Active Exploration for Learning Symbolic Representations

Garrett Andersen
PROWLER.io
Cambridge, United Kingdom
garrett@prowler.io

George Konidaris
Department of Computer Science
Brown University
gdk@cs.brown.edu

Abstract

We introduce an online active exploration algorithm for data-efficiently learning an abstract symbolic model of an environment. Our algorithm is divided into two parts: the first part quickly generates an intermediate Bayesian symbolic model from the data that the agent has collected so far, which the agent can then use along with the second part to guide its future exploration towards regions of the state space that the model is uncertain about. We show that our algorithm outperforms random and greedy exploration policies on two different computer game domains. The first domain is an Asteroids-inspired game with complex dynamics, but basic logical structure. The second is the Treasure Game, with simpler dynamics, but more complex logical structure.

1 Introduction

Much work has been done in artificial intelligence and robotics on how high-level state abstractions can be used to significantly improve planning [21]. However, building these abstractions is difficult, and consequently, they are typically hand-crafted [4, 5, 6, 8, 10, 14, 16, 22].

A major open question is then the problem of abstraction: how can an intelligent agent learn high-level models that can be used to improve decision making, using only noisy observations from its high-dimensional sensing and actuation spaces? Recent work [12, 13] has shown how to automatically generate symbolic representations suitable for planning in high-dimensional, continuous domains. This work is based on the hierarchical reinforcement learning framework [1], where the agent has access to high-level skills which abstract away the low-level details of control. The agent then learns representations for the (potentially abstract) effect of using these skills. For instance, opening a door is a high-level skill, while knowing that opening a door typically allows one to enter a building would be part of the representation for this skill. The key result of this work was that the symbols required to determine the probability of a plan succeeding are directly determined by characteristics of the skills available to an agent. The agent can learn these symbols autonomously by exploring the environment to collect data, which removes the need to hand-design symbolic representations of the world.

It is therefore possible to learn the symbols by naively collecting samples from the environment, for example by random exploration. However, in an online setting, the agent can use its previously collected data to compute an exploration policy which leads to better data efficiency. We introduce such an algorithm, which is divided into two parts: the first part quickly generates an intermediate Bayesian symbolic model from the data that the agent has collected so far, while the second part uses the model plus Monte-Carlo tree search to guide the agent's future exploration towards regions of the state space that the model is uncertain about. We show that our algorithm is significantly more data-efficient versus more naive methods in two different computer game domains. The first domain is an Asteroids-inspired game with complex dynamics, but basic logical structure. The second is the Treasure Game, with simpler dynamics, but more complex logical structure.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2 Background

As a motivating example, imagine deciding the route you are going to take to the grocery store; instead of planning over the various sequences of muscle contractions that could be used to complete the trip, you would consider a small number of high-level alternatives such as whether to take one route or another. You also would avoid considering how your exact low-level state affected your decision making, and instead use an abstract (symbolic) representation of your state with components such as whether you are at home or an work, whether you have to get gas, whether there is traffic, etc. This simplification is helpful because it reduces computational complexity, and allows for increased generalization over past experiences.

In the following sections, we introduce the frameworks that we use for the agent's high-level skills, and symbolic models for those skills.

2.1 Semi-Markov Decision Processes

We assume that the agent's environment can be described by a semi-Markov decision process (SMDP), which is given by a tuple $D = (S, O, R, P, \gamma)$, where $S \subseteq \mathbb{R}^d$ is a d-dimensional continuous state space, $O(s)$ returns a set of temporally extended actions, or *options* [21] available in state $s \in S$, $R(s', t, s, o)$ and $P(s', t \mid s, o)$ are the reward received and probability density of terminating in state $s' \in S$ after t time steps following the execution of option $o \in O(s)$ in state $s \in S$, and $\gamma \in (0, 1]$ is a discount factor. In this paper, we are not concerned with the time taken to execute o, so we use $P(s' \mid s, o) = \int P(s', t \mid s, o)\mathrm{d}t$.

An option o is given by three components: π_o, the *option policy* that is executed when the option is invoked, I_o, the *initiation set* consisting of the states where the option can be executed from, and $\beta_o(s) \to [0, 1]$, the *termination condition*, which returns the probability that the option will terminate upon reaching state s. Learning models for the initiation set, rewards, and transitions for each option, allows the agent to reason about the effect of its actions in the environment. To learn these *option models*, the agent has the ability to collect observations of the forms $(s, O(s))$ when entering a state s and (s, o, s', r, t) upon executing option o from s.

2.2 Abstract Representations for Planning

We are specifically interested in learning option models which allow the agent to easily evaluate the success probability of plans. A *plan* is a sequence of options to be executed from some starting state, and it *succeeds* if and only if it is able to be run to completion (regardless of the reward). Thus, a plan $\{o_1, o_2, ..., o_n\}$ with starting state s succeeds if and only if $s \in I_{o_1}$ and the termination state of each option (except for the last) lies in the initiation set of the following option, i.e. $s' \sim P(s' \mid s, o_1) \in I_{o_2}$, $s'' \sim P(s'' \mid s', o_2) \in I_{o_3}$, and so on.

Recent work [12, 13] has shown how to automatically generate a symbolic representation which is suitable for planning. This work is based on the idea of a *probabilistic symbol*, a compact representation of a distribution over infinitely many continuous, low-level states. For example, a probabilistic symbol could be used to classify whether or not the agent is currently in front of a door, or one could be used to represent the state that the agent would find itself in after executing its 'open the door' option. In both cases, using probabilistic symbols also allows the agent to be uncertain about its state.

The following two probabilistic symbols are provably sufficient for evaluating the success probability of any plan [13]; *the probabilistic precondition*: $\mathrm{Pre}(o) = P(s \in I_o)$, which expresses the probability that an option o can be executed from each state $s \in S$, and *the probabilistic image operator*:

$$\mathrm{Im}(o, Z) = \frac{\int_S P(s' \mid s, o) Z(s) P(I_o \mid s) \mathrm{d}s}{\int_S Z(s) P(I_o \mid s) \mathrm{d}s},$$

which represents the distribution over termination states if an option o is executed from a distribution over starting states Z. These symbols can be used to compute the probability that each successive option in the plan can be executed, and these probabilities can then be multiplied to compute the overall success probability of the plan; see Figure 1 for a visual demonstration of a plan of length 2.

Subgoal Options Unfortunately, it is difficult to model $\mathrm{Im}(o, Z)$ for arbitrary options, so we focus on restricted types of options. A *subgoal option* [19] is a special class of option where the distribution

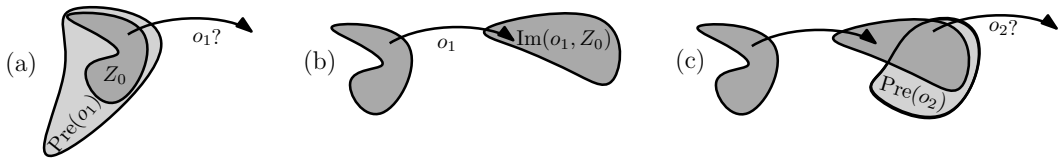


Figure 1: Determining the probability that a plan consisting of two options can be executed from a starting distribution Z_0. (a): Z_0 is contained in $\Pr(o_1)$, so o_1 can definitely be executed. (b): Executing o_1 from Z_0 leads to distribution over states $\mathrm{Im}(o_1, Z_0)$. (c): $\mathrm{Im}(o_1, Z_0)$ is not completely contained in $\Pr(o_2)$, so the probability of being able to execute o_2 is less than 1. Note that $\Pr$ is a set and Im is a distribution, and the agent typically has uncertain models for them.

over termination states (referred to as the subgoal) is independent of the distribution over starting states that it was executed from. Examples include walking to your kitchen or driving to work.

For subgoal options, the image operator can be replaced with the *effects distribution*: $\mathrm{Eff}(o) = \mathrm{Im}(o, Z), \forall Z(S)$, the resulting distribution over states after executing o from any start distribution $Z(S)$. Planning with a set of subgoal options is simple because for each ordered pair of options o_i and o_j, it is possible to compute and store $G(o_i, o_j)$, the probability that o_j can be executed immediately after executing o_i: $G(o_i, o_j) = \int_S \Pr(o_j, s)\mathrm{Eff}(o_i)(s)\mathrm{d}s$.

Abstract subgoal options model the more general case where executing an option leads to a subgoal for a subset of the state variables (called the *mask*), leaving the rest unchanged. For example, walking to the kitchen shouldn't change the amount of gas in your car. More formally, the state vector can be partitioned into two parts $s = [a, b]$, such that executing o leaves the agent in state $s' = [a, b']$, where $P(b')$ is independent of the distribution over starting states.

A further generalization of the (abstract) subgoal option is the (abstract) *partitioned subgoal option*, which can be partitioned into a finite number of (abstract) subgoal options. For instance, an option for opening doors is not a subgoal option because there are many doors in the world, however it can be partitioned into a set of subgoal options, with one for every door. We assume that the agent has only abstract partitioned subgoal options.

3 Online Active Symbol Acquisition

We now introduce our method for quickly building symbolic models for high-level skills from data, and an algorithm for learning them in a data-efficient manner. It is important to learn the models data-efficiently because real world data from high-level skills is very expensive to collect.

3.1 Fast Construction of a Distribution over Symbolic Option Models

Now we show how to construct a more general model than G that can be used for planning with abstract partitioned subgoal options. The advantages of our approach versus previous methods are that our algorithm is much faster, and the resulting model is Bayesian, both of which are important for the approach to exploration introduced in the next section.

The first step is to find the *factors* [13], partitions of state variables that always change together in the observed data. We use factors to reduce the number of potential masks, i.e. we assume that if state variables i and j always change together in the observations, then this will always occur. An example of a factor could be the (x, y, z) position of your keys, because they are almost never moved along only one axis.

Finding the Factors Recall that the agent can collect observations of the forms (s, o, s') upon executing option o from s, and $(s, O(s))$ when entering a state s, where $O(s)$ is the set of available options in state s. We compute the set of observed masks M from the (s, o, s') observations: each observation's mask is the subset of state variables that differ between s and s'. For each state variable i, let $M_i \subseteq M$ be the subset of the observed masks that contain i. Two state variables i and j belong to the same factor $f \in F$ if and only if $M_i = M_j$. Each factor f is given by a set of state variables and thus corresponds to a subspace S_f.

Let S^* be the set of states that the agent has observed and let S^*_f be the projection of S^* onto the subspace S_f for some factor f. It is important to note that the agent's observations come only from executing partitioned abstract subgoal options. This means that S^*_f consists only of abstract subgoals, because for each $s \in S^*$, s_f was either unchanged from the previous state, or changed to another abstract subgoal. Returning to the keys example, after getting home from work, you normally would not leave your keys at an arbitrary (x, y, z) position, but rather in a small number of specific locations within your house. Thus, the states in S^* can be imagined as a collection of abstract subgoals for each of the factors. Our next step is to build a set of symbols for each factor to represent its abstract subgoals, which we do using unsupervised clustering.

Finding the Symbols For each factor $f \in F$, we find the set of symbols Z^f by clustering S^*_f. Let $Z^f(s_f)$ be the corresponding symbol for state s and factor f. We then map the observed states $s \in S^*$ to their corresponding symbolic states $s^d = \{Z^f(s_f), \forall f \in F\}$, and the observations $(s, O(s))$ and (s, o, s') to $(s^d, O(s))$ and (s^d, o, s'^d), respectively.

We want to build our models within the symbolic state space S^d. Thus we define the *symbolic precondition*, $Pre(o, s^d)$, which returns the probability that the agent can execute an option from some symbolic state, and the *symbolic effects distribution* for a subgoal option o, $\mathit{Eff}(o)$, maps to a subgoal distribution over symbolic states. The next step is to partition the options into abstract subgoal options (in the symbolic state space).

Partitioning the Options For each option o, we initialize the partitioning P^o so that each symbolic state starts in its own partition. We use independent Bayesian sparse Dirichlet-categorical models [20] for the symbolic effects distribution of each option partition.[1] We then perform Bayesian Hierarchical Clustering [9] to merge partitions which have similar symbolic effects distributions.[2]

There is a special case where the agent has observed that an option o was available in some symbolic states S^d_a, but has yet to actually execute it from any $s^d \in S^d_a$. These are not included in the Bayesian Hierarchical Clustering, instead we have a special prior for the partition of o that they belong to. After completing the merge step, the agent has a partitioning P^o for each option o. Our prior is that with probability q_o,[3] each $s^d \in S^d_a$ belongs to the partition $p^o \in P^o$ which contains the symbolic states most similar to s^d, and with probability $1 - q_o$ each s^d belongs to its own partition.

To determine the partition which is most similar to some symbolic state, we first find A^o, the smallest subset of factors which can still be used to correctly classify P^o. We then map each $s^d \in S^d_a$ to the most similar partition by trying to match s^d masked by A^o with a masked symbolic state already in one of the partitions. If there is no match, s^d is placed in its own partition.

Our final consideration is how to model the symbolic preconditions. The main concern is that many factors are often irrelevant for determining if some option can be executed. For example, whether or not you have keys in your pocket does not affect whether you can put on your shoe.

Modeling the Symbolic Preconditions Given an option o and subset of factors F^o, let $S^d_{F^o}$ be the symbolic state space projected onto F^o. We use independent Bayesian Beta-Bernoulli models for the symbolic precondition of o in each masked symbolic state $s^d_{F^o} \in S^d_{F^o}$. For each option o, we use Bayesian model selection to find the the subset of factors F^o_* which maximizes the likelihood of the symbolic precondition models.

The final result is a distribution over symbolic option models H, which consists of the combined sets of independent symbolic precondition models $\{Pre(o, s^d_{F^o}); \forall o \in O, \forall s^d_{F^o_*} \in S^d_{F^o_*}\}$ and independent symbolic effects distribution models $\{\mathit{Eff}(o, p^o); \forall o \in O, \forall p^o \in P^o\}$.

The complete procedure is given in Algorithm 1. A symbolic option model $h \sim H$ can be sampled by drawing parameters for each of the Bernoulli and categorical distributions from the corresponding Beta and sparse Dirichlet distributions, and drawing outcomes for each q_o. It is also possible to consider distributions over other parts of the model such as the symbolic state space and/or a more complicated one for the option partitionings, which we leave for future work.

[1] We use sparse Dirichlet-categorical models because there are a combinatorial number of possible symbolic state transitions, but we expect that each partition has non-zero probability for only a small number of them.

[2] We use the closed form solutions for Dirichlet-multinomial models provided by the paper.

[3] This is a user specified parameter.

Algorithm 1 Fast Construction of a Distribution over Symbolic Option Models

1: Find the set of observed masks M.
2: Find the factors F.
3: $\forall f \in F$, find the set of symbols Z^f.
4: Map the observed states $s \in S^*$ to symbolic states $s^d \in S^{*d}$.
5: Map the observations $(s, O(s))$ and (s, o, s') to $(s^d, O(s))$ and (s^d, o, s'^d).
6: $\forall o \in O$, initialize P^o and perform Bayesian Hierarchical Clustering on it.
7: $\forall o \in O$, find A^o and F^o_*.

3.2 Optimal Exploration

In the previous section we have shown how to efficiently compute a distribution over symbolic option models H. In this section, we first introduce a measure for the amount of uncertainty in H, and then we introduce an algorithm for finding the exploration policy which leads to the greatest expected reduction of this measure. This allows the agent to learn H more data-efficiently.

First we define the *standard deviation* σ_H, the quantity we use to represent the amount of uncertainty in H. To define the standard deviation, we need to also define the distance and mean.

We define the *distance* K from $h_2 \in H$ to $h_1 \in H$, to be the sum of the Kullback-Leibler (KL) divergences[4] between their individual symbolic effect distributions plus the sum of the KL divergences between their individual symbolic precondition distributions:[5]

$$K(h_1, h_2) = \sum_{o \in O} [\sum_{s^d_{F^o_*} \in S^d_{F^o_*}} D_{KL}(Pre^{h_1}(o, s^d_{F^o_*}) \mid\mid Pre^{h_2}(o, s^d_{F^o_*})) + \sum_{p^o \in P^o} D_{KL}(\mathit{Eff}^{h_1}(o, p^o) \mid\mid \mathit{Eff}^{h_2}(o, p^o))].$$

We define the *mean*, $\mathbb{E}[H]$, to be the symbolic option model such that each Bernoulli symbolic precondition and categorical symbolic effects distribution is equal to the mean of the corresponding Beta or sparse Dirichlet distribution:

$$\forall o \in O,\ \forall p^o \in P^o,\ \mathit{Eff}^{\mathbb{E}[H]}(o, p^o) = \mathbb{E}_{h \sim H}[\mathit{Eff}^h(o, p^o)],$$
$$\forall o \in O,\ \forall s^d_{F^o_*} \in S^d_{F^o_*},\ Pre^{\mathbb{E}[H]}(o, s^d_{F^o_*})) = \mathbb{E}_{h \sim H}[Pre^h(o, s^d_{F^o_*}))].$$

The standard deviation σ_H is then simply: $\sigma_H = \mathbb{E}_{h \sim H}[K(h, \mathbb{E}[H])]$. This represents the expected amount of information which is lost if $\mathbb{E}[H]$ is used to approximate H.

Now we define the optimal exploration policy for the agent, which aims to maximize the expected reduction in σ_H after H is updated with new observations. Let $H(w)$ be the posterior distribution over symbolic models when H is updated with symbolic observations w (the partitioning is not updated, only the symbolic effects distribution and symbolic precondition models), and let $W(H, i, \pi)$ be the distribution over symbolic observations drawn from the posterior of H if the agent follows policy π for i steps. We define the optimal exploration policy π^* as:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\ \sigma_H - \mathbb{E}_{w \sim W(H,i,\pi)}[\sigma_{H(w)}].$$

For the convenience of our algorithm, we rewrite the second term by switching the order of the expectations: $\mathbb{E}_{w \sim W(H,i,\pi)}[\mathbb{E}_{h \sim H(w)}[K(h, \mathbb{E}[H(w)])]] = \mathbb{E}_{w \sim W(h,i,\pi)}[K(h, \mathbb{E}[H(w)])]]$.

Note that the objective function is non-Markovian because H is continuously updated with the agent's new observations, which changes σ_H. This means that π^* is non-stationary, so Algorithm 2 approximates π^* in an online manner using Monte-Carlo tree search (MCTS) [3] with the UCT tree policy [11]. π_T is the combined tree and rollout policy for MCTS, given tree T.

There is a special case when the agent simulates the observation of a previously unobserved transition, which can occur under the sparse Dirichlet-categorical model. In this case, the amount of information

[4] The KL divergence has previously been used in other active exploration scenarios [15, 17].

[5] Similarly to other active exploration papers, we define the distance to depend only on the transition models and not the reward models.

Algorithm 2 Optimal Exploration

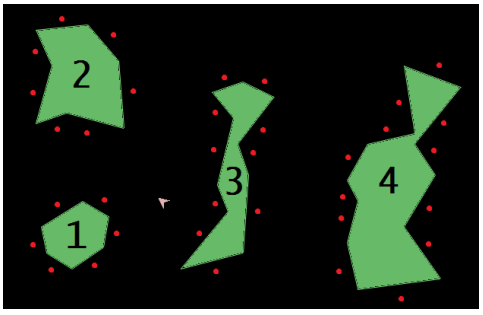
Input: Number of remaining option executions i.
while $i \geq 0$ **do**
  Build H from observations (Algorithm 1).
  Initialize tree T for MCTS.
  while number updates $<$ threshold **do**
    Sample a symbolic model $h \sim H$.
    Do an MCTS update of T with dynamics given by h.
    Terminate current update if depth g is $\geq i$, or unobserved transition is encountered.
    Store simulated observations $w \sim W(h, g, \pi_T)$.
    Score = $K(h, \mathbb{E}[H]) - K(h, \mathbb{E}[H(w)]) - zg$.
  end while
  return most visited child of root node.
  Execute corresponding option; Update observations; i--.
end while

gained is very large, and furthermore, the agent is likely to transition to a novel symbolic state. Rather than modeling the unexplored state space, instead, if an unobserved transition is encountered during an MCTS update, it immediately terminates with a large bonus to the score, a similar approach to that of the R-max algorithm [2]. The form of the bonus is -zg, where g is the depth that the update terminated and z is a constant. The bonus reflects the opportunity cost of not experiencing something novel as quickly as possible, and in practice it tends to dominate (as it should).

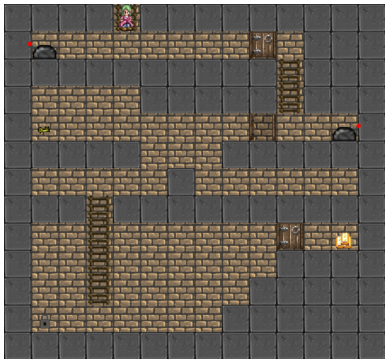
4 The Asteroids Domain

The Asteroids domain is shown in Figure 2a and was implemented using physics simulator pybox2d. The agent controls a ship by either applying a thrust in the direction it is facing or applying a torque in either direction. The goal of the agent is to be able to navigate the environment without colliding with any of the four "asteroids." The agent's starting location is next to asteroid 1.

The agent is given 6 options: move-clockwise, move-counterclockwise, move-to-asteroid-1, move-to-asteroid-2, move-to-asteroid-3, and move-to-asteroid-4. These are described in more detail in Appendix A. Exploring with these options results in only one factor (for the entire state space), with symbols corresponding to each of the 35 asteroid faces as shown in Figure 2a.



(a)



(b)

Figure 2: (a): The Asteroids Domain, and the 35 symbols which can be encountered while exploring with the provided options. (b): The Treasure Game domain. Although the game screen is drawn using large image tiles, sprite movement is at the pixel level.

Results We tested the performance of three algorithms: random, greedy, and our algorithm. For the greedy algorithm, the agent first computes the symbolic state space using steps 1-5 of Algorithm 1, and then chooses the option with the lowest execution count from its current symbolic state. The hyperparameter settings that we use for our algorithm are given in Appendix A.

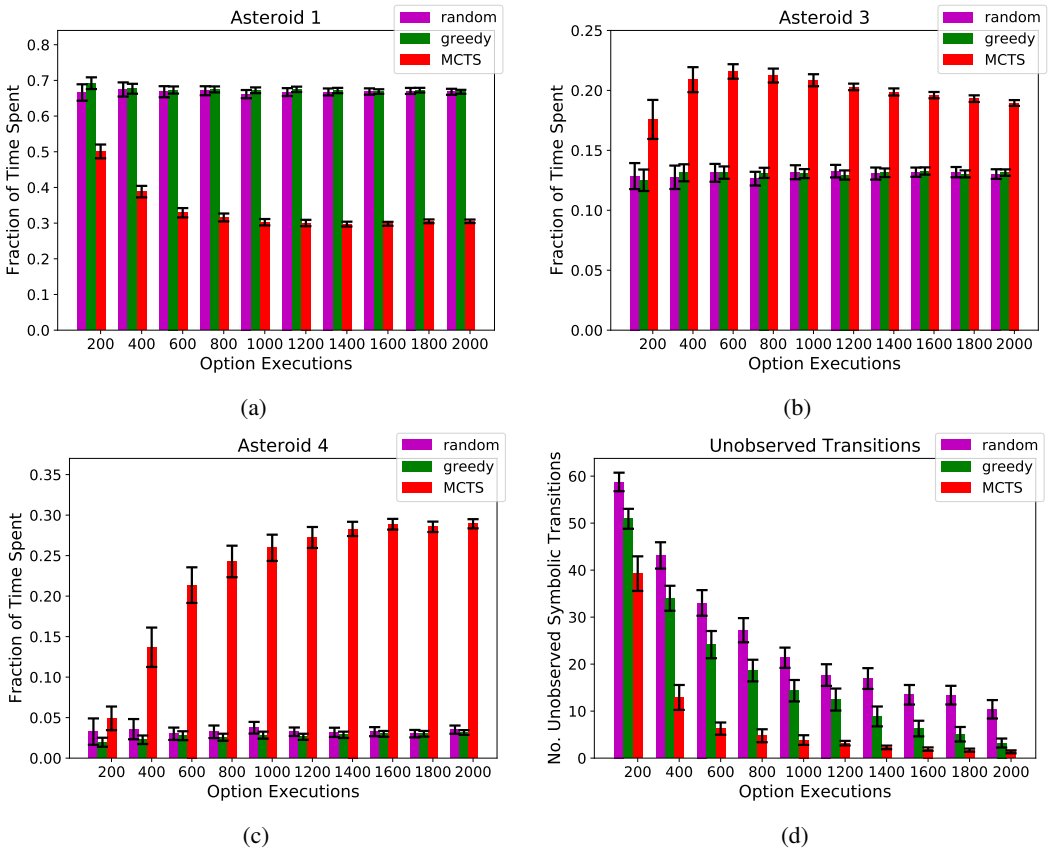


Figure 3: Simulation results for the Asteroids domain. Each bar represents the average of 100 runs. The error bars represent a 99% confidence interval for the mean. (a), (b), (c): The fraction of time that the agent spends on asteroids 1, 3, and 4, respectively. The greedy and random exploration policies spend significantly more time than our algorithm exploring asteroid 1 and significantly less time exploring asteroids 3 and 4. (d): The number of symbolic transitions that the agent has not observed (out of 115 possible). The greedy and random policies require 2-3 times as many option executions to match the performance of our algorithm.

Figures 3a, 3b, and 3c show the percentage of time that the agent spends on exploring asteroids 1, 3, and 4, respectively. The random and greedy policies have difficulty escaping asteroid 1, and are rarely able to reach asteroid 4. On the other hand, our algorithm allocates its time much more proportionally. Figure 4d shows the number of symbolic transitions that the agent has not observed (out of 115 possible). As we discussed in Section 3, the number of unobserved symbolic transitions is a good representation of the amount of information that the models are missing from the environment.

Our algorithm significantly outperforms random and greedy exploration. Note that these results are using an uninformative prior and the performance of our algorithm could be significantly improved by starting with more information about the environment.

5 The Treasure Game Domain

The Treasure Game [13], shown in Figure 2b, features an agent in a 2D, 528×528 pixel video-game like world, whose goal is to obtain treasure and return to its starting position on a ladder at the top of the screen. The 9-dimensional state space is given by the x and y positions of the agent, key, and treasure, the angles of the two handles, and the state of the lock.

The agent is given 9 options: go-left, go-right, up-ladder, down-ladder, jump-left, jump-right, down-right, down-left, and interact. See Appendix A for a more detailed description of the options and the environment dynamics. Given these options, the 7 factors with their corresponding number of

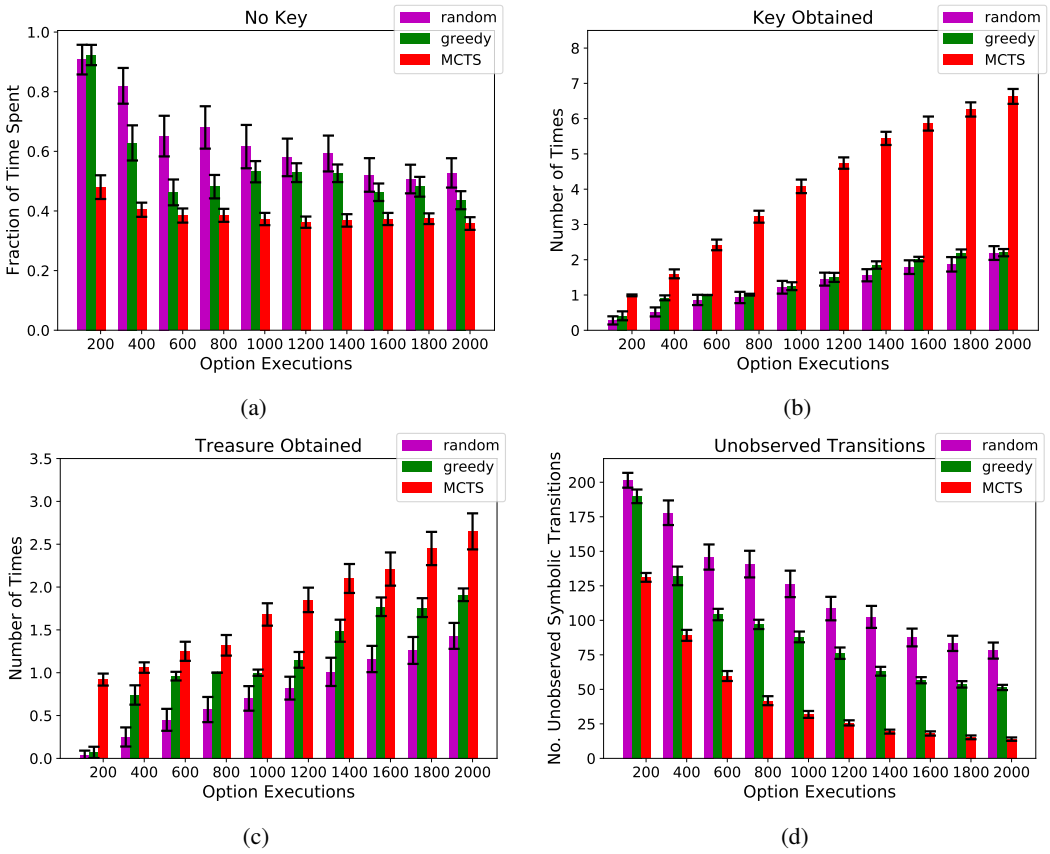


Figure 4: Simulation results for the Treasure Game domain. Each bar represents the average of 100 runs. The error bars represent a 99% confidence interval for the mean. (a): The fraction of time that the agent spends without having the key and with the lock still locked. The greedy and random exploration policies spend significantly more time than our algorithm exploring without the key and with the lock still locked. (b), (c): The average number of times that the agent obtains the key and treasure, respectively. Our algorithm obtains both the key and treasure significantly more frequently than the greedy and random exploration policies. (d): The number of symbolic transitions that the agent has not observed (out of 240 possible). The greedy and random policies require 2-3 times as many option executions to match the performance of our algorithm.

symbols are: $player\text{-}x$, 10; $player\text{-}y$, 9; $handle1\text{-}angle$, 2; $handle2\text{-}angle$, 2; $key\text{-}x$ and $key\text{-}y$, 3; $bolt\text{-}locked$, 2; and $goldcoin\text{-}x$ and $goldcoin\text{-}y$, 2.

Results We tested the performance of the same three algorithms: random, greedy, and our algorithm. Figure 4a shows the fraction of time that the agent spends without having the key and with the lock still locked. Figures 4b and 4c show the number of times that the agent obtains the key and treasure, respectively. Figure 4d shows the number of unobserved symbolic transitions (out of 240 possible). Again, our algorithm performs significantly better than random and greedy exploration.

6 Conclusion

We have introduced a two-part algorithm for data-efficiently learning an abstract symbolic representation of an environment which is suitable for planning with high-level skills. The first part of the algorithm quickly generates an intermediate Bayesian symbolic model directly from data. The second part guides the agent's exploration towards areas of the environment that the model is uncertain about. This algorithm is useful when the cost of data collection is high, as is the case in most real world artificial intelligence applications. Our results show that the algorithm is significantly more data efficient than using more naive exploration policies.

References

[1] A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[2] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[3] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte-Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[4] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28(1):104–126, 2009.

[5] J. Choi and E. Amir. Combining planning and motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4374–4380, 2009.

[6] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics*, November 2009.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[8] E. Gat. On three-layer architectures. In D. Kortenkamp, R.P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. AAAI Press, 1998.

[9] K.A. Heller and Z. Ghahramani. Bayesian hierarchical clustering. In *Proceedings of the 22nd international conference on Machine learning*, pages 297–304. ACM, 2005.

[10] L. Kaelbling and T. Lozano-Pérez. Hierarchical planning in the Now. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2011.

[11] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[12] G.D. Konidaris, L.P. Kaelbling, and T. Lozano-Perez. Constructing symbolic representations for high-level planning. In *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence*, pages 1932–1940, 2014.

[13] G.D. Konidaris, L.P. Kaelbling, and T. Lozano-Perez. Symbol acquisition for probabilistic high-level planning. In *Proceedings of the Twenty Fourth International Joint Conference on Artificial Intelligence*, pages 3619–3627, 2015.

[14] C. Malcolm and T. Smithers. Symbol grounding via a hybrid architecture in an autonomous assembly system. *Robotics and Autonomous Systems*, 6(1-2):123–144, 1990.

[15] S.A. Mobin, J.A. Arnemann, and F. Sommer. Information-based learning by agents in unbounded state spaces. In *Advances in Neural Information Processing Systems*, pages 3023–3031, 2014.

[16] N.J. Nilsson. Shakey the robot. Technical report, SRI International, April 1984.

[17] L. Orseau, T. Lattimore, and M. Hutter. Universal knowledge-seeking agents for stochastic environments. In *International Conference on Algorithmic Learning Theory*, pages 158–172. Springer, 2013.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[19] D. Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst, 2000.

[20] N.F.Y. Singer. Efficient Bayesian parameter estimation in large discrete domains. In *Advances in Neural Information Processing Systems 11: Proceedings of the 1998 Conference*, volume 11, page 417. MIT Press, 1999.

[21] R.S. Sutton, D. Precup, and S.P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[22] J. Wolfe, B. Marthi, and S.J. Russell. Combined task and motion planning for mobile manipulation. In *International Conference on Automated Planning and Scheduling*, 2010.

A Environment Descriptions

A.1 Asteroids Domain Cont.

The agent's 6 options are implemented using PD controllers for the torque and thrust:

1. **move-counterclockwise** and **move-clockwise**: the ship moves from the current face it is adjacent to, to the midpoint of the face which is counterclockwise/clockwise on the same asteroid from the current face. Only available if the ship is at an asteroid.
2. **move-to-asteroid-1**, **move-to-asteroid-2**, **move-to-asteroid-3**, and **move-to-asteroid-4**: the ship moves to the midpoint of the closest face of asteroid 1-4 to which it has an unobstructed path. Only available if the ship is not already at the asteroid and an unobstructed path to some face exists.

The options do not always work as intended, sometimes the ship will crash during the execution of an option, which resets the environment. If the ship tries to move from asteroid a to asteroid $b > a$, it crashes with probability 0.5. As designed, these options do not have the subgoal property because the outcome of executing each option is dependent on which face of which asteroid the option was executed from. However, they can be partitioned so that the subgoal property holds because the outcome is only dependent on which face the option was executed from.

A.2 Treasure Game Domain Cont.

The low-level actions available to the agent are move up, down, left, and right, jump, and interact. The 4 movement actions move the agent between 2 and 4 pixels uniformly at random in the appropriate direction. There are three doors which may block the path of the agent. The top two doors are oppositely open and closed; flipping one of the two handles switches their status. The bottom door which guards the treasure can be opened by the agent obtaining the key and using it on the lock. The interact action is available when the agent is standing in front of a handle, or when it possesses the key and is standing in front of the lock. In the first case, executing the interact action flips the handle's position with probability 0.8, and in the second case, the lock is unlocked and the agent loses the key. Whenever the agent has possession of the key and/or the treasure, they are displayed in the lower-right corner of the screen. The agent returning to the top ladder resets the environment. The agent's 9 options are implemented using simple control loops:

1. **go-right** and **go-left**: the agent moves continuously right/left until it reaches a wall, edge, object it can interact with, or ladder. Only available when the agent's way is not directly blocked.
2. **up-ladder** and **down-ladder**: the agent ascends/descends a ladder. Only available when the agent is directly below/above a ladder.
3. **down-left** and **down-right**: the agent falls off an edge onto the nearest solid cell on its left/right. Only available when they would succeed.
4. **jump-left** and **jump-right**: the agent jumps and moves left/right for about 48 pixels. Only available when the area above the agent's head, and above its head and to the left/right, are clear.
5. **interact**: same as the low-level interact action.

These options, like the low-level actions they are composed of, all have at least a small amount of stochasticity in their outcomes. Additionally, when the agent executes one of the jump options to reach a faraway ledge, for instance when it is trying to get the key, it succeeds with probability 0.53, and misses the ledge and lands directly below with probability 0.47.

A.3 Hyperparameter Settings

In each run the agent had access to the exact number of option executions it had to explore with. For MCTS, we used the UCT tree policy with $c = 2$, a random rollout policy, and performed 1000 updates. Also, during UCT option selection, we normalized a node's score using the highest and lowest scores seen so far. For the sparse Dirichlet-Multinomial models, we used hyperparameter $\alpha = 0.5$ and the

prior probability over the size of the support was given by a geometric distribution with parameter 0.5. For the state clustering (step 3 of Algorithm 1), we used the DBSCAN algorithm [7] implemented in scikit-learn [18] with parameters $min\text{-}samples = 1$, and $\epsilon = 1.5$ for the Asteroids domain and $\epsilon = 0.05$ for the Treasure Game domain. For Algorithm 2, we set $z = 10$. For all options o, we set $q_o = 0.3$.