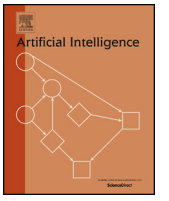
Artificial Intelligence 324 (2023) 104001

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Reward-respecting subtasks for model-based reinforcement learning

Richard S. Sutton [a,b,c,d], Marlos C. Machado [a,b,c,d,*], G. Zacharias Holland [a], David Szepesvari [a], Finbarr Timbers [a], Brian Tanner [a], Adam White [a,b,c,d]

[a] DeepMind, Edmonton, Alberta, Canada
[b] University of Alberta, Edmonton, Alberta, Canada
[c] Alberta Machine Intelligence Institute (Amii), Edmonton, Alberta, Canada
[d] Canada CIFAR AI Chair, Canada

ARTICLE INFO

Article history:
Received 4 November 2022
Received in revised form 27 June 2023
Accepted 29 August 2023
Available online 6 September 2023

Keywords:
Planning
Model-based reinforcement learning
Temporal abstraction
Options
Feature attainment
STOMP progression

ABSTRACT

To achieve the ambitious goals of artificial intelligence, reinforcement learning must include planning with a model of the world that is abstract in state and time. Deep learning has made progress with state abstraction, but temporal abstraction has rarely been used, despite extensively developed theory based on the options framework. One reason for this is that the space of possible options is immense, and the methods previously proposed for option discovery do not take into account how the option models will be used in planning. Options are typically discovered by posing subsidiary tasks, such as reaching a bottleneck state or maximizing the cumulative sum of a sensory signal other than reward. Each subtask is solved to produce an option, and then a model of the option is learned and made available to the planning process. In most previous work, the subtasks ignore the reward on the original problem, whereas we propose subtasks that use the original reward plus a bonus based on a feature of the state at the time the option terminates. We show that option models obtained from such reward-respecting subtasks are much more likely to be useful in planning than eigenoptions, shortest path options based on bottleneck states, or reward-respecting options generated by the option-critic. Reward respecting subtasks strongly constrain the space of options and thereby also provide a partial solution to the problem of option discovery. Finally, we show how values, policies, options, and models can all be learned online and off-policy using standard algorithms and general value functions.

1. The challenge of discovering temporal abstractions

A major goal of artificial intelligence (AI) is to understand how an AI agent can obtain and reason with a high-level model of the world. The interaction between AI agent and world consists entirely of low-level, fleeting signals such as pixel values and motor torques, yet from this the agent must produce a model of the world that supports reasoning about high-level things such as which object to pick up, whether to walk to work or drive, or which country to vacation in. How abstractions can be obtained to bridge the gap between the low and high levels in planning is a recurring theme in the history of AI (e.g., [1–3]).

* Corresponding author at: University of Alberta, Edmonton, Alberta, Canada.
E-mail address: machado@ualberta.ca (M.C. Machado).

https://doi.org/10.1016/j.artint.2023.104001

In the reinforcement learning (RL) approach to AI, the issue of abstraction in planning arises in model-based RL, in which an RL agent learns a model of the transition dynamics of its environment and then converts that model into improvements in its policy and, commonly, in its approximate value function. This conversion process is planning and is typically computationally expensive and distributed over many time steps, or even performed offline. Learning and planning with a model may enable dramatically faster adaptation whenever the agent is long-lived, the environment is non-stationary, and much of the environment's transition dynamics is stable (as approximately modeled by the agent).

For planning to be tractable on large problems, the RL agent's model must be abstract in state and time. Abstraction in state is important because the original states of the world are too numerous to deal with individually, or may not be observable by the agent. In these cases, how the state should be constructed from observations is an important problem on which much work has been done with deep learning (e.g., [4–6]) and other methods (e.g., [7–9]). We do not address state abstraction in this paper other than by allowing the agent's state representation to be a non-Markov feature vector.

This paper concerns how we should create and work with environment models that are abstract in *time*. The most common way of formulating temporally-extended and temporally-variable ways of behaving in a reinforcement learning agent is as *options* [10], each of which comprises a way of behaving (a policy) and a way of stopping. The appeal of options is that they are in some ways interchangeable with actions. Just as we can learn models of actions' consequences and plan with those models, so we can learn and plan with models of options' effects.

There remains the critical question of where the options come from. A common approach to option discovery is to pose subsidiary tasks such as reaching a bottleneck state or maximizing the cumulative sum of a sensory signal other than reward. Given such subtasks, the agent can develop temporally abstract structure for its cognition by following a standard progression in which each subtask is solved to produce an option, the option's consequences are learned to produce a model, and the model is used in planning. We refer to this progression (SubTask, Option, Model, Planning) as the STOMP progression for the development of temporally-abstract cognitive structure. All the steps of the STOMP progression were described in the original paper on options [10], and the progression has been used in several previous works (e.g., [11–15]). The steps of the progression are often most simply described and implemented sequentially, as they are in this paper, but there is no reason that they cannot proceed simultaneously, in parallel. This is in fact the architecture envisioned in the long run.

The primary conceptual innovation of the current work is to introduce the notion of a *reward-respecting subtask*, that is, of a subtask that optimizes the rewards of the original task until terminating in a state that is sometimes of high value. Reward-respecting subtasks contrast with commonly used subtasks, such as shortest path to bottleneck states (e.g., [16–18]), pixel maximization [19], and diffusion maximization [20], many of which explicitly maximize the cumulative sum of a signal other than the reward of the original task.

For example, consider the two-room gridworld shown inset in Fig. 1, with a start state in one room, a terminal goal state in the other, and a bottleneck or *hallway* state in-between. The usual four actions move the agent one cell up, down, right, or left, unless blocked by a wall. A reward of $+1$ is received on reaching the goal state, which ends the episode. Transitions ending in the gray region between the start and hallway states produce a reward of -1 per step, while all other transitions produce a reward of zero. The discount factor is $\gamma = 0.99$, so the optimal path from start to goal, traveling the roundabout path that avoids the field of negative reward, yields a return of $v_*(s_0) = 0.99^{17} \approx 0.843$. The hallway state is a bottleneck and thus is a natural terminating subgoal for a subtask on this problem (as in [16,21]). With a reward-respecting subtask, the agent would learn a path to the hallway state that maximizes the reward along the way; in this gridworld it finds the option that goes down from the start state and *around* the field of negative rewards. In contrast, solving the version of this subtask that does not take the reward into consideration would learn the option taking the shortest path from start to hallway, passing *through* the field of negative rewards.

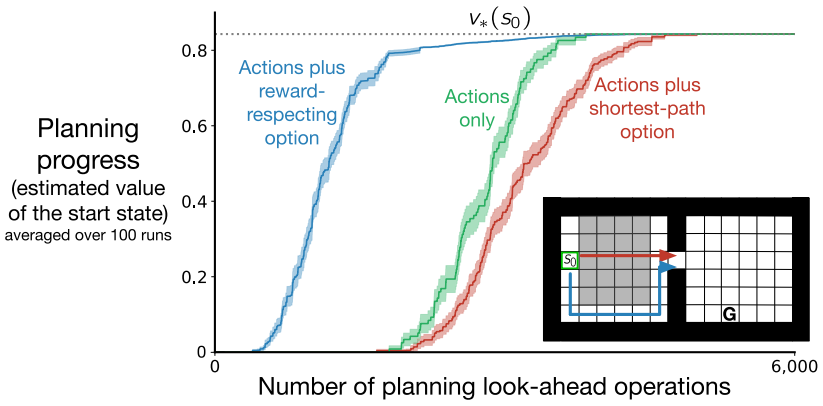


Fig. 1. Illustrative example using the two-room gridworld (shown inset) contrasting planning with reward-respecting and shortest-path options for reaching the bottleneck state. In all cases the planner was given accurate models of the actions and options. Planning with the model of the reward-respecting option was much more efficient. The shading represents one standard error.

Which of these two options—*reward respecting* or *shortest path*—is the more useful when their models are learned and used in planning? Assuming optimal options and that their models and those of the primitive actions are accurate, we can compare the progress of planning by value iteration (as detailed later in this paper) when augmented with models of each of the two options (Fig. 1). In all cases, planning eventually found the optimal policy and the correct value of the start state. However, planning using the reward-respecting option was much faster than planning using the shortest-path option or planning using the primitive actions alone. Planning with the shortest-path option was actually *slower* by this measure than planning with actions alone. This could be because the planning with options required one additional look-ahead operation per state updated (four instead of five). (Planning with the reward-respecting option was more efficient *despite* this disadvantage.) Or the poor performance of planning with the shortest-path option may just have been because that option was rarely part of a good trajectory. The shortest-path option passes through the field of negative reward, while the reward-respecting option follows the roundabout path hugging the bottom of the first room. The latter is much more likely to be part of the final optimal policy.

The next four sections detail the four steps of the STOMP progression in sequence. We formalize the concept of subtasks through the language of general value functions (GVFs) in Section 2. This perspective allows us to clearly introduce the notion of reward-respecting subtask. Moreover, it provides clarity in the option-discovery process by showing that different option-discovery methods differ only in how they define the GVFs' cumulants and stopping values. Given these subtasks, the subsequent sections detail option learning (Section 3), model learning (Section 4), and planning (Section 5). In Section 3 we introduce a general update procedure, UWT, for learning both options and models through various temporal-difference errors, showing how the learning problems differ only in their targets. We further analyze the impact of different ways of defining subtasks in Section 6. Our most important empirical results, comparing different methods for discovering options in a stochastic environment, are in the penultimate Section 7.

2. Reward-respecting subtasks

In this section we define the agent–environment interaction, the main task, GVF subtasks, reward-respecting subtasks, reward-respecting subtasks of feature attainment, and the specific subtasks used in the illustrative example (Fig. 1). Subtask specification is the first step in the STOMP progression for developing temporally-abstract cognitive structure.

We consider an agent interacting with its environment in a sequence of episodes, each beginning in environment state $S_0 \doteq s_0 \in \mathcal{S}$ and ending in terminal state $S_L \doteq \perp$[1] at time step $L \in \mathbb{N}$. At time steps $t < L$, the agent selects an action $A_t \in \mathcal{A}$, and in response the environment emits a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and transitions to a next state $S_{t+1} \in \mathcal{S} + \perp$ with probability $p(s', r|s, a) \doteq \Pr\{S_{t+1}=s', R_{t+1}=r \mid S_t=s, A_t=a\}$, where $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{R}$ are finite sets. Capitalized letters denote random variables that differ from step to step and episode to episode; technically these should be indexed by an episode number, but we suppress that in our notation.

The agent's main task is to find a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximizes the expected discounted sum of rewards $\mathbb{E}\left[R_1 + \gamma R_2 + \cdots + \gamma^{L-1} R_L\right]$ where $\gamma \in [0, 1)$ is the *discount rate*, a parameter of the problem. The value function $v_\pi : \mathcal{S} \to \mathbb{R}$ specifies this expectation for any starting state s:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[\sum_{j=1}^{L} \gamma^{j-1} R_j \,\middle|\, S_0=s\right], \quad \forall s \in \mathcal{S}, \tag{1}$$

where the expectation is implicitly conditional on the actions being selected according to π. Value functions are defined for any π, but most commonly π is the policy currently being followed by the agent and which gradually approaches an optimal policy (which maximizes (1)).

A general value function (GVF [22,23]) extends the idea of a value function in three ways. First, the reward R_t is generalized to an arbitrary quantity to be added up, a *cumulant* $C_t \doteq c(S_t)$ for any *cumulant function* $c : \mathcal{S} \to \mathbb{R}$. Second, the accumulation can stop not just at termination, but at a time with probability $\beta(S_t)$ for an arbitrary *stopping function* $\beta : \mathcal{S} \to [0, 1]$. Third, at the time of stopping, K, there is a further *stopping value* $z(S_K)$ added to the accumulation, for any *stopping-value function* $z : \mathcal{S} \to \mathbb{R}$. Stopping is different from termination, as termination resets the state, whereas stopping has no effect on the state trajectory. At termination, the accumulation always stops ($\beta(\perp) \doteq 1$) with zero stopping value ($z(\perp) \doteq 0$). Formally, a GVF $v^{c,z}_{\pi,\beta} : \mathcal{S} \to \mathbb{R}$ is defined by

$$v^{c,z}_{\pi,\beta}(s) \doteq \mathbb{E}_{\pi,\beta}\left[\sum_{j=1}^{K} \gamma^{j-1} c(S_j) + \gamma^{K-1} z(S_K) \,\middle|\, S_0=s\right], \quad \forall s \in \mathcal{S}, \tag{2}$$

where the expectation is conditional on the actions being determined by π and the stopping time $K \leq L$ being determined by β. The GVF gives the expected sum of the cumulant plus the stopping value if the policy were followed from s, stopping according to β.

[1] The symbol $\doteq$ is used in this paper to denote an equality that holds by definition rather than one that follows from previously established definitions.

To use GVFs to formulate subtasks, the superscript functions c, z are taken as fixed and as defining the subtask, and the subscript functions π, β are varied and taken as a possible solution. The subscript functions specify a policy and a stopping function, in other words, an *option* [10].[2] Options are possible ways of behaving and stopping. If option π, β were initiated in state S_t, then A_t and subsequent actions would be selected according to π until the option ended, or *stopped*, according to β at step K. To solve the subtask is to find an option which maximizes (2).

The main task is a special case of a GVF task in which $C_t \doteq R_t$ and stopping is not allowed ($\beta \doteq 0$ or $z \doteq -\infty$). Shortest path subtasks are defined by $C_t \doteq -1$ and $z(s) \doteq 0$ at subgoal states and $\beta \doteq 0$ or $z(s) \doteq -\infty$ otherwise. GVF tasks include all the common subtasks in the literature including those based on curiosity and intrinsic motivation (e.g., [24,25]).

Reward-respecting subtasks are GVF tasks whose cumulant is identical to the reward and whose stopping values take into account the estimated value of the state stopped in. That is, for a reward respecting subtask, $c(S_t) \doteq R_t$, and z is based on, but not identical to, v_π. The stopping values should not equal the estimated values because then the subtask would approximate the main task and solving it would probably add nothing new. Moreover, if there are multiple subtasks, each should have its own stopping values.

In this paper we focus on reward-respecting subtasks whose stopping values are designed to encourage solutions (options) that stop when a particular state feature is high. Such *subtasks of feature attainment* are appealing in several ways, but are not essential to this paper's main conclusions. Other kinds of reward-respecting subtasks could have been used without otherwise affecting the STOMP progression.

To describe subtasks of feature attainment precisely, we need to describe our value-function approximation. We assume that the state is represented as a feature vector $\mathbf{x}_t \doteq \mathbf{x}(S_t)$ for some *feature function* $\mathbf{x} : \mathcal{S} \to \mathbb{R}^d$ (with $\mathbf{x}(\perp) \doteq \vec{0}$) which might be provided by a domain expert or might be hidden-unit activities of a neural network. We further assume that the approximation $\hat{v}$ of v_π is linear in $\mathbf{x}(s)$ and a modifiable weight vector $\mathbf{w} \in \mathbb{R}^d$:

$$v_\pi(s) \approx \hat{v}(\mathbf{x}(s), \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s), \quad \forall s \in \mathcal{S}, \tag{3}$$

where w_i and $x_i(s)$ are individual components of $\mathbf{w}$ and $\mathbf{x}(s)$, respectively.

Informally, to solve the reward-respecting subtask for attaining the ith feature is to find an option that obtains a lot of reward and stops when x_i is high. More formally, let i be a feature whose weight w_i in the linear approximate value function (3) varies over time, and let $\overline{w}^i$ denote one of its largest values, called the *bonus weight*. The stopping-value function for the ith subtask is then defined as the estimated value, except using the optimistic bonus weight $\overline{w}^i$ in place of the usual weight w_i. That is, the ith subtask's stopping-value function is

$$z^i(s) \doteq \mathbf{w}^\top \mathbf{x}(s) - w_i x_i(s) + \overline{w}^i x_i(s). \tag{4}$$

The quantity $(\overline{w}^i - w_i)x_i(s)$ is sometimes called the *stopping bonus* because it is the bonus for stopping in state s beyond its (estimated) value on the main task. The stopping bonus is zero if $x_i(s)$ is zero; to get a large bonus, the option must stop in states of high estimated value in which feature i is also high. Note that z^i does not actually depend on w_i; the inner product contains one term of $w_i x_i$ which cancels with the equation's second term, leaving w_i effectively *replaced* by the bonus weight. Also note we are using the index i both as a subtask number when in the superscript position and as a feature number when in the subscript position. We use this convention throughout the paper. Subtasks of this form are termed *reward-respecting subtasks of feature attainment*.

Generally, it is only useful to construct subtasks for attaining a feature i if w_i, its estimated contribution to value on the main task, is sometimes high and sometimes low. If w_i never varied, then its static value could be learned once and never have to be changed by planning. As the ultimate use of all subtasks in the STOMP progression is for planning, such a subtask would never be useful. If w_i does vary, then its bonus weight is set to one of its higher values so that an option can be learned in preparation for the times at which it is high.

The reward-respecting subtask used in the illustrative example (Fig. 1) was for the tabular feature for the hallway state, with bonus weight $\overline{w}^h \doteq 1$, where h denotes the index of the feature for the hallway state. The shortest-path subtask used $C_t \doteq -1$ and stopped upon reaching the hallway or terminal goal states.

3. Option learning

In this section we specify the off-policy learning algorithms we use to approximate the optimal value functions and optimal options, which is the second step in the STOMP progression.

We describe these algorithms in a somewhat unusual way that lets us cover all the cases very compactly and uniformly, including the model learning cases in the next section. First we define a general TD (Temporal Difference) error function $\delta : \mathbb{R}^4 \times [0, 1] \to \mathbb{R}$:

[2] Options as originally formulated also specified a set of states in which they could be initiated. We don't use that in this work so we elide it for simplicity.

$$\delta(c, z, v, v', \beta) \doteq c + \beta z + \gamma(1-\beta)v' - v. \tag{5}$$

Second, we define a general update procedure for learning with traces, which we call *UpdateWeights&Traces* (UWT):

Procedure UWT$\big(\mathbf{w}, \mathbf{e}, \nabla, \alpha\delta, \rho, \gamma\lambda(1-\beta)\big)$:
$\mathbf{e} \leftarrow \rho(\mathbf{e} + \nabla)$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{e}$
$\mathbf{e} \leftarrow \gamma\lambda(1-\beta)\mathbf{e}$

The first two arguments to UWT are a weight vector and an eligibility-trace vector. These arguments are both inputs and outputs; the same pair are expected to be provided together on every time step. The weight vector is the ultimate result of learning. The eligibility trace is a short-term memory that helps with credit assignment. The third argument is usually a gradient vector with respect to the weight vector. The fourth and sixth arguments to UWT are scalars—the names of the formal arguments are just suggestive of their use. Finally, the fifth argument is a scalar importance-sampling ratio used in off-policy learning (for on-policy learning it should be one). Notice we do not use UWT in a prescriptive way; nothing prevents us from using more efficient learning algorithms in each step of the STOMP progression. We use UWT here to compactly describe the learning algorithms used in each step of the STOMP progression while also highlighting how similar they are.

As an example, here is how these tools would be used to implement on-policy linear TD(λ) [26] on the main task. First we would zero-initialize $\mathbf{w}$ and the corresponding eligibility-trace vector $\mathbf{e} \in \mathbb{R}^d$. Then we would have the agent behave according to some policy π and, on each time step in which S_t is nonterminal, we would do:

$$\delta \leftarrow \delta(R_{t+1}, 0, \hat{v}(\mathbf{x}_t, \mathbf{w}), \hat{v}(\mathbf{x}_{t+1}, \mathbf{w}), 0), \text{ and} \tag{6}$$

$$\text{UWT}(\mathbf{w}, \mathbf{e}, \nabla_{\mathbf{w}}\hat{v}(\mathbf{x}_t, \mathbf{w}), \alpha\delta, 1, \gamma\lambda), \tag{7}$$

where α and λ are step-size and bootstrapping parameters respectively. Note that in the linear case, $\nabla\hat{v}(\mathbf{x}_t, \mathbf{w})$ in (7) is just $\mathbf{x}_t$.

As another example, suppose the policy π is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and we want to learn it as well, in an actor-critic algorithm [27–29]. This would be achieved by invoking UWT one more time on each step, immediately after (7):

$$\text{UWT}(\boldsymbol{\theta}, \mathbf{e}', \nabla_{\boldsymbol{\theta}} \ln\pi(A_t|S_t, \boldsymbol{\theta}), \alpha'\delta, 1, \gamma\lambda'), \tag{8}$$

where $\mathbf{e}' \in \mathbb{R}^{d'}$ is another zero-initialized eligibility-trace vector, and α' and λ' are step-size and bootstrapping parameters for the actor.

Now we show how to use these tools in the second step of the STOMP progression to learn the value functions and options for the subtasks from off-policy experience. Let $\mathcal{T} \subset \{1, \ldots, d\}$ be the set of features for which we have subtasks. Each subtask $i \in \mathcal{T}$ will have a value-function weight vector $\mathbf{w}^i \in \mathbb{R}^d$ and a policy weight vector $\boldsymbol{\theta}^i \in \mathbb{R}^{d'}$ such that $\hat{v}(\mathbf{x}(s), \mathbf{w}^i) \doteq \mathbf{w}^{i\top}\mathbf{x}(s) \approx v^{r,z^i}_{\pi^i,\beta^i}(s)$, as in (2), where $r(S_t) \doteq R_t$, $\pi^i \doteq \pi(\cdot|\cdot, \boldsymbol{\theta}^i)$, and

$$\beta^i(s) \doteq \begin{cases} 1, & \text{if } z^i(s) \geq \mathbf{w}^{i\top}\mathbf{x}(s); \\ 0, & \text{otherwise,} \end{cases} \qquad \forall i \in \mathcal{T}, s \in \mathcal{S}, \tag{9}$$

with $\beta^i(\perp) \doteq 1$. Under this definition, the option π^i, β^i stops in state s if the stopping value, $z^i(s)$, which is the estimated main-task value using the bonus weight $\overline{w}^i$ instead of $\mathbf{w}_i$, is greater than or equal to the estimated subtask value. That is, the option does not stop if the estimated subtask value of continuing is better than the stopping value.

We are learning off-policy, so we need to use the importance sampling ratios $\rho^i_t \doteq \frac{\pi^i(A_t|S_t)}{\mu(A_t|S_t)}$, where $\mu : \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the *behavior policy* (the policy actually used to select actions, as opposed to the policies being learned about). For each subtask $i \in \mathcal{T}$, in addition to $\mathbf{w}^i$ and $\boldsymbol{\theta}^i$ we will also have eligibility-trace vectors $\mathbf{e}^i \in \mathbb{R}^d$ and $\mathbf{e}'^i \in \mathbb{R}^{d'}$, all initialized to zero. Then, on each time step on which S_t is nonterminal, for each $i \in \mathcal{T}$ we do:

$$\begin{aligned} &\delta \leftarrow \delta\big(R_{t+1}, z^i(S_{t+1}), \hat{v}(\mathbf{x}_t, \mathbf{w}^i), \hat{v}(\mathbf{x}_{t+1}, \mathbf{w}^i), \beta^i(S_{t+1})\big), \\ &\text{UWT}\Big(\mathbf{w}^i, \mathbf{e}^i, \nabla_{\mathbf{w}}\hat{v}(\mathbf{x}_t, \mathbf{w}^i), \alpha\delta, \rho^i_t, \gamma\lambda\Big(1-\beta^i(S_{t+1})\Big)\Big), \text{ and} \\ &\text{UWT}\Big(\boldsymbol{\theta}^i, \mathbf{e}'^i, \nabla_{\boldsymbol{\theta}^i} \ln\pi(A_t|S_t, \boldsymbol{\theta}^i), \alpha'\delta, \rho^i_t, \gamma\lambda'\Big(1-\beta^i(S_{t+1})\Big)\Big). \end{aligned} \tag{10}$$

Under this algorithm, the learned approximate values $\hat{v}(\mathbf{x}(s), \mathbf{w}^i) \doteq \mathbf{w}^{i\top}\mathbf{x}(s)$ come to approximate the *optimal subtask values* $v^i_*(s) \doteq \max_{\pi,\beta} v^{r,z^i}_{\pi,\beta}(s)$, for all $s \in \mathcal{S}$ and $i \in \mathcal{T}$, and the options $\langle\pi(\cdot|\cdot, \boldsymbol{\theta}^i), \beta^i\rangle$ come to approximate corresponding optimal options. In the tabular case, the approximations become exact with sufficient exploration and if the step sizes are decreased appropriately.

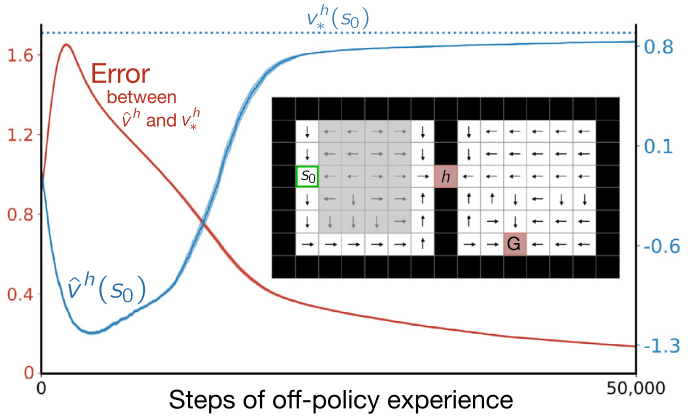


Fig. 2. Option learning experiment. Algorithm (10) finds an optimal option and its value function for the reward-respecting subtask for attaining the hallway feature. The red line shows the root-mean-squared error between the estimated and optimal values decreasing toward zero (left scale). The blue line shows the estimated value on the hallway subtask approaching its optimal value of $v_*^h(s_0) = \gamma^{11} \approx .895$ (right scale). These data are averages over 100 runs and the shading is one standard error. **Inset**, the arrows show a learned option policy and the red cells show the states in which that option stops. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

We applied this algorithm to the hallway-feature-attaining subtask introduced as an illustrative example in Fig. 1, using a behavior policy that selected all four actions with equal probability: $\mu(a|s) \doteq 0.25$, for all $s \in \mathcal{S}, a \in \mathcal{A}$. The state-feature vectors were one-hot, with $d = 72$ for the 72 non-terminal grid cells. The policy was of the softmax form with linear preferences:

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{e^{\boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a)}}{\sum_b e^{\boldsymbol{\theta}^\top \boldsymbol{\phi}(s,b)}}, \tag{11}$$

where the state-action feature vectors $\boldsymbol{\phi}(s, a) \in \mathbb{R}^{d'}$ were again one-hot ($d' = 288$). The weight vectors were all initialized to zero, so the initial approximate value function was everywhere zero and the initial policy was equi-probable random. The parameters where $\alpha = \alpha' = 0.1$ and $\lambda = \lambda' = 0$.

As a measure of the quality of the learned value function, we recorded the root mean squared error (RMSE) between the estimated and optimal state values at each step, averaged over all states, and as a measure of the quality of the policy, we recorded the estimated value of the start state $\hat{v}^h(s_0) \doteq \hat{v}(s_0, \mathbf{w}^h)$ at each step (see Fig. 2). The error from optimal values first rises as $\hat{v}^h$ approximates the value of the random policy, then falls toward zero as the option becomes optimal for the hallway-attaining subtask. The estimated value $\hat{v}^h(s_0)$ starts at zero, then falls as the near-random policy wanders into the gray field of negative reward, and finally rises toward $v_*^h(s_0) = \gamma^{11} \approx .895$, the value of the start state under the optimal policy for the hallway subtask.

These results show that the actor-critic algorithm (10) is able to learn the correct option policy and value function from off-policy data. Notice that the option stops either at the hallway state or the goal state (shown in red in Fig. 2). Reward-respecting options tradeoff rewards, the value of the state, and the stopping bonus. In the right room, close to the hallway state it is more beneficial to go to the hallway state because the stopping bonus plus the value of that state is greater than the value of directly going to the goal state. In states closer to the goal state, it is better to directly go to the goal state rather than back to the hallway.

Note that the policy and value function for the main task can also be learned by the actor-critic algorithm (10) simply by considering the main task to be one more subtask, say subtask 0, with $\beta^0 \doteq 0$.

4. Model learning

In this section we describe the third step in the STOMP progression: learning a model of the environment's action and option transitions. Recall that an option is a pair, $o \doteq (\pi_o, \beta_o)$, consisting of a policy and a stopping function. Actions are a special case of options in which the policy π_o always selects the action and the stopping function always stops, $\beta_o(s) = 1$, for all $s \in \mathcal{S}$. Let $\mathcal{O}(s)$ denote the set of options (including actions) available in state s, and let $\mathcal{O}$ denote the set of all options (unioning over all states). With a slight abuse of notation, we allow these sets of options to include the indices of state-features $i \in \mathcal{I}$ for which the agent has learned an option; when such a feature index appears in a position that is expecting an option, we mean the option corresponding to that feature.

The *ideal* model is expressed in terms of the underlying environment states $s \in \mathcal{S}$ and exactly matches the true underlying dynamics. Like all models, it is comprised of a reward part and a state-transition part. The reward part is a function $r : \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ returning the expected cumulative discounted reward if the option were executed starting from the state:

$$r(s,o) \doteq \mathbb{E}_{\pi_o,\beta_o}\left[\sum_{t=1}^{K}\gamma^{t-1}R_t \;\middle|\; S_0{=}s\right], \quad \forall s \in \mathcal{S}, o \in \mathcal{O}(s), \tag{12}$$

where the expectation is conditional on actions being selected by π_o and the stopping time K being determined by β_o. The state-transition part of an ideal model is a function $p : \mathcal{S} \times \mathcal{S} \times \mathcal{O} \to [0,1]$ returning, for each state s in which an option might be started, the probability of stopping in each state s', discounted by the time until stopping:

$$p(s'|s,o) \doteq \sum_{t=1}^{\infty}\gamma^k \Pr\{K{=}t, S_t{=}s' \,|\, S_0{=}s\}, \quad \forall s \in \mathcal{S}, o \in \mathcal{O}(s), \tag{13}$$

where the probability is conditional on the actions being selected according to the policy of option o and K being determined by its stopping function. Note that we write the p function with a $|$, suggesting that it is a probability distribution, but for $\gamma < 1$ it is not. This precise form is dictated by the requirement that option models and action models be interchangeable in planning methods such as value iteration (see next section).

Approximate models do not in general have access to environmental states, but instead must work with a representation of state constructed by the agent, which might be called the *agent state*. Unlike the environment state, the agent state is generally not a Markov summary of the past. In this paper we assume the agent state is a feature vector $\mathbf{x} \in \mathbb{R}^d$ which can be determined from the environment state by a known function $\mathbf{x} : \mathcal{S} \to \mathbb{R}^d$. The reward part of an approximate model of an option is a function $\hat{r} : \mathbb{R}^d \times \mathcal{O} \to \mathbb{R}$ such that

$$\hat{r}(\mathbf{x}(s), o) \approx r(s,o), \quad \forall s \in \mathcal{S}, \tag{14}$$

weighted over some distribution of states such as the state distribution under the behavior policy. The state-transition part of an approximate model can take several forms [30]. In this paper we use an *expectation* model [31], in which the state-transition part is a function $\hat{\mathbf{n}} : \mathbb{R}^d \times \mathcal{O} \to \mathbb{R}^d$ such that

$$\hat{\mathbf{n}}(\mathbf{x}(s), o) \approx \sum_{s' \in \mathcal{S}} p(s'|s,o)\mathbf{x}(s') = \mathbb{E}_o\left[\gamma^K \mathbf{x}(S_{t+K}) \;\middle|\; S_t{=}s\right], \quad \forall s \in \mathcal{S}, \tag{15}$$

under some state weighting, perhaps given by the behavior policy, and where the expectation is conditional on the actions being selected according to o's policy and the stopping time K being determined by o's stopping function.

For the illustrative example in Fig. 1, we learned models of the four options corresponding to actions and of the one reward-respecting option for attaining the hallway state. The approximate model was linear in the state-feature vector, meaning that

$$\hat{r}(\mathbf{x}, o) \doteq \mathbf{w}_r^{o\top}\mathbf{x} \quad \text{and} \quad \hat{\mathbf{n}}(\mathbf{x}, o) \doteq \mathbf{W}^o\mathbf{x}, \tag{16}$$

where each $\mathbf{w}_r^o \in \mathbb{R}^d$ is a learned weight vector, and each $\mathbf{W}^o$ is a $d \times d$ matrix, with rows $\mathbf{w}_j^o$. In this tabular problem the states were represented by one-hot feature vectors ($d = |\mathcal{S}| = 72$). To learn the weights, each weight vector (the $\mathbf{w}_r^o$ and the $\mathbf{w}_j^o$, for $o \in \mathcal{O}$, $j = 1, \ldots, d$) was paired with an eligibility trace vector, $\mathbf{e}_r^o$ or $\mathbf{e}_j^o$. All these vectors were initialized to zero. The agent wandered throughout the two rooms following the equi-probable random policy for 50,000 time steps. On each transition $(S_t, A_t, R_{t+1}, S_{t+1})$ for which S_t was non-terminal, for each $o \in \mathcal{O}$, we did:

$$\begin{aligned}
&\delta \leftarrow \delta\big(R_{t+1}, 0, \hat{r}(\mathbf{x}_t, o), \hat{r}(\mathbf{x}_{t+1}, o), \beta^o(S_{t+1})\big)\\
&\mathsf{UWT}\big(\mathbf{w}_r^o, \mathbf{e}_r^o, \nabla\hat{r}(\mathbf{x}_t, o), \alpha_r\delta, \rho_t^o, \gamma\lambda\big(1{-}\beta^o(S_{t+1})\big)\big)\\
&\text{and, for each } j = 1, \ldots, d\text{:}\\
&\quad \delta \leftarrow \delta\big(0, x_{j,t+1}, \hat{n}_j(\mathbf{x}_t, o), \hat{n}_j(\mathbf{x}_{t+1}, o), \beta^o(S_{t+1})\big)\\
&\quad \mathsf{UWT}\big(\mathbf{w}_j^o, \mathbf{e}_j^o, \nabla\hat{n}_j(\mathbf{x}_t, o), \alpha_p\delta, \rho_t^o, \gamma\lambda\big(1{-}\beta^o(S_{t+1})\big)\big),
\end{aligned} \tag{17}$$

where $x_{j,t+1}$ denotes the jth component of $\mathbf{x}_{t+1}$ and $\hat{n}_j$ denotes the jth component of the vector returned by $\hat{\mathbf{n}}$. This way of using TD(λ) algorithms to learn a temporally abstract model of the world was introduced by Sutton [32,33, Section 17.2]. The parameters were $\alpha_r = \alpha_p = 0.1$ and $\lambda = 0$.

The procedure described above allows us to efficiently learn both transition and reward models, as shown in Fig. 3. We also evaluated the impact of planning with imperfect models, using the planning algorithm described in the next section. The results are shown in the inset plot. Planning leads to near-optimal performance after the model has been trained for about 20,000 steps. Further training improves model error but does not significantly improve planning in this environment. The final model, after 50,000 steps, was used for the planning results in Fig. 1.

A linear expectation model is not the most general form of an environmental model, but still may be a good choice. More general would be a world model in which the transition-part is able to produce result states with the correct joint

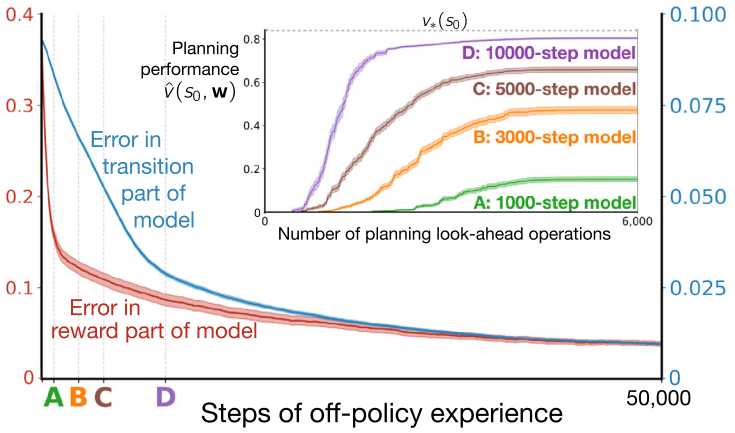


Fig. 3. Model learning experiment. The root-mean-square error in the model of the hallway-attaining option falls toward zero under off-policy training. **Inset**: The models at various times A-D are assessed for their utility in planning; the later, more-accurate models enable faster planning to better policies. All lines are averages over 100 runs.

distribution. These *distribution* models are important theoretically and have been used effectively when the possible distributions can be assumed to be of a special form (e.g., Gaussian), as in the PILCO method [34]. Generally, distribution models are very large objects and would be hopelessly unwieldy for large d (unless the distributions are of particular forms, such as Gaussians). Fortunately, if the value function is linear in the state features, then there is no loss of generality when an expectation model is used in planning [31,30].

The deeper issue is that no model can be complete and accurate, as the world is much larger and more complex than the agent [35]. An expectation model is one strategy for accepting this gracefully.

5. Planning with options

Our planning method approximates *asynchronous value iteration*, a classical operations-research planning algorithm for finite MDPs, extended to options [10]. In the tabular (non-approximate) algorithm, the state-value estimates $V(s)$, for all $s \in \mathcal{S}$, are initialized arbitrarily and then updated one-by-one, in some sequence, by:

$$V(s) \leftarrow \max_{o \in \mathcal{O}(s)} \left[r(s,o) + \sum_{s' \in \mathcal{S}} p(s'|s,o) V(s') \right], \tag{18}$$

where r and p are the reward and state-transition parts of the ideal model of option o (as defined in the previous section), and $\mathcal{O}(s)$ is the set of options considered in state s, which may include all or some of the primitive actions available. If $\mathcal{O}(s)$ is exactly the primitive actions, then $\hat{p}(s'|s,o)$ is exactly the state-transition probabilities, times γ, and the general form (18) reduces to classical value iteration.

In our planning method, *approximate value iteration*, the estimated value function is maintained not as a table $V(s)$, but as a parameterized form $\hat{v}(\mathbf{x}(s), \mathbf{w})$ with feature function $\mathbf{x} : \mathcal{S} \to \mathbb{R}^d$ and weight vector $\mathbf{w} \in \mathbb{R}^d$ with $d \ll |\mathcal{S}|$. Moreover, the planner will iterate over state-feature vectors $\mathbf{x} \in \mathbb{R}^d$ instead of environmental states $s \in \mathcal{S}$. In this paper we use a linear form $\hat{v}(\mathbf{x}, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}$, which combines favorably with expectation models, but in general any differentiable parameterized form could be used. The weight vector is initialized arbitrarily and then updated, for each state-feature vector $\mathbf{x}$ in some sequence of state-feature vectors, by

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \Big[\max_{o \in \mathcal{O}(\mathbf{x})} \big[\hat{r}(\mathbf{x}, o) + \hat{v}(\hat{\mathbf{n}}(\mathbf{x}, o), \mathbf{w}) \big] - \hat{v}(\mathbf{x}, \mathbf{w}) \Big] \nabla_{\mathbf{w}} \hat{v}(\mathbf{x}, \mathbf{w}), \tag{19}$$

where $\alpha > 0$ is a step-size parameter, and $\hat{r}$ and $\hat{\mathbf{n}}$ are the reward and transition parts of an approximate model as described in the previous section.

The quantity $\hat{r}(\mathbf{x}, o) + \hat{v}(\hat{\mathbf{n}}(\mathbf{x}, o), \mathbf{w})$ in (19) is the called the *backed-up value* of the state represented by $\mathbf{x}$, when projected ahead by the model of option o, using the approximate value function given by the weights $\mathbf{w}$. Computing the backed-up value for a state and option counts as one *planning look-ahead operation* in our result figures (the x-axis of, e.g., Fig. 1). The backed-up value is a target, analogous to the quantity in brackets in (18). Equation (19) is a standard stochastic-gradient-descent update rule toward the backed-up value (ignoring the effect of the update on the target, as is commonly done in reinforcement learning). Note that the sum in (18) over $|\mathcal{S}|$ terms has been replaced in (19) by a single call to $\hat{\mathbf{n}}$ whose complexity is linear in the number of model parameters (e.g., only d^2 for $d \ll |\mathcal{S}|$ in the linear case). As discussed earlier, this can be done without introducing any additional approximation error if the value function is linear in the state-feature vector $\mathbf{x}$ [31].

Approximate value iteration (19) was applied to the two-room gridworld to obtain the results in Fig. 1. We sampled states s randomly from the full state set $\mathcal{S}$, computed their feature vectors $\mathbf{x}(s)$, and then performed (19) on each state in sequence. The weight vector $\mathbf{w}$ was initialized to zero. The models of the actions and options were those learned after 50,000 random steps by off-policy methods as described in the preceding section. The step-size parameter was $\alpha = 1$. Additional planning results with incompletely-learned models are shown inset in Fig. 3.

In this paper our presumption is that the model of the options will normally be accurate and stable, while the approximate value function will not have been previously fully learned and will not be stable (as otherwise no further planning would be necessary). These assumptions are noteworthy and deserve examination. Certainly there are cases where they are appropriate. In Chess, for example, the model of the game's dynamics is known completely, and could have been learned, but the value of almost all states can only be approximated, and the planning problem is so large that it is never completely solved and the state values are never all known exactly.

The primary definition of a useful option is one whose model takes the maximum in (18) or (19) at some state and thus makes a difference in planning. To make the backed-up values large (and thus more likely to take the maximum), we certainly seek options for which $r(s, o)$, the cumulative reward during the option, is large. This is the main point about reward-respecting options. We also want the second term of the backed-up value to be large, which is achieved if the option terminates in states of high approximate value. This is the reason for optimistic bonus weights. We seek options that produce high rewards and that drive the environment to states that, occasionally, have high value.

6. Stopping bonuses matter

In Section 2 we defined reward-respecting subtasks for attaining any feature i with respect to a *bonus weight*, $\overline{w}^i$, that determines a *stopping bonus*, $(\overline{w}^i - w_i)x_i(s)$, for stopping in state s when feature $x_i(s)$ is high. The bonus weight controls the size of the stopping bonus and thereby the tradeoff between attaining feature i and attaining reward, and thus the bonus weight is an important parameter of the subtask. So far we have considered only the case in which $\overline{w}^i \doteq 1$. We now evaluate the impact of different choices of the bonus weight and, in particular, show that very large bonus weights result in subtasks whose solutions closely approximate shortest-path options.

Fig. 4 shows the reward-respecting options learned for the two-room gridworld for various values of $\overline{w}^h \in \{0.1, 1, 10, 100\}$. The $\overline{w}^h \doteq 1$ case is the one we have already seen; its learned option takes a roundabout path that avoids the gray region where -1 rewards are received. For the largest bonus weight, $\overline{w}^h \doteq 100$, on the other hand, the learned option takes shortest paths to the hallway state, while for $\overline{w}^h \doteq 10$ the learned option exhibits intermediate behavior. Note that the policies are stochastic; the arrows in the figure show just the action with the *largest probability* of being selected. The choice of $\overline{w}^h \doteq 0.1$ is still optimistic for this task because at the time these options were learned the estimated value function for the main task was still at its initial value of zero. In this case, in the first room the learned option still avoids the region of negative reward, but in the second room it is more directed towards the goal state and less toward the hallway state than in the other three cases. As expected, small values of $\overline{w}^i$ make attaining feature i less important.

For each of the four options, we continued with the other steps of the STOMP progression. We learned a model of the option, and then applied approximate value iteration to plan with the model. The progress of planning in the four cases was assessed, as before, by the estimated value of the start state (see Fig. 5). The blue ($\overline{w}^h \doteq 1$) and purple ($\overline{w}^h \doteq 100$) lines replicate results from Fig. 1; a reward-respecting subtask can result in much faster planning than a shortest-path subtask. The green line ($\overline{w}^h \doteq 10$) suggests that the advantage of reward-respecting subtasks is robust to the choice of the

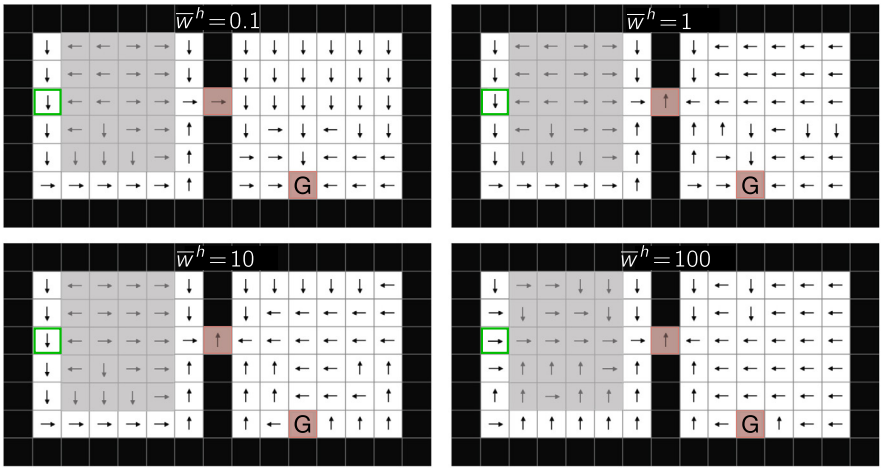


Fig. 4. Reward-respecting options learned for different values of the bonus weight $\overline{w}^h$.

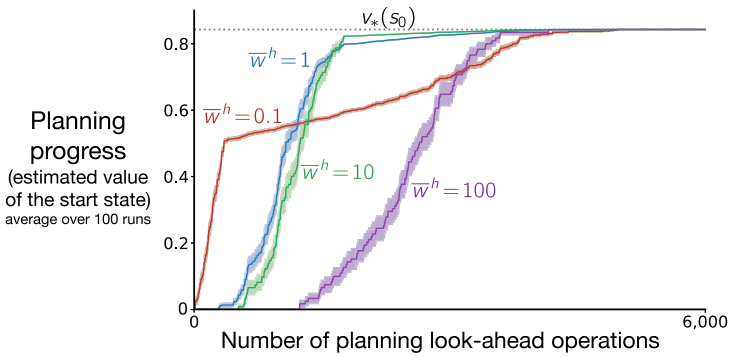


Fig. 5. Impact of the bonus weight $\overline{w}^h$ on the efficiency of planning.

bonus weight. The $\overline{w}^h \doteq 0.1$ case is mixed; initially planning is fastest, presumably because the learned option itself closely approximates the optimal policy, but in the longer term planning is retarded according to this measure.

7. STOMP in a larger, stochastic gridworld

In this section we further test the ideas of reward-respecting subtasks and the STOMP progression by applying them with multiple subtasks and options in a larger problem with stochastic dynamics. We also compare and contrast the reward-respecting approach with that of eigenoptions [36,37], the option-critic architecture [38], and, again, shortest-path options. For each of these four ways of producing options, we learn models of their options and use the models for planning exactly as described earlier in this paper.

The larger problem used in these comparisons is the four-room episodic gridworld depicted in each of the four parts of Fig. 6, with a start state in the upper-left room (highlighted in green) and a terminal goal state in the lower-right room. As in the two-room gridworld, a reward of $+1$ is received on reaching the goal, which ends the episode, and passing through the gray region produces a reward of -1 per step, while all other transitions produce a reward of zero. Again the discount factor is $\gamma = 0.99$, and again there are four actions for moving in each one of the four directions, but now they are *stochastic*, moving in the expected direction only with probability $2/3$, and in one of the other three directions with probability $1/9$. If the direction of movement (after stochasticity) is blocked by a wall, then the agent's location is unchanged.

For the reward-respecting approach, we defined four reward-respecting subtasks of feature attainment, each directed toward attaining the feature for one of the four hallways states H1–H4, and learned close approximations to their optimal options using the algorithms described in Section 3. The bonus weight for all four subtasks was $\overline{w}^i = 1$. The learned options are depicted in Fig. 6—the policy by arrows and the stopping states by red cells. For the most part, the options took the shortest path to the hallway or the goal state, but the negative reward region caused some options to prefer a longer path. In particular, the options often took a roundabout way around the gray region in the lower-left room. The step-size parameter was $\alpha = 0.05$. Fig. 7 shows the progression of learning of the four options.

Eigenoptions are defined as the solutions to subtasks with intrinsic rewards unrelated to the reward of the main task [36, 37]. The intrinsic rewards are constructed from the successor representation as an approximation to the graph Laplacian of the interconnection topology of the environment. We defined four subtasks corresponding to the first four eigenvectors (by largest eigenvalue) of the approximate graph Laplacian. In our notation, the ith subtask was to maximize the GVF (2) with $z(S_K) \doteq 0$ and $C_t \doteq \mathbf{e}_i{}^\top \big(\mathbf{x}(S_t) - \mathbf{x}(S_{t-1})\big)$, where $\mathbf{e}_i$ is the ith eigenvector. Stopping is viewed as a special action [37] whose action value is defined to be zero (and thus need not be learned). In our experiments we learned the value of the other

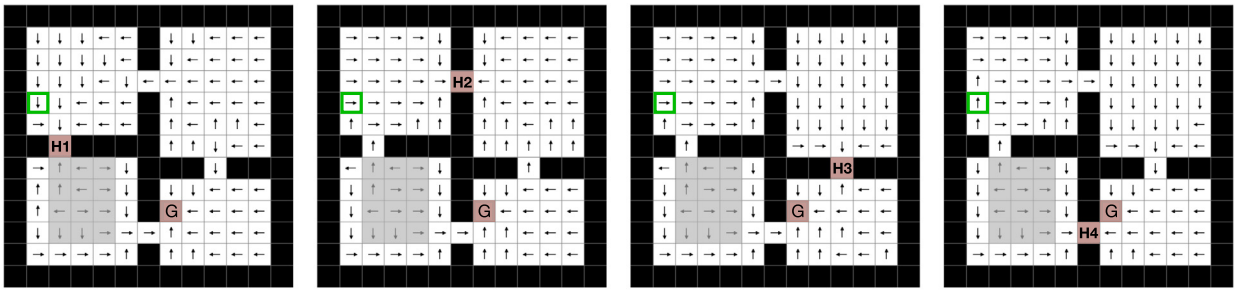


Fig. 6. The four-room gridworld and four reward-respecting options learned for attaining its hallway states H1–H4. The start state is highlighted in green. The arrows show the action most favored by the learned option in each state (the actual policies are stochastic), and the red cells are states in which the option deterministically stopped.

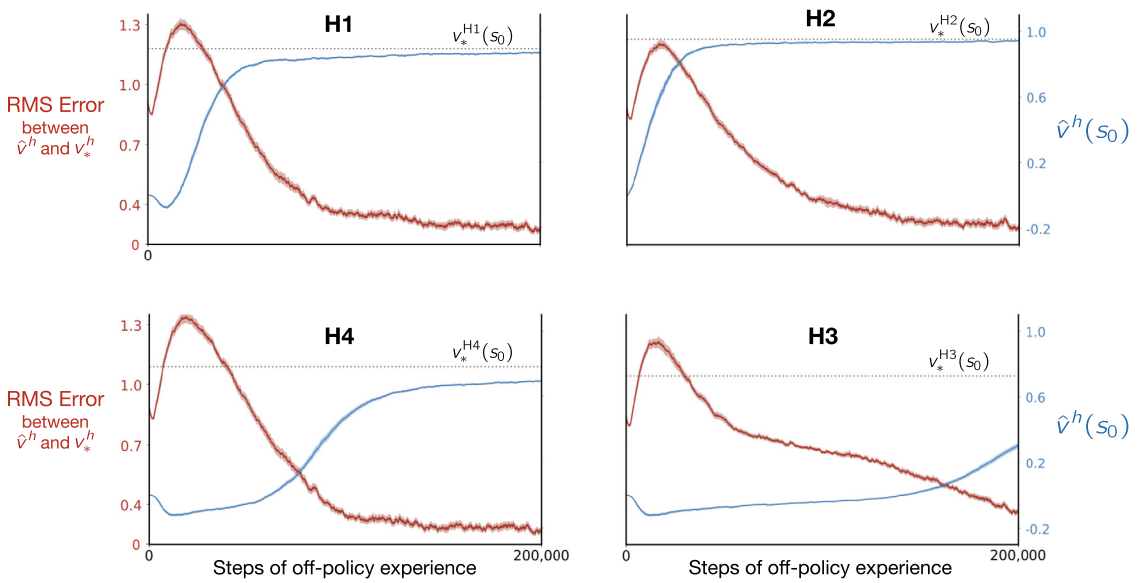


Fig. 7. Off-policy learning of value functions for four reward-respecting options in the four-room gridworld. For each option, the learned value function's difference from the optimal value function falls to near zero (red line, left scale) while increasing towards its optimal value at the start state (blue line, right scale).

actions by one-step Q-learning with a random behavior policy. Finally, the options were defined as greedy with respect to the action values.

The *option-critic architecture* uses an alternative way of discovering options without explicit subtasks or reward signals other than the main-task rewards [38]. The options are initialized randomly and then climb the gradient of a global objective function. We used a re-implementation of the option-critic architecture that is part of DeepMind's software suite. For each run, that software produces a new set of options from experience solving the main task.

Finally, four shortest-path options were approximated for the hallway states of the four-room gridworld in the same way as described earlier for the smaller gridworld. For each hallway, a GVF subtask was created with $C_t \doteq -1$ for all t and $z(s) \doteq 0$ at the hallway state and $-\infty$ otherwise.

Given options created in the above four ways and the options corresponding to the primitive actions, we computed their models and conducted planning as described earlier in this paper. Detail on the progression of model learning for the reward-respecting options is provided in Appendix A. As a measure of the progress in planning, we recorded the approximate value of the start state after each step of AVI (19). Fig. 8 presents these planning results.

Planning with models of reward-respecting options was the fastest, followed by shortest-path options, eigenoptions, and option-critic options. Planning with models of the actions only was initially slower than planning with models of the eigenoptions, but then became faster and ultimately slightly surpassed all the other methods by this performance measure. The asymptotic superiority of the actions-only case may not be significant; remember that these are estimated values and not actual ones. An example of the difference is that the estimated values for many of the methods slightly exceed the largest possible actual value, $v_*(s_0)$, presumably due to maximization bias [39,40] due to the stochastic environment. The actual values can be estimated directly by averaging Monte Carlo returns, but then the data is noisier and it is harder to

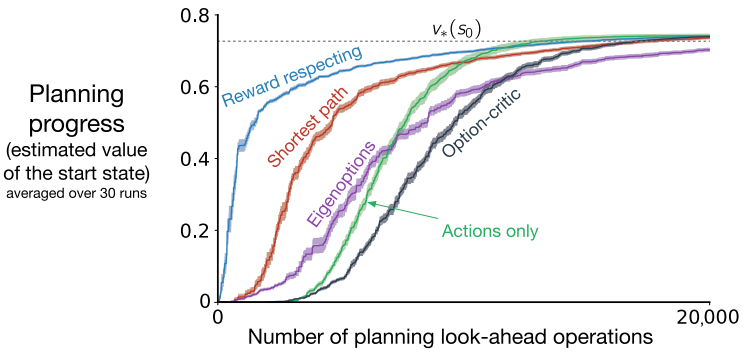


Fig. 8. Progress of planning in the four-room gridworld with the models of options discovered in different ways.

see the differences between methods (see Appendix B). The relatively poor performance of the option-critic approach may be surprising given that it only considers the main-task rewards. See Appendix C for a closer look suggesting that these options were poorly learned in much of the state space.

In the four-room gridworld, reward-respecting options seem more appropriate than alternative subtask formulations when these options are to be used in the STOMP progression. The advantages we saw in the two-rooms gridworld were retained as we extended to the larger problem, to multiple subtasks and options and to stochastic dynamics.

8. Conclusions and future work

In this paper we have further developed four ideas none of which is entirely new. The most important is the idea of subtasks that respect the rewards of the original problem, both in their cumulants and in taking into account the estimated value of the state stopped in. Reward-respecting subtasks are a tiny subset of all possible subtasks, but they may be the most important in model-based reinforcement learning. The second idea is that of the STOMP progression for the development of temporally-abstract cognitive structure, an old idea which we have illustrated in greater detail and generality than in previous works. The third idea, arguably both the most novel and the most unproved, is that of subtasks for attaining state features. These are an appealing subset of reward-respecting subtasks because they achieve something pertaining to a state without requiring that states be completely observable. Achieving a state feature may be a part of effective plans, and feature attainment reduces the option discovery problem to that of deciding what features to maximize. Finally, the fourth idea developed in this paper is that of structuring learning algorithms in terms of a generic TD error (5) and a generic update procedure (UWT).

There are several important areas in which future work could extend that presented here. One natural next step would be to demonstrate all the steps of the STOMP progression operating simultaneously rather than sequentially as we have done them here. This appears straightforward, but doubtless some new issues will arise. Another natural extension would be to general (not one-hot) linear function approximation and multi-step eligibility traces. Although the algorithmic machinery described here applies immediately to both cases, the functionality was not exercised in this paper's experiments. An obvious further extension would be to use deep-learning neural networks for non-linear function approximation. However, a better idea might be to keep linear function approximation and instead extend the feature representation, which would retain the advantageous combination of linear function approximation and expectation models. A related issue is feedback from the later stages of the progression to earlier. For example, the effectiveness of planning as judged by search control methods (yet to be invented) should influence which models are used in planning and which features are used to form feature-attainment options and their models. Really, the feature construction step that precedes STOMP should also ultimately be influenced by the utility of features in all stages of learning and planning. This extension might be called FCSTOMP, or the "Oak" architecture (Options And Knowledge [41]).

Here we have used planning only to improve the value function for the main task. A natural non-obvious extension would be to apply planning to improve the value functions for the subtasks. Extending still further, everything in the form of a general value function should arguably be capable of being planned as well as learned. Because we have formulated the transition model of the environment as a collection of GVFs, it is intriguing to think that the model itself could be planned. That is, the higher-level parts of the transition model could be planned from the lower-level parts. The model of any option, for example, could be formed from planning with models of the primitive actions. This kind of planning—reaching a conclusion about how the world works from lower-level knowledge of how it works—seems deserving of the name *reasoning*.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

The authors thank Joseph Modayil, Martha White, Michael Bowling, and Mark Ring for useful discussions, the anonymous reviewers for helping to clarify the paper's contributions, and particularly Martin Klissarov for assistance analyzing our option-critic results and Francesco Visin for his insightful feedback on an earlier draft. We also thank John Aslanides for the software producing option-critic options.

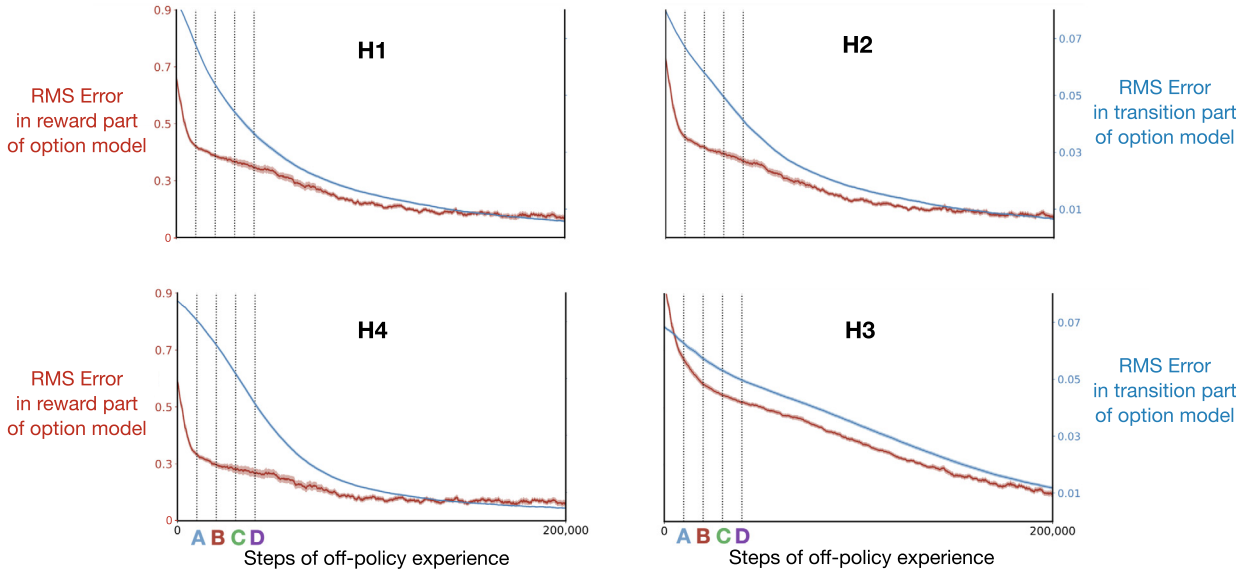


Fig. A.1. Model learning in the four-room gridworld. The time course of learning of the transition parts (blue, right scale) and reward parts (red, left scale) of the models of each of the four options. In all cases the error becomes dramatically smaller, but here the error will never converge to zero because of the stochasticity of the environmental dynamics. All lines were averaged over 30 runs and the shading represents the standard error.

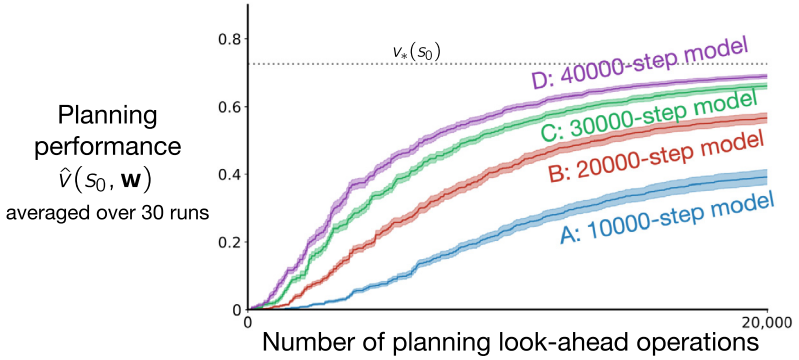


Fig. A.2. Performance of planning for learned, approximate models with different amounts of training, corresponding to the vertical lines in Fig. A.1.

Appendix A. Additional results in the four-room gridworld

In Section 7 we presented results with the STOMP progression using four reward-respecting feature-attainment subtasks. In this appendix we present additional results pertaining to model learning (Fig. A.1) and to planning with the imperfectly learned models (Fig. A.2).

Appendix B. Monte Carlo estimates of planning performance

Throughout the paper we have presented planning results in which the *estimated* value of the start state, $\hat{v}(s_0, \mathbf{w})$, was used as a proxy for the value of the policy $\pi_{\mathbf{w}}$ induced by the state values. However, in some cases $\hat{v}(s_0, \mathbf{w})$ is a poor proxy for $v_{\pi_{\mathbf{w}}}(s_0)$. An example of the discrepancy is that in early stages of planning the value estimates will always be near zero because $\mathbf{w}$ starts near zero and $\hat{v}$ is linear in $\mathbf{w}$, but $v_{\pi_{\mathbf{w}}}(s_0)$ could be positive or negative depending on the environment. Specifically, in our initial illustrative example, the two-room gridworld with the field of -1s, the initial returns will have -1s in them and overall will probably be negative, whereas the estimated values in the initial stages of planning (see Fig. 1) are slightly positive.

In this appendix we redo all the paper's planning results with a more direct Monte-Carlo estimate of $v_{\pi_{\mathbf{w}}}(s_0)$.

First we need a clear specification of $\pi_{\mathbf{w}}$. Let $g(s)$ denote the greedy option in state s given the current state-value weight vector for the main task, $\mathbf{w}$, and the model, $\hat{r}$ and $\hat{\mathbf{n}}$:

$$g(s) \doteq \arg\max_{o \in \mathcal{O}} \big[\hat{r}(\mathbf{x}(s), o) + \hat{v}(\hat{\mathbf{n}}(x(s), o), \mathbf{w})\big], \qquad \forall s \in \mathcal{S}. \tag{B.1}$$

For states s in which $g(s)$ is an *action*, $\pi_{\mathbf{w}}$ was defined to take that action deterministically. For states in which $g(s)$ was an *option*, $\pi_{\mathbf{w}}$ was defined as a stochastic selection from the actions with probabilities given by the soft-max policy for that option (11).

During planning, after each update of $\mathbf{w}$ by (19), one trajectory from start state to termination was generated by following $\pi_{\mathbf{w}}$ with the real environmental dynamics. The return on that trajectory was recorded as a noisy Monte Carlo estimate of $v_{\pi_{\mathbf{w}}}(s_0)$ If the trajectory did not terminate after 1000 steps, then the partial return was used as the estimate of the return ($\gamma^{1000} \approx 0.00004$). To reduce the noise, these estimates were averaged over 30 runs and then averaged over a bin of updates to produce the plots that follow. Figs. B.1 and B.2 used a bin size of 10, and Figs. B.3–B.5 used a bin size of 50. The values for the first few hundred updates in all cases were negative and are not shown in the plots (they are clipped at zero).

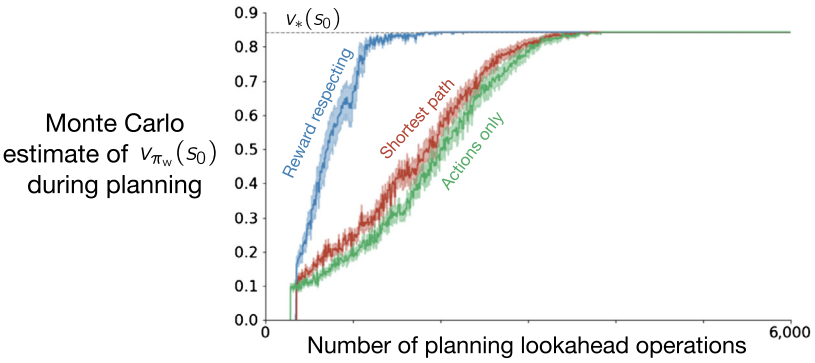


Fig. B.1. Same experiment as Fig. 1, but with Monte Carlo estimate of value.

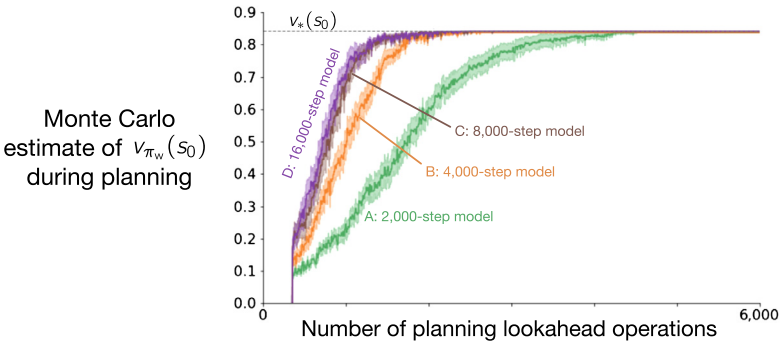


Fig. B.2. Same experiment as inset in Fig. 3, but with Monte Carlo estimate of value.

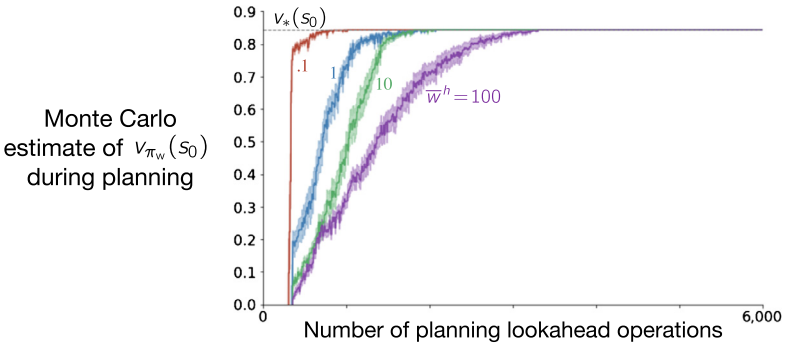


Fig. B.3. Same experiment as Fig. 5, but with Monte Carlo estimate of value.

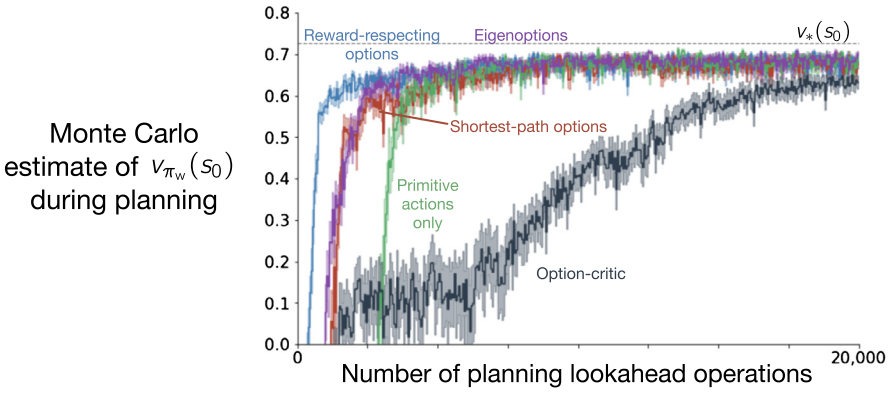


Fig. B.4. Same experiment as Fig. 8, but with Monte Carlo estimate of value.

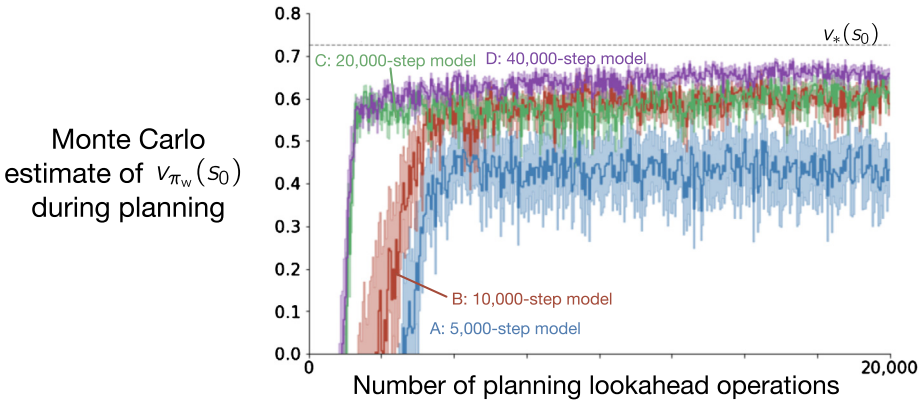


Fig. B.5. Same experiment as Fig. A.2, but with Monte Carlo estimate of value.

Appendix C. Understanding the option-critic's performance

As discussed in the main text, the poor performance of planning with option-critic options (Figs. 8 and B.4) may be surprising at first because these are reward-respecting options. Further inspection highlights the importance of appropriately choosing the state distribution we learn options from, and the distribution we use to sample state-feature vectors when performing AVI.

In the four-room gridworld, as expected, the option-critic learns near-optimal policies that consistently reach the goal state. However, because the option-critic is an *on-policy* method, it does not learn accurate estimates of the option values across the whole state-space, but only across the set of states the options are likely to visit—see representative learned policies in Fig. C.1 and stopping probabilities in Fig. C.2. Because we are selecting state-feature vectors $\mathbf{x}$ in a random sequence when performing AVI, there are several state-feature vectors $\mathbf{x}$ with inaccurate values that hinder planning performance, even though the models we learn are very accurate (see Fig. C.3).

More effective techniques for selecting the states which will be updated, a problem known as *search-control*, could make options learned by the option-critic more effective, but this is currently an open problem.

The options learned by the option-critic, shown in Figs. C.1 and C.2, highlight the importance of defining stopping values that are different from the estimated values of the state the option stops in. Because the option-critic does not do that, it rarely learns four distinct, useful options. Some of the options learned by the option-critic often look like random policies in most of the state space. Notice that different stopping values, such as deliberation cost [42], would not completely address this issue because of the on-policy nature of the underlying method. This discussion highlights the importance of taking into account how the option models will be used in planning.

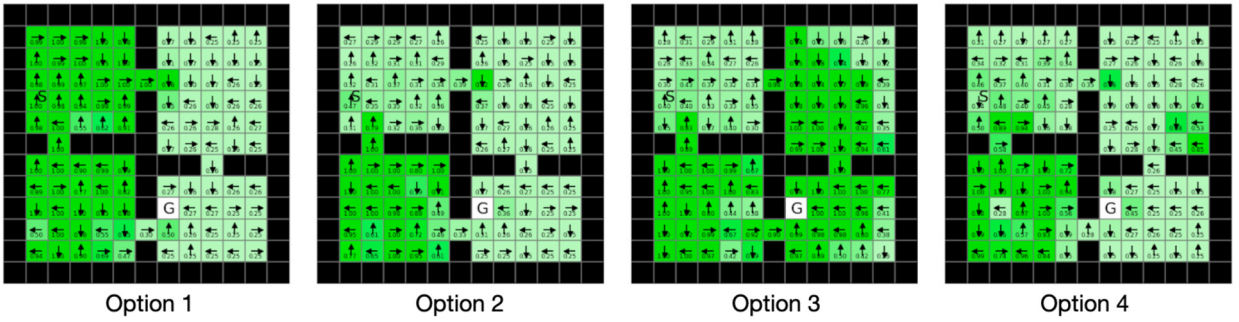


Fig. C.1. Option policies learned by the option-critic method in a representative run. The arrows indicate the greedy action in each state, and the numbers indicate its probability of being taken (the actual policies were stochastic). Darker green indicates states in which the policy is more deterministic.

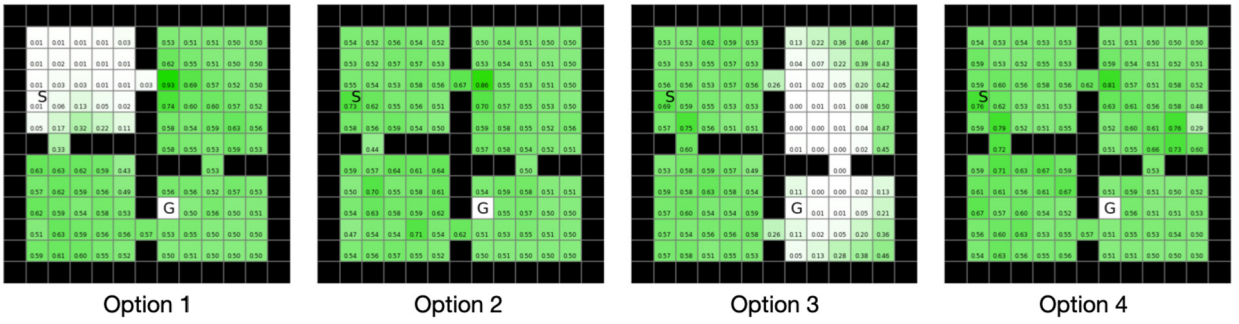


Fig. C.2. Stopping probabilities for the option policies depicted above in Fig. C.1. Darker green indicates states in which the option stops more frequently. In this case, starting from the start state S, Option 1 is unlikely to stop until reaching the hallway state, and then Option 3 is unlikely to stop until reaching the goal state.

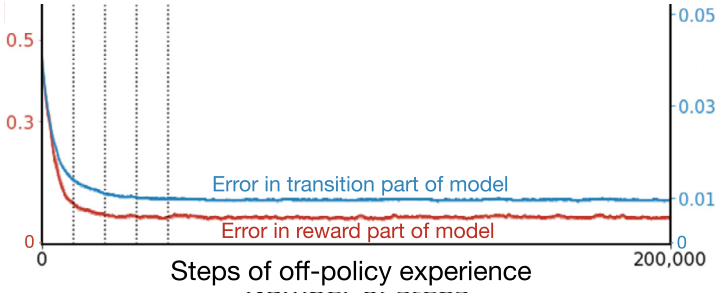


Fig. C.3. Model learning in the four-room gridworld with an option-critic option. The time course of learning of the transition part (blue, right scale) and reward part (red, left scale) of the models of one of the options learned by the option-critic. All lines were averaged over 30 runs.

References

[1] E.D. Sacerdoti, Planning in a hierarchy of abstraction spaces, Artif. Intell. 5 (2) (1974) 115–135.
[2] C.A. Knoblock, Automatically generating abstractions for planning, Artif. Intell. 68 (2) (1994) 243–302.
[3] G. Konidaris, L.P. Kaelbling, T. Lozano-Perez, From skills to symbols: learning symbolic representations for abstract high-level planning, J. Artif. Intell. Res. 61 (2018) 215–289.
[4] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[5] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[6] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.
[7] R.L. Rivest, R.E. Schapire, Diversity-based inference of finite automata, J. ACM 41 (3) (1994) 555–589.
[8] M. Littman, R.S. Sutton, S. Singh, Predictive representations of state, Adv. Neural Inf. Process. Syst. 14 (2002).
[9] H. Jaeger, Observable operator models for discrete stochastic time series, Neural Comput. 12 (6) (2000) 1371–1398.
[10] R.S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning, Artif. Intell. 112 (1–2) (1999) 181–211.
[11] S.P. Singh, A.G. Barto, N. Chentanez, Intrinsically motivated reinforcement learning, in: Advances in Neural Information Processing Systems, 2004.
[12] J. Sorg, S.P. Singh, Linear options, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010.
[13] D. Silver, K. Ciosek, Compositional planning using optimal option models, in: Proceedings of the International Conference on Machine Learning, 2012.

[14] M.B. Ring, Representing knowledge as forecasts (and state as knowledge), arXiv:2112.06336, 2021.
[15] V. Veeriah, Discovery in Reinforcement Learning (Doctoral dissertation), Department of Computer Science and Engineering, University of Michigan, 2022, See Chapter 7.
[16] A. McGovern, A.G. Barto, Automatic discovery of subgoals in reinforcement learning using diverse density, in: Proceedings of the International Conference on Machine Learning, 2001.
[17] Ö. Simsek, A.G. Barto, Using relative novelty to identify useful temporal abstractions in reinforcement learning, in: Proceedings of the International Conference on Machine Learning, 2004.
[18] Ö. Simsek, A.P. Wolfe, A.G. Barto, Identifying useful subgoals in reinforcement learning by local graph partitioning, in: Proceedings of the International Conference on Machine Learning, 2005.
[19] M. Jaderberg, V. Mnih, W.M. Czarnecki, T. Schaul, J.Z. Leibo, D. Silver, K. Kavukcuoglu, Reinforcement learning with unsupervised auxiliary tasks, in: Proceedings of the International Conference on Learning Representations, 2017.
[20] M.C. Machado, A. Barreto, D. Precup, M. Bowling, Temporal abstraction in reinforcement learning with the successor representation, J. Mach. Learn. Res. 24 (2023) 1–69.
[21] A. Solway, C. Diuk, N. Cordova, D. Yee, A.G. Barto, Y. Niv, M. Botvinick, Optimal behavioral hierarchy, PLoS Comput. Biol. 10 (8) (2014).
[22] R.S. Sutton, J. Modayil, M. Delp, T. Degris, P.M. Pilarski, A. White, D. Precup, Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2011.
[23] J. Modayil, A. White, R.S. Sutton, Multi-timescale nexting in a reinforcement learning robot, Adapt. Behav. 22 (2) (2014) 146–160.
[24] A. Baranes, P.-Y. Oudeyer, Active learning of inverse models with intrinsically motivated goal exploration in robots, Robot. Auton. Syst. 61 (1) (2013) 49–73.
[25] B. Eysenbach, A. Gupta, J. Ibarz, S. Levine, Diversity is all you need: learning skills without a reward function, in: Proceedings of the International Conference on Learning Representations, 2019.
[26] R.S. Sutton, Learning to predict by the methods of temporal differences, Mach. Learn. 3 (1988) 9–44 (important erratum p. 377).
[27] R.S. Sutton, Temporal Credit Assignment in Reinforcement Learning (PhD dissertation), Department of Computer Science, University of Massachusetts, Amherst, 1984.
[28] R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy Gradient Methods for Reinforcement Learning with Function Approximation, In Advances in Neural Information Processing Systems, vol. 12, MIT Press, Cambridge, MA, 2000, pp. 1057–1063.
[29] V.R. Konda, J.N. Tsitsiklis, On actor-critic algorithms, SIAM J. Control Optim. 42 (4) (2003) 1143–1166.
[30] K. Kudashkina, Model-based Reinforcement Learning Methods for Developing Intelligent Assistants (PhD dissertation), Department of Engineering, University of Guelph, 2022.
[31] Y. Wan, M. Zaheer, A. White, M. White, R.S. Sutton, Planning with expectation models, in: Proceedings of the International Joint Conference on Artificial Intelligence, 2019.
[32] R.S. Sutton, TD models: modeling the world at a mixture of time scales, in: Proceedings of the International Conference on Machine Learning, 1995.
[33] R.S. Sutton, A. Barto, Reinforcement Learning: An Introduction, 2nd ed., MIT Press, 2018.
[34] M.P. Deisenroth, C.E. Rasmussen, PILCO: a model-based and data-efficient approach to policy search, in: Proceedings of the International Conference on Machine Learning, 2011.
[35] K. Javed, R.S. Sutton, The big world hypothesis and the necessity of online continual learning in big worlds (in preparation).
[36] M.C. Machado, M.G. Bellemare, M. Bowling, A Laplacian framework for option discovery in reinforcement learning, in: Proceedings of the International Conference on Machine Learning, 2017.
[37] M.C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro, M. Campbell, Eigenoption discovery through the deep successor representation, in: Proceedings of the International Conference on Learning Representations, 2018.
[38] P.-L. Bacon, J. Harb, D. Precup, The option-critic architecture, in: Proceedings of the Association for Advancement of Artificial Intelligence, 2017.
[39] H. van Hasselt, Double Q-learning, in: Advances in Neural Information Processing Systems 23, Curran Associates, Inc., 2010, pp. 2613–2621.
[40] H. van Hasselt, Insights in Reinforcement Learning: Formal Analysis and Empirical Evaluation of Temporal-Difference Learning, SIKS dissertation series number 2011-04 2011.
[41] R.S. Sutton, M. Bowling, P.M. Pilarski, The Alberta plan for AI research, arXiv:2208.11173, 2022.
[42] J. Harb, P.-L. Bacon, M. Klissarov, D. Precup, When waiting is not an option: learning options with a deliberation cost, in: Proceedings of the Association for Advancement of Artificial Intelligence, 2018.