1 \cdot 99$ in the weighted

average (line 6 in Alg. 2), making the action appealing. In contrast, without abstraction all actions get weight 0, because the full state s^i_{t+1} is not reachable from s^m_t (i.e. $p(s^i_{t+1}|s^m_t, a) = 0$). Therefore, episode i cannot be used to compute the Q-function. For this reason abstraction requires fewer samples to converge to a near-optimal policy.

This idea is valid in continuous domains. For example, in the *objpush* scenario, the goal is to put any object in a given region; if the goal is reached, only one object is responsible, any other object is irrelevant in that particular state.

5.2 Mathematical Derivation

In this section we formalize sample-based abstraction and describe the assumptions that justify the Q-function estimation on abstract states (line 6 of Alg. 2).

5.2.1 Abstraction applied to importance sampling

The Q-function estimation (16) can be reformulated for abstract states as follows. For an episode from time t, $E_t = <s_t, a_t, ..., s_T, a_T>$, let us consider an arbitrary partition $E_t = \{\hat{E}_t, E'_t\}$ such that $G(E_t) = G(\hat{E}_t)$, i.e., the total reward depends only on $\hat{E}_t$. The relevant part of the episode has the form $\hat{E}_t = <\hat{s}_t, a_t, ..., \hat{s}_T, a_T>$, while $E'_t = E_t \backslash \hat{E}_t = <s'_t, ..., s'_T>$ is the remaining non-relevant part[1]. The partial episode $\hat{E}_t$ is called *abstract* because the irrelevant variables have been marginalized, in contrast E_t is called full or complete. The Q-function estimation (16) is reformulated for abstract states marginalizing irrelevant variables:

$$
\begin{aligned}
Q^\pi_d(s^m_t, a) &= \int_{E_t} p(E_t|s^m_t, a, \pi) G(E_t) dE_t = \\
&= \int_{\hat{E}_t} \left(\int_{E'_t} p(\hat{E}_t, E'_t|s^m_t, a, \pi) dE'_t \right) G(\hat{E}_t) d\hat{E}_t = \\
&= \int_{\hat{E}_t} p(\hat{E}_t|s^m_t, a, \pi) G(\hat{E}_t) d\hat{E}_t = \\
&= R(\hat{s}^m_t, a) + \gamma \int_{\hat{E}_{t+1}} p(\hat{E}_{t+1}|s^m_t, a, \pi) G(\hat{E}_{t+1}) d\hat{E}_{t+1} = \\
&= R(\hat{s}^m_t, a) + \gamma \int_{\hat{E}_{t+1}} \underbrace{\frac{p(\hat{E}_{t+1}|s^m_t, a, \pi)}{q(\hat{E}_{t+1})}}_{w(\hat{E}_{t+1})} q(\hat{E}_{t+1}) G(\hat{E}_{t+1}) d\hat{E}_{t+1} \\
&\approx R(\hat{s}^m_t, a) + \gamma \frac{1}{\sum_{i=0}^{m-1} w(\hat{E}^i_{t+1})} \sum_{i=0}^{m-1} w(\hat{E}^i_{t+1}) G(\hat{E}^i_{t+1}). \qquad (21)
\end{aligned}
$$

[1] We assumed that the actions are relevant, otherwise they will belong to E'.

The above estimation is based on importance sampling just like in the non-abstract case (16), with similar target and proposal distributions. The main difference is the marginalization of irrelevant variables.

5.2.2 Importance weights for abstract episodes

Formula (21) is valid for any partition such that $G(E_t) = G(\hat{E}_t)$, but computing the weights $w(\hat{E}_{t+1})$ might be hard in general. To simplify the weight computation let us assume that the chosen partition guarantees the Markov property on abstract states, i.e., $p(\hat{s}_{t+1}|s_{0:t}, a_{0:t}) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$. To estimate $Q^{\pi}_d(s^m_t, a)$ (episode m), the weight for abstract episode $i < m$ becomes the following:

$$w(\hat{E}_{t+1}) = \frac{p(\hat{E}_{t+1}|s^m_t, a, \pi)}{q^i(\hat{E}_{t+1})} = \frac{\int_{E'_{t+1}} p(s_{t+1:T}, a_{t+1:T}|s^m_t, a, \pi) dE'_{t+1}}{\int_{E'_{t+1}} p(s_{t+1:T}, a_{t+1:T}|s_0, \pi^i) dE'_{t+1}}$$

$$= \frac{p(\hat{s}_{t+1:T}, a_{t+1:T}|s^m_t, a, \pi)}{p(\hat{s}_{t+1:T}, a_{t+1:T}|s_0, \pi^i)}$$

$$= \frac{p(\hat{s}_{t+1}|s^m_t, a)\pi(a_{t+1}|\hat{s}_{t+1}, s^m_t, a)}{q^i(\hat{s}_{t+1})\pi^i(a_{t+1}|\hat{s}_{t+1}, s_0)} \frac{\prod_{k=t+1}^{T-1} \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s^m_t, a) p(\hat{s}_{k+1}|\hat{s}_k, a_k)}{\prod_{k=t+1}^{T-1} \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) p(\hat{s}_{k+1}|\hat{s}_k, a_k)}$$

$$= \frac{p(\hat{s}_{t+1}|s^m_t, a)}{q^i(\hat{s}_{t+1})} \frac{\prod_{k=t}^{T-1} \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s^m_t, a)}{\prod_{k=t}^{T-1} \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0)} \tag{22}$$

$$\approx \frac{p(\hat{s}_{t+1}|s^m_t, a)}{q^i(\hat{s}_{t+1})} \frac{\prod_{k=t}^{T-1} \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0)}{\prod_{k=t}^{T-1} \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0)} \tag{23}$$

$$\approx \frac{p(\hat{s}_{t+1}|s^m_t, a)}{q^i(\hat{s}_{t+1})} \alpha^{(m-i)}. \tag{24}$$

Where $q^i(\hat{s}_{t+1}) = p(\hat{s}_{t+1}|s_0, \pi^i)$ and can be approximated with (19) by replacing s_{t+1} with $\hat{s}_{t+1}$. The final weight formula for abstracted states is similar to the non-abstract case. The difference is abstraction of the next state $\hat{s}_{t+1}$, while the state s^m_t in which the Q-function is estimated remains a complete state.

We will now explain the weight derivation and motivate the approximations adopted. Until formula (22) the only assumption made is the Markov property on abstract states. No assumptions are made about the action distributions (policies) π, π^i, thus the probability of an action a_t might depend on abstracted states in previous steps. Then (23) is replaced by (22) as discussed later. Finally, the policy ratio in (23) is replaced in (24) as in HYPE without abstraction.

Let us now discuss the approximation introduced in (23). Using (23) instead of (22) is equivalent to using the following target distribution:

$$\frac{p(\hat{s}_{t+1}|s^m_t, a)}{q^i(\hat{s}_{t+1})} p(\hat{s}_{t+1:T}, a_{t+1:T}|s_0, \pi) = p(\hat{s}_{t+1}|s^m_t, a) p(\hat{s}_{t+2:T}, a_{t+1:T}|\hat{s}_{t+1}, s_0, \pi),$$

instead of

$$p(\hat{s}_{t+1:T}, a_{t+1:T}|s_t^m, a, \pi) = p(\hat{s}_{t+1}|s_t^m, a)p(\hat{s}_{t+2:T}, a_{t+1:T}|\hat{s}_{t+1}, s_t^m, a, \pi). \quad (25)$$

Since the state transition model is the same in both distributions, the only difference is the marginalized action distribution (target policy). The one used in (23) is

$$\pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) \quad (26)$$

instead of $\pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_t^m, a)$ for $k = t, ..., T-1$. It is not straightforward to analyze this result because these actions distributions are obtained from the same policy π by applying a different marginalization. Nonetheless, it is worth mentioning that the marginalized target policy (26) does not depend on the specific state s_t^m, but only on abstract states and on the initial state s_0. This is arguably a desirable property for the (marginalized) target policy.

Using (26) as target policy, and thus (23) as weight, is useful when the proposal policies are equal to the target policy: $\forall i : \pi = \pi^i$. In this case the weight is exactly:

$$w(E_{t+1}^i) = \frac{p(\hat{s}_{t+1}^i|s_t^m, a)}{q^i(\hat{s}_{t+1}^i)}, \quad (27)$$

because the policy ratio cancels out. This formula is also applicable when $\forall i : \pi = \pi^i$ and $\pi(a|s_t) = \pi(a|\hat{s}_t)$ or at least $\pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_t^m, a) = \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) = \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k})$, that are indeed special cases of (26) and (23). Imposing or assuming $\pi(a|s_t) = \pi(a|\hat{s}_t)$ seems a reasonable choice, even though (26) is a weaker assumption. The optimal policy π^*, might depend only on abstract states, thus $\pi^*(a|s_t) = \pi^*(a|\hat{s}_t)$. Indeed, we expect that the optimal policy depends only on the relevant part of the state. However, we can neither assume $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ nor $\pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) = \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k})$ as proposal policy. This is because the proposal policy π^i used to sample episode i has to explore with a non-zero probability all the actions: abstract states are generally not sufficient to determine the admissible actions. Thus, the dependence on the initial state s_0 is inevitable. In conclusion, the marginal target policy (26) is one of the weakest assumptions to guarantee a weight (27) for $\forall i : \pi = \pi^i$. For $\pi \neq \pi^i$ the weight becomes (23).

Now let us focus on (24) derived from (23). Since the policies π^i used in the episodes are assumed to improve over time, we replaced the policy ratio in (23) with a quantity that favors recent episodes as in the propositional case (formula (18)). Another way of justifying (24) is estimating for each stored abstract episode i, the Q-function $Q_d^{\pi^i}(s_t^m, a)$, with target policy $\pi = \pi^i$, and using only the i-th sample. With a marginalized target policy given by (26), the single weight of each estimate $Q_d^{\pi^i}(s_t^m, a)$ is exactly (27). The used Q-function estimate can be a weighted average of $Q_d^{\pi^i}(s_t^m, a)$, where recent estimates (higher index i) receive higher weights because the policy is assumed to improve over time. Thus, the final weights are given by (24).

HYPE with abstraction adopts formula (21) and weights (24) for Q-function estimation. Note that during episode sampling the states are complete, nonetheless, to compute $Q_d^\pi(s_t^m, a)$ at episode m all previously abstracted episodes $i < m$ are considered. Finally, when the sampling of episode m is terminated, it is abstracted (line 11) and stored (line 14).

5.2.3 Ineffectiveness of lazy instantiation

Before explaining the proposed abstraction in detail, let us consider an alternative solution that samples abstract episodes directly, instead of sampling a complete episode and performing abstraction afterwards. If we are able to determine and sample partial states $\hat{s}_t^m$, we can sample abstract episodes directly and perform Q-function estimation. Sampling the relevant partial episode $\hat{E}_t$ can be easily performed using lazy instantiation, where given the query $G(E_t)$, only relevant random variables are sampled until the query can be answered. Lazy instantiation can exploit context-specific independencies and be extended for distributions with a countably infinite number of variables, as in BLOG [30, 29]. Similarly, Distributional Clauses search relevant random variables (or facts) using backward reasoning, while sampling is performed in a forward way. For example, to prove R_t the algorithm needs to sample the variables $\hat{s}_t$ relevant for R_t, $\hat{s}_t$ depends on $\hat{s}_{t-1}$ and the action a_{t-1} depends on the admissible actions that again depend on $\hat{s}_{t-1}$, and so on. At some point variables can be sampled because they depend on known facts (e.g., initial state s_0). This procedure guarantees that $G(E_t) = G(\hat{E}_t)$, $p(\hat{s}_{t+1}|s_{0:t}, a_t) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$ and $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$, thus (22) is exactly equal to (23) and it simplifies to $\frac{p(\hat{s}_{t+1}|s_t^m,a)}{q^i(\hat{s}_{t+1})} \frac{\prod_{k=t}^{T-1} \pi(a_{k+1}|\hat{s}_{k+1})}{\prod_{k=t}^{T-1} \pi^i(a_{k+1}|\hat{s}_{k+1})}$. Finally, the approximation (24) can be used. Unfortunately, this method avoids only sampling variables that are completely irrelevant, therefore in many practical domains it will sample (almost) the entire state. Indeed, evaluating the admissible actions often requires sampling the entire state. In other words, the abstract state $\hat{s}_t \subseteq s_t$ that guarantees $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ is often equal to s_t. The solution adopted in this paper is ignoring the requirement $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ and approximate (22) with (23), or equivalently using (26) as marginalized target policy distribution.

5.3 Sample-based abstraction by logical regression

In this section we describe how to implement the proposed sample-based abstraction. The implementation is based on Dynamic Distributional Clauses for two reasons: DDC allow to represent complex hybrid relational domains and provides backward reasoning procedures useful for abstraction as we will now describe.

Algorithm 2 samples complete episodes and performs abstraction afterwards. The abstraction of $\hat{E}_t$ from E_t (REGRESS function at line 11) is decomposed recursively employing backward reasoning (regression) from the

last step $t = T$ till reaching s_0. We first regress the query $R(s_T, a_T)$ using s_T to obtain the abstract state $\hat{s}_T = \hat{E}_T$ (computing the most general $\hat{s}_T$ such that $R(\hat{s}_T, a_T) = R(s_T, a_T)$). For $t = T-1, ..., 0$ we regress the query $R(s_t, a_t) \wedge \hat{s}_{t+1}$ using $a_t, s_t \in E_t$ to obtain the most general $\hat{s}_t \subseteq s_t$ that guarantees $R(\hat{s}_t, a_t) = R(s_t, a_t)$ and $p(\hat{s}_{t+1}|s_t, a_t) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$. Note that $\hat{E}_t = \hat{s}_t \cup \hat{E}_{t+1}$. This method assumes that the actions are given, thus it avoids determining the admissible actions, keeping the abstract states smaller. For this reason, REGRESS guarantees only $G(E_t) = G(\hat{E}_t)$ and $p(\hat{s}_{t+1}|\hat{s}_{0:t}, a_t) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$, in contrast $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ is not guaranteed. Those conditions are sufficient to apply (23) and (24) as described in the previous section. Note that derivation (21) assumes a fixed partition, thus exploits only conditional independencies, but the idea can be extended to context-specific independencies.

Algorithm 3 Episode abstraction

```
1: function REGRESS(Query,Facts)                          ▷ regress Query using Facts
2:     S ← ∅
3:     for L ∈ Query, L ≠ action(_) do
4:         Find θ = mgu(L, F) with F ∈ Facts
5:         if ∃(H ∼ D ← B), ∃β s.t. Facts ⪯ Bβ and F is ≃(Hβ) = v then    ▷ the clause could have generated F
6:             Query ← Queryθβ\F ∪ Bβ
7:         else                                           ▷ Lθ = F not regressable
8:             S ← S ∪ F
9:             Query ← Queryθ\F
10:        end if
11:    end for
12:    return S
13: end function
```

The algorithm REGRESS for regressing a query (formula) using a set of facts is depicted in Algorithm 3. The algorithm tries to repeatedly find literals in the query that could have been generated using the set of facts and a distributional clause. If it finds such a literal, it will be replaced by the condition part of the clause in the query. If not, it will add the fact to the state to be returned.

Example 4 To illustrate the algorithm, consider the blocksworld example in Fig. 2. Let us consider the abstraction of the episode on the left. To prove the last reward we need to prove the goal, thus $\hat{s}_2 = \texttt{on}(2,1)_2$. Now let us consider time step 1, the proof for the immediate reward is $\texttt{not}(\texttt{on}(2,1)_1)$, while the proof for the next abstract state $\hat{s}_2$ is $\texttt{on}(2,\texttt{table})_1, \texttt{clear}(1)_1, \texttt{clear}(2)_1$, therefore the abstract state becomes $\hat{s}_1 = \texttt{on}(2,\texttt{table})_1, \texttt{clear}(1)_1, \texttt{clear}(2)_1$, $\texttt{not}(\texttt{on}(2,1)_1)$. Analogously, $s'_0 = \texttt{on}(1,2)_0, \texttt{on}(2,\texttt{table})_0, \texttt{clear}(1)_0, \texttt{not}(\texttt{on}(2,1)_0)$. The same procedure is applicable to continuous variables.

6 Related work

6.1 Non-relational planners

There is an extensive literature on MDP planners, we will focus mainly on Monte Carlo approaches. The most notable sample-based planners include Sparse Sampling (SST) [17], UCT [22] and their variations. SST creates a lookahead tree of depth D, starting from state s_0. For each action in a given state, the algorithm samples C times the next state. This produces a near-optimal solution with theoretical guarantees. In addition, this algorithm works with continuous and discrete domains with no particular assumptions. Unfortunately, the number of samples grows exponentially with the depth D, therefore the algorithm is extremely slow in practice. Some improvements have been proposed [52], although the worst-case performance remains exponential. UCT [22] uses *upper confidence bound* for multi-armed bandits to trade off between exploration and exploitation in the tree search, and inspired successful Monte Carlo tree search methods [4]. Instead of building the full tree, UCT chooses the action a that maximizes an upper confidence bound of $Q(s, a)$, following the principle of optimism in the face of uncertainty. Several improvements and extensions for UCT have been proposed, including handling continuous actions [26] (see [31] for a review), and continuous states [5] with a simple Gaussian distance metric; however the knowledge of the probabilistic model is not directly exploited. For continuous states, parametric function approximation is often used (e.g., linear regression), nonetheless the model needs to be carefully tailored for the domain to solve [54].

There exist algorithms that exploit instance-based methods (e.g. [9,45,6]) for model-free reinforcement learning. They basically store Q-point estimates, and then use e.g., neighborhood regression to evaluate $Q(s, a)$ given a new point (s, a). While these approaches are effective in some domains, they require the user to design a distance metric that takes into account the domain. This is straightforward in some cases (e.g., in Euclidean spaces), but it can be harder in others. We argue that the knowledge of the model can avoid (or simplify) the design of a distance metric in several cases, where the importance sampling weights and the transition model, can be considered as a kernel.

The closest related works include [43,44,38,39], they use importance sampling to evaluate a policy from samples generated with another policy. Nonetheless, they adopt importance sampling differently without knowledge of the MDP model. Although this property seems desirable, the availability of the actual probabilities cannot be exploited, apart from sampling, in their approaches. The same conclusion is valid for practically any sample-based planner, which only needs a sample generator of the model. The work of [18] made a similar statement regarding PROST, a state-of-the-art discrete planner based on UCT, without providing a way to use the state transition probabilities directly. Our algorithm tries to alleviate this, exploiting the probabilistic model in a sample-based planner via importance sampling.

For more general domains that contain discrete and continuous (hybrid) variables several approaches have been proposed under strict assumptions. For example, [41] provide exact solutions, but assume that continuous aspects of the transition model are deterministic. In a related effort [8], hybrid MDPs are solved using dynamic programming, but assuming that transition model and reward is piecewise constant or linear. Another planner HAO* [28] uses heuristic search to find an optimal plan in hybrid domains with theoretical guarantees. However, they assume that the Bellman equation integral can be computed.

For domains with an unknown number of objects, some probabilistic programming languages such as BLOG [29], Church [11], Anglican [55], and DC [12] can cope with such uncertainty. To the best of our knowledge DTBLOG [46] and [51] are the only proposals that are able to perform decision making in such domains using a POMDP framework. Furthermore, BLOG is one of the few languages that explicitly handles data association and identity uncertainty. The current paper does not focus on POMDP, nor on identity uncertainty; however, interesting domains with unknown number of objects can be easily described as an MDP that HYPE can solve.

Among the mentioned sample-based planners, one of the most general is SST, which does not make any assumption on the state and action space, and only relies on Monte Carlo approximation. In addition, it is one of the few planners that can be easily applied to any DDC program, including MDPs with an unknown number of objects. For this reason SST was implemented for DDC and used as baseline for our experiments.

6.2 Relational planners and abstraction

There exists several modeling languages for planning, the most recent is RDDL [40] that supports hybrid relational domains. A RDDL domain can be mapped in DDC and solved with HYPE. Nonetheless, RDDL does not support a state space with an unknown number of variables as in Example 2.

Relational MDPs can be solved using model-free approaches based on Relational Reinforcement Learning [7,48,6], or model-based methods such as ReBel [19], FODD [53], PRADA [23], FLUCAP [13] and many others. However, those approaches only support discrete action-state (relational) spaces.

Among model-based approaches, several symbolic methods have been proposed to solve MDPs exactly in propositional (see [27] for a review) and relational domains [19,53,16,13]. They perform Dynamic Programming at the level of abstract states; this approach is generally called Symbolic Dynamic Programming (SDP). Similar principles have been applied in (propositional) continuous and hybrid domains [41,57], where compact structures (e.g., ADD and XADD) are used to represent the V-function. Despite the effectiveness of such approaches, they make restrictive assumptions (e.g., deterministic transition model for continuous variables) to keep exact inference tractable. For more general domains approximations are needed, for example sample-based meth-

ods or confidence intervals [56]. Another issue of SDP is keeping the structures that represent the V-function compact. Some solutions are available in the literature, such as pruning or real-time SDP [50]. Despite the recent progress, and the availability of regression methods for inference in hybrid domains [3], SDP remains a challenging approach in general hybrid relational domains.

Recently, abstraction has received a lot of attention in the Monte Carlo planning literature. Like in our work, the aim is to simplify the planning task by aggregating together states that behave similarly. There are several ways to define state equivalence, see [24] for a review. Some approaches adopt model equivalence: states are equivalent if they have the same reward and the probabilities to end up in other abstract states are the same. Other approaches define the equivalence in terms of the V/Q-function. In particular, we take note of the following advances: (a) Givan et al [10] who compute equivalence classes of states based on exact model equivalence, (b) Jiang et al. [15] who appeal to approximate local homomorphisms derived from a learned model, (c) Anand et al. [1] who extend Jiang and Givan in grouping state-action pairs, and (d) Hostetler et al [14] who aggregate states considering the V/Q-function with tight loss bounds.

In our work, in contrast, we consider equivalence (abstraction) at the level of episodes, not states. Two episodes are equivalent if they have the same total reward. In addition, a Markov property condition on abstract states is added to make the weights in (21) easier to compute. Abstraction is performed independently in each episode, determining, by logical regression, the set of facts (or random variables) sufficient to guarantee the mentioned conditions. Note that the same full state s_t might have different abstractions in different episodes, even for the same action a_t. This is generally not the case in other works. The proposed abstraction directly exploits the structure of the model (independence assumptions) to perform abstraction. For this reason it relies on the (context-specific) independence assumptions explicitly encoded in the model. However, it is possible to discover independence assumptions not explicitly encoded and include them in the model (e.g., using independence tests).

7 Experiments

This section answers the following questions:
(Q1) Does HYPE without abstraction obtain the correct results?
(Q2) How is the performance of HYPE in different domains?
(Q3) How does HYPE compare with state-of-the-art planners?
(Q4) Is abstraction beneficial?

The domains used in the experiments are written in DDC; the algorithms were implemented in YAP Prolog and C++, and run on a Intel Core i7 Desktop. We will first describe experiments without abstraction, then compare HYPE with and without abstraction.

7.1 HYPE without abstraction

In this section we consider HYPE without abstraction. The algorithm HYPE and its theoretical foundations are based on approximations (e.g., Monte Carlo). For this reason we tested the correctness of HYPE results in different planning domains (Q1). In particular, we tested HYPE on a nonlinear version of the hybrid mars rover domain [41] (called *simplerover1*) for which the exact V-function is available. In this domain there is a rover that needs to take pictures. The state consists of a two-dimensional continuous rover position (x, y) and one discrete variable h to indicate whether the picture at a target point was taken. In this simplified domain we consider two actions: *move* with reward -1 that moves the rover towards the target point $(0, 0)$ by $1/3$ of the current distance, and *take-pic* that takes the picture at the current location. The reward of *take-pic* is $max(0, 4 - x^2 - y^2)$ if the picture has not been already taken ($h = false$) and 0 otherwise. In other words, the agent has to minimize the movement cost and take a picture as close as possible to the target point $(0, 0)$. We choose 31 initial rover positions and ran the algorithm with depth $d = 3$ for 100 episodes each. An experiment took on average $1.4s$. Fig. 3 shows the results where the line is the exact V provided by [41], and dots are estimated V points. The results show that the algorithm converges to the optimal V-function with a negligible error.

The domain *simplerover1* is deterministic, and so, to make it more realistic we converted it to a probabilistic MDP adding Gaussian noise to the state transition model (with a variance $\sigma^2 = 0.0005$ when the rover does not move and $\sigma^2 = 0.02$ when the rover moves). The resulting MDP (*simplerover2*) is hard (if not impossible) to solve exactly. Then we performed experiments for different horizons, number of pictures points (1 to 4, each one is a discrete variable) and summed the rewards. For each instance the planner searches for an optimal policy and executes it, and after each executed action it samples additional episodes to refine the policy (replanning). The proposed planner is compared with SST which must replan every step. The results for both planners are always comparable, which confirms the empirical correctness of HYPE (Table 1).

To answer (Q2) and (Q3) we studied the planner in a variety of settings, from discrete, to continuous, to hybrid domains, to those with an unknown number of objects. We performed experiments in a more realistic mars rover domain that is publicly available[2], called *marsrover* (Fig. 3). In this domain we consider one robot and 5 picture points that need to be taken. The state is similar to *simplerover* domains: a continuous two-dimensional position and a binary variable for each picture point. However, in *marsrover* the robot can move an arbitrary displacement along the two dimensions. The continuous action space is discretized as required by HYPE and SST. The movement of the robot causes a negative reward proportional to the displacement and

[2] `http://users.cecs.anu.edu.au/~ssanner/IPPC_2014/index.html`

Table 1 Experiments without abstraction: d is the horizon used by the planner, T the total number of steps, M is the maximum number of episodes sampled for HYPE, while C is the SST parameter (number of samples for each state and action). Time refers to the plan execution of one instance, from the starting state till the goal or the maximum number of steps is reached, with a timeout of 1800s. PROST results refer to the IPPC2011 planning competition.

Domain	Planner	d	T	Param.	Reward	Time	Size
game1	HYPE	5	40	M = 1200	0.87 ± 0.11	662	9 discrete variables 10 actions
	SST	5	40	C = 1	0.34 ± 0.15	986	
	HYPE	4	40	M = 1200	**0.89 ± 0.07**	312	
	SST	4	40	C = 2	0.79 ± 0.08	1538	
	PROST				0.99 ± 0.02		
game2	HYPE	5	40	M = 1200	0.67 ± 0.18	836	
	SST	5	40	C = 1	0.14 ± 0.20	1000	
	HYPE	4	40	M = 1200	**0.76 ± 0.19**	582	
	SST	4	40	C = 2	0.27 ± 0.22	1528	
	PROST				1.00 ± 0.19		
sysadmin1	HYPE	5	40	M = 1200	0.94 ± 0.07	422	10 discrete variables 11 actions
	SST	5	40	C = 1	0.47 ± 0.13	1068	
	HYPE	4	40	M = 1200	**0.98 ± 0.06**	346	
	SST	4	40	C = 2	0.66 ± 0.08	1527	
	PROST				1.00 ± 0.05		
sysadmin2	HYPE	5	40	M = 1200	**0.87 ± 0.11**	475	
	SST	5	40	C = 1	0.31 ± 0.12	1062	
	HYPE	4	40	M = 1200	0.86 ± 0.11	392	
	SST	4	40	C = 2	0.46 ± 0.12	1532	
	PROST				0.98 ± 0.09		

Domain	Planner	d	T	Param.	Reward	Time	Size
objpush	HYPE	9	30	M = 4500	83.7 ± 7.6	472	2 continuous variables 4 actions
	SST	9	30	C = 1	82.7 ± 2.7	330	
	HYPE	10	30	M = 4500	86.4 ± 1.0	1238	
	SST	10	30	C = 1	82.4 ± 1.9	1574	
	HYPE	12	30	M = 2000	**87.5 ± 0.5**	373	
	SST	≥ 11	30	C = 1	N/A	timeout	
simplerover2	HYPE	8	8	M = 200	11.8 ± 0.2	38	2 continuous variables 1, 2, 3, 4 discrete variables 2, 3, 4, 5 actions
	SST	8	8	C = 1	11.4 ± 0.3	48	
	HYPE	9	9	M = 500	11.7 ± 0.2	195	
	SST	9	9	C = 1	11.3 ± 0.3	238	
	HYPE	10	10	M = 500	**11.9 ± 0.3**	218	
	SST	10	10	C = 1	11.2 ± 0.3	1043	
marsrover	HYPE	6	40	M = 6000	249.8 ± 33.5	985	2 continuous variables 5 discrete variables 10 actions
	SST	6	40	C = 1	227.7 ± 27.3	787	
	HYPE	7	40	M = 6000	269.0 ± 29.4	983	
	SST	7	40	C = 1	N/A	timeout	
	HYPE	10	40	M = 4000	**296.3 ± 19.5**	1499	
	SST	≥ 8	40	C = 1	N/A	timeout	
objsearch	HYPE	5	5	M = 500	2.53 ± 1.03	13	variable size
	SST	5	5	C = 5	1.46 ± 1.00	45	
	HYPE	5	5	M = 600	**3.64 ± 1.09**	17	
	SST	5	5	C = 6	2.48 ± 1.00	138	
	HYPE	6	6	M = 600	3.30 ± 1.60	20	
	SST	6	6	C = 5	0.58 ± 1.40	889	

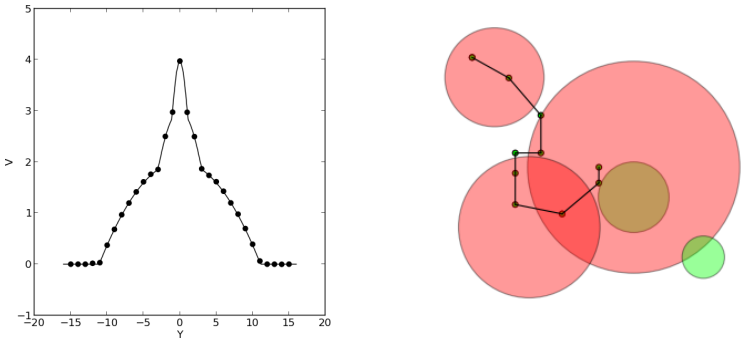


Fig. 3 *V*-function for different rover positions (with fixed $X = 0.16$) in *simplerover1* domain (left). A possible episode in *marsrover* (right): each picture can be taken inside the respective circle (red if already taken, green otherwise).

the pictures can be taken only close to the interest point. Each taken picture provides a different reward.

Other experiments were performed in the continuous *objpush* MDP described in Section 4 (Fig. 1), and in discrete benchmark domains of the IPPC 2011 competition. In particular, we tested a pair of instances of Game of life (called *game* in the experiments) and *sysadmin* domains. Game of life consists of a grid X times Y cells, where each cell can be dead or alive. The state of each cell changes over time and depends on the neighboring cells. In addition, the agent can set a cell to alive or do nothing. The reward depends on the number of cells alive. We consider instances with 3x3 cells, and thus a state of 9 binary variables and 10 actions. The *sysadmin* domain consists of a network of computers. Each computer might be running or crashed. The probability of a computer to crash depends on how many computers it is connected to. The agent can choose at each step to reboot a computer or do nothing. The goal is to maximize the number of computers running and minimize the number of reboots required. We consider instances with 10 computers (i.e. 10 binary random variables) and so there are 11 actions. The results in the discrete IPPC 2011 domains are compared with PROST [18], the IPPC 2011 winner, and shown in table 1 in terms of scores, i.e., the average reward normalizated with respect to IPPC 2011 results; score 1 is the highest result obtained (on average), score 0 is the maximum between the random and the no operation policy.

As suggested by [18], limiting the horizon of the planner increases the performance in several cases. We exploited this idea for HYPE as well as SST (*simplerover2* excluded). For SST we were forced to use small horizons to keep plan time under 30 minutes. In all experiments we followed the IPPC 2011 schema, that is each instance is repeated 30 times (*objectsearch* excluded), the results are averaged and the 95% confidence interval is computed. However, for every instance we replan from scratch for a fair comparison with SST. In addition, time and number of samples refers to the plan execution of one instance.

The results (Table 1) highlight that our planner obtains generally better results than SST, especially at higher horizons. HYPE obtains good results in discrete domains but does not reach state-of-art results (score 1) for two main reasons. The first is the lack of a heuristic, that can dramatically improve the performance, indeed, heuristics are an important component of PROST [18], the IPPC winning planner. The second reason is the time performance that allows us to sample a limited number of episodes and will not allow all the IPPC 2011 domains to finish in 24 hours. This is caused by a non-optimized Prolog implementation and by the expensive Q-function evaluation; however, we are confident that heuristics and other improvements will significantly improve performance and results. In particular, the weight computation can be performed on a subset of stored V-state points, excluding points for which the weight is known to be small or zero. For example, in the *objpush* domain, the points too far from the current position plus action displacement can be discarded because the weight will be negligible. Thus, a nearest neighbor search can be used for a fast retrieval of relevant stored V-points.

Moreover, we performed experiments in the *objectsearch* scenario (Section 3), where the number of objects is unknown, even though the domain is modeled as a fully observable MDP. The results are averaged over 400 runs, and confirm better performance for HYPE with respect to SST.

7.2 HYPE with abstraction

To evaluate the effectiveness of abstraction (Q4) we performed experiments with the blocksworld (BW with 4 or 6 objects) and a continuous version of it (BWC with 4 or 6 objects) with an energy level of the agent and object weights. The energy decreases with a quantity proportional to the weight of the object moved plus Gaussian noise. If the energy becomes 0 the action fails, otherwise the probability of success is 0.9. The reward is -1 before reaching the goal and $10 + Energy$ if the goal is reached. Then we performed experiments with the *objpush* scenario. This time we consider multiple objects on the table. We considered different goals: move an arbitrary object in the goal region (domain $push1$ with 2 objects), and move a specific object in the goal region ($push2$ and $push3$ with 3 objects). Finally, we performed experiments with the *marsrover* domain, with one robot (mars1) or two of them ($mars2$), and 5 picture points that need to be taken.

The current implementation supports negation only for ground formulas. Regression of nonground formulas is possible when the domain is purely relational. However, it becomes challenging when there are continuous random variables and logical variables in a negated formula. If we assume that the domain is fixed (e.g., known number of objects used), logical variables can be replaced with objects in the domain, making the formulas ground. For this reason we will not consider domains with an unknown number of objects, which HYPE without abstraction can solve.

Table 2 Experiments with and without abstraction. N is the number of sampled episodes, d is the horizon used by the planner, T is the maximum number of steps, 'success' is the number of times the goal is reached.

Domain	Abstract	d	T	M	Reward	Success	Time	Size
BW 4	NO	10	10	200	74.6 ± 7.4	82%	38	4 objects
	YES	10	10	200	**80.3 ± 7.2**	**88%**	32	
BW 6	NO	16	16	200	-16.0 ± 0.0	0%	112	6 objects
	YES	16	16	200	**54.8 ± 13.4**	**68%**	42	
BWC 4	NO	10	10	200	11.8 ± 2.7	84%	52	4 objects 1 cont. variables
	YES	10	10	200	**14.2 ± 1.9**	**94%**	24	
BWC 6	NO	18	18	200	-18.0 ± 0.0	0%	186	6 objects 1 cont. variables
	YES	18	18	200	**8.4 ± 2.4**	**94%**	70	
push1	NO	20	30	1000	67.6 ± 11	86%	734	4 cont. variables
	YES	20	30	1000	**84.0 ± 4.7**	**98%**	652	
push2	NO	20	30	1000	-30.0 ± 0.0	0%	1963	6 cont. variables
	YES	20	30	1000	**30.5 ± 14.0**	**58%**	910	
push3	NO	20	40	500	-17.4 ± 12.6	20%	638	6 cont. variables
	YES	20	40	500	**89.3 ± 1.5**	**100%**	122	
mars1	NO	30	40	1500	**280.0 ± 11.8**	**90%**	1492	2 cont. variables 5 discrete variables
	YES	30	40	1500	273.3 ± 11.0	86%	780	
mars2	NO	30	40	1000	209.3 ± 27.7	37%	2817	4 cont. variables 5 discrete variables
	YES	30	40	1000	**287.7 ± 23.6**	**87%**	902	

The experiments are shown in Table 2. The rewards are averaged over 50 runs and a 95% confidence interval is computed. The results highlight that abstraction improves the expected total reward for the same number of samples or achieves comparable results. In addition, HYPE with abstraction is always faster. The latter is probably due to a faster weight computation with abstract states and due to the generation of better plans that are generally shorter and thus faster. This suggests that the overhead caused by the abstraction procedure is negligible and worthwhile. We do remark that in domains where the whole state is always *relevant*, abstraction gives no added value. For example, the reward in Game of life and sysadmin depends always on the full state, thus abstraction is not useful because abstract state and full state coincide. Nonetheless, other type of abstractions can be beneficial. Indeed, the proposed abstraction (Alg. 3) produces grounded abstract states (i.e., states where the facts do not have logical variables), this is required to allow abstraction in complex domains. In more restricted domains (e.g., discrete) more effective abstractions can improve the performance. For example, abstractions used in SDP [19,53,16,13] produce abstract states with logical variables.

8 Conclusions

We proposed a sample-based planner for MDPs described in DDC under weak assumptions, and showed how the state transition model can be exploited in off-policy Monte Carlo. The experimental results show that the algorithm produces good results in discrete, continuous, hybrid domains as well as those

with an unknown number of objects. Most significantly, it challenges and outperforms SST. Moreover, we extended HYPE with abstraction. We formally described how (context-specific) independence assumptions can be exploited to perform episode abstraction. This is valid for propositional as well as relational domains. A theoretical derivation has been provided to justify the assumptions and the approximations used. Finally, empirical results showed that abstraction provides significant improvements.

References

1. Anand, A., Grover, A., Mausam, Singla, P.: ASAP-UCT: abstraction of state-action pairs in UCT. In: Proceedings of IJCAI, pp. 1509–1515 (2015)
2. Apt, K.: From logic programming to Prolog. Prentice-Hall international series in computer science. Prentice Hall (1997)
3. Belle, V., Levesque, H.J.: PREGO: an action language for belief-based cognitive robotics in continuous domains. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada., pp. 989–995 (2014)
4. Browne, C., Powley, E.J., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Trans. Comput. Intellig. and AI in Games **4**(1), 1–43 (2012). URL `http://dblp.uni-trier.de/db/journals/tciaig/tciaig4.html`
5. Couetoux, A.: Monte Carlo Tree Search for Continuous and Stochastic Sequential Decision Making Problems. Theses, Université Paris Sud - Paris XI (2013)
6. Driessens, K., Ramon, J.: Relational instance based regression for relational reinforcement learning. In: Proc. ICML, pp. 123–130 (2003)
7. Džeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. Machine learning **43**(1-2), 7–52 (2001)
8. Feng, Z., Dearden, R., Meuleau, N., Washington, R.: Dynamic programming for structured continuous Markov decision problems. In: Proc. UAI, pp. 154–161 (2004)
9. Forbes, J., Andre, D.: Representations for learning control policies. In: Proc. of the ICML Workshop on Development of Representations, pp. 7–14 (2002)
10. Givan, R., Dean, T., Greig, M.: Equivalence notions and model minimization in markov decision processes. Artificial Intelligence **147**(12), 163 – 223 (2003)
11. Goodman, N., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: A language for generative models. In: Proc. UAI, pp. 220–229 (2008)
12. Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., De Raedt, L.: The magic of logical inference in probabilistic programming. Theory and Practice of Logic Programming (2011)
13. Hölldobler, S., Karabaev, E., Skvortsova, O.: Flucap: A heuristic search planner for first-order MDPs. Journal of Artificial Intelligence Research **27**, 419–439 (2006)
14. Hostetler, J., Fern, A., Dietterich, T.: State aggregation in monte carlo tree search. In: Proceedings of AAAI (2014)
15. Jiang, N., Singh, S., Lewis, R.: Improving uct planning via approximate homomorphisms. In: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, pp. 1289–1296. International Foundation for Autonomous Agents and Multiagent Systems (2014)
16. Joshi, S., Kersting, K., Khardon, R.: Self-Taught Decision Theoretic Planning with First Order Decision Diagrams. In: ICAPS, pp. 89–96 (2010)
17. Kearns, M., Mansour, Y., Ng, A.Y.: A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. Machine Learning **49**(2-3), 193–208 (2002)
18. Keller, T., Eyerich, P.: PROST: Probabilistic Planning Based on UCT. In: Proc. ICAPS (2012)

19. Kersting, K., Otterlo, M.V., De Raedt, L.: Bellman goes relational. In: Proc. ICML, p. 59 (2004)
20. Kimmig, A., Demoen, B., De Raedt, L., Costa, V.S., Rocha, R.: On the Implementation of the Probabilistic Logic Programming Language ProbLog. CoRR (2010)
21. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the Efficient Execution of ProbLog Programs. In: Logic Programming, Lecture Notes in Computer Science, pp. 175–189. Springer Berlin / Heidelberg (2008)
22. Kocsis, L., Szepesvári, C.: Bandit Based Monte-carlo Planning. In: Proc. ECML (2006)
23. Lang, T., Toussaint, M.: Planning with Noisy Probabilistic Relational Rules. Journal of Artificial Intelligence Research **39**, 1–49 (2010)
24. Li, L., Walsh, T.J., Littman, M.L.: Towards a unified theory of state abstraction for mdps. In: ISAIM (2006)
25. Lloyd, J.: Foundations of logic programming. Springer-Verlag New York, Inc. (1987)
26. Mansley, C.R., Weinstein, A., Littman, M.L.: Sample-Based Planning for Continuous Action Markov Decision Processes. In: Proc. ICAPS (2011)
27. Mausam, Kolobov, A.: Planning with Markov Decision Processes: An AI Perspective. Morgan & Claypool Publishers (2012)
28. Meuleau, N., Benazera, E., Brafman, R.I., Hansen, E.A., Mausam, M.: A heuristic search approach to planning with continuous resources in stochastic domains. Journal of Artificial Intelligence Research **34**(1), 27 (2009)
29. Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., Kolobov, A.: BLOG: Probabilistic Models with Unknown Objects. In: Proc. IJCAI (2005)
30. Milch, B., Marthi, B., Sontag, D., Russell, S., Ong, D.L., Kolobov, A.: Approximate Inference for Infinite Contingent Bayesian Networks. In: Proc. of the 10th International Workshop on Artificial Intelligence and Statistics (2005)
31. Munos, R.: From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. Foundations and Trends(r) in Machine Learning. Now Publishers (2014)
32. Nilsson, U., Małiszyński, J.: Logic, Programming And Prolog, 2nd edn. Wiley & Sons (1996)
33. Nitti, D., Belle, V., De Laet, T., De Raedt, L.: Sample-based abstraction for hybrid relational MDPs (2015). European Workshop on Reinforcement Learning (EWRL 2015), 10 11 July 2015
34. Nitti, D., Belle, V., De Raedt, L.: Planning in discrete and continuous markov decision processes by probabilistic programming. Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD) 2015 (2015)
35. Nitti, D., De Laet, T., De Raedt, L.: A Particle Filter for Hybrid Relational Domains. Proc. IROS (2013)
36. Nitti, D., De Laet, T., De Raedt, L.: Relational object tracking and learning. In: Proc. ICRA (2014)
37. Owen, A.B.: Monte Carlo theory, methods and examples (2013)
38. Peshkin, L., Shelton, C.R.: Learning from Scarce Experience. In: Proc. ICML, pp. 498–505 (2002)
39. Precup, D., Sutton, R.S., Singh, S.P.: Eligibility Traces for Off-Policy Policy Evaluation. In: Proc. ICML (2000)
40. Sanner, S.: Relational Dynamic Influence Diagram Language (RDDL): Language Description. Unpublished paper.
41. Sanner, S., Delgado, K.V., de Barros, L.N.: Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In: Proc. UAI, pp. 643–652 (2011)
42. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the Twelth International Conference on Logic Programming, pp. 715–729. MIT Press (1995)
43. Shelton, C.R.: Importance Sampling for Reinforcement Learning with Multiple Objectives. Ph.D. thesis, MIT (2001)
44. Shelton, C.R.: Policy Improvement for POMDPs Using Normalized Importance Sampling. In: Proc. UAI, pp. 496–503 (2001)
45. Smart, W.D., Kaelbling, L.P.: Practical Reinforcement Learning in Continuous Spaces. In: Proc. ICML (2000)

46. Srivastava, S., Russell, S., Ruan, P., Cheng, X.: First-order open-universe POMDPs. Proc. UAI (2014)
47. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
48. Tadepalli, P., Givan, R., Driessens, K.: Relational reinforcement learning: An overview. In: Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning, pp. 1–9 (2004)
49. Van den Broeck, G., Thon, I., van Otterlo, M., De Raedt, L.: DTProbLog: A decision-theoretic probabilistic Prolog. In: Proc. AAAI, pp. 1217–1222 (2010)
50. Vianna, L.G.R., de Barros, L.N., Sanner, S.: Real-time symbolic dynamic programming. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA., pp. 3402–3408 (2015)
51. Vien, N.A., Toussaint, M.: Model-Based Relational RL When Object Existence is Partially Observable. In: Proc. ICML (2014)
52. Walsh, T.J., Goschin, S., Littman, M.L.: Integrating Sample-Based Planning and Model-Based Reinforcement Learning. In: proc. AAAI (2010)
53. Wang, C., Joshi, S., Khardon, R.: First Order Decision Diagrams for Relational MDPs. Journal of Artificial Intelligence Research (JAIR) **31**, 431–472 (2008)
54. Wiering, M., van Otterlo, M.: Reinforcement Learning: State-of-the-Art. Adaptation, Learning, and Optimization. Springer (2012)
55. Wood, F., van de Meent, J.W., Mansinghka, V.: A new approach to probabilistic programming inference. In: Proceedings of the 17th International conference on Artificial Intelligence and Statistics, pp. 1024–1032 (2014)
56. Zamani, Z., Sanner, S., Delgado, K.V., de Barros, L.N.: Robust optimization for hybrid mdps with state-dependent noise. In: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013 (2013)
57. Zamani, Z., Sanner, S., Fang, C.: Symbolic Dynamic Programming for Continuous State and Action MDPs. In: Proc. AAAI (2012)