Learning Portable Representations for High-Level Planning

Steven James [1] Benjamin Rosman [1] George Konidaris [2]

Abstract

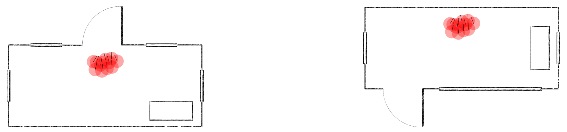
We present a framework for autonomously learning a portable representation that describes a collection of low-level continuous environments. We show that these abstract representations can be learned in a task-independent egocentric space *specific to the agent* that, when grounded with problem-specific information, are provably sufficient for planning. We demonstrate transfer in two different domains, where an agent learns a portable, task-independent symbolic vocabulary, as well as operators expressed in that vocabulary, and then learns to instantiate those operators on a per-task basis. This reduces the number of samples required to learn a representation of a new task.

1. Introduction

A major goal of artificial intelligence is creating agents capable of acting effectively in a variety of complex environments. Robots, in particular, face the difficult task of generating behaviour while sensing and acting in high-dimensional and continuous spaces. Decision-making at this level is typically infeasible—the robot's innate action space involves directly actuating motors at a high frequency, but it would take thousands of such actuations to accomplish most useful goals. Similarly, sensors provide high-dimensional signals that are often continuous and noisy. Hierarchical reinforcement learning (Barto & Mahadevan, 2003) tackles this problem by abstracting away the low-level action space using higher-level *skills*, which can accelerate learning and planning. Skills alleviate the problem of reasoning over low-level actions, but the state space remains unchanged; efficient planning may therefore also require state space abstraction.

One approach is to build a state abstraction of the environment that supports planning. Such representations can then be used as input to task-level planners, which plan using far more compact abstract state descriptors. This mitigates the issue of *reward sparsity* and admits solutions to long-horizon tasks, but raises the question of how to build the appropriate abstract representation of a problem. This is often resolved manually, requiring substantial effort and expertise on the part of the human designer.

Fortunately, recent work demonstrates how to learn a provably sound symbolic representation autonomously, given only the data obtained by executing the high-level actions available to the agent (Konidaris et al., 2018). A major shortcoming of that framework is the lack of generalisability—the learned symbols are grounded in the current task, so an agent must relearn the appropriate representation for each new task it encounters (see Figure 1). This is a data- and computation-intensive procedure involving clustering, probabilistic multi-class classification, and density estimation in high-dimensional spaces, and requires repeated execution of actions within the environment.



(a) The distribution over positions from where the agent is able to interact with the door.

(b) In the new task, the learned distribution is no longer useful since the door's location has changed.

Figure 1: An illustration of the shortcomings of learning task-specific state abstractions (Konidaris et al., 2018). (a) An agent (represented by a red circle) learns a distribution over states (x, y, θ tuples, describing its position in a room) in which it can interact with a door. (b) However, this distribution cannot be reused in a new room with a different layout.

The contribution of this work is twofold. First, we introduce a framework for deriving a symbolic abstraction over an egocentric state space (Agre & Chapman, 1987; Guazzelli et al., 1998; Finney et al., 2002; Konidaris et al., 2012).[1] Because such state spaces are relative to the agent, they provide a

[1] School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, South Africa [2] Department of Computer Science, Brown University, Providence RI 02912, USA. Correspondence to: Steven James <steven.james@wits.ac.za>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

[1] Egocentric state spaces have also been adopted by recent

suitable avenue for representation transfer. However, these abstractions are necessarily non-Markov, and so are insufficient for planning. Our second contribution is thus to prove that the addition of very particular problem-specific information (learned autonomously from the task) to the portable abstractions results in a representation that *is* sufficient for planning. This combination of portable abstractions and task-specific information results in lifted action operators that are transferable across tasks, but which have parameters that must be instantiated on a per-task basis.

We describe our framework using a simple toy domain, and then demonstrate successful transfer in two domains. Our results show that an agent is able to learn abstractions that generalise to tasks with different dynamics, reducing the experience required to learn a representation of a new task.

2. Background

We assume that the tasks faced by an agent can be modelled as a semi-Markov decision process (SMDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S} \subseteq \mathbb{R}^n$ is the n-dimensional continuous state space and $\mathcal{O}(s)$ is the set of temporally-extended actions known as *options* available to the agent at state s. The reward function $\mathcal{R}(s, o, \tau, s')$ specifies the feedback the agent receives from the environment when it executes option o from state s and arrives in state s' after τ steps. $\mathcal{T}$ describes the dynamics of the environment, specifying the probability of arriving in state s' after option o is executed from s for τ timesteps: $\mathcal{T}^o_{ss'} = \Pr(s', \tau \mid s, o)$. An option o is defined by the tuple $\langle I_o, \pi_o, \beta_o \rangle$, where $I_o = \{s \mid o \in \mathcal{O}(s)\}$ is the *initiation set* that specifies the states in which the option can be executed, π_o is the *option policy* which specifies the action to execute, and β_o is the *termination condition*, where $\beta_o(s)$ is the probability of option o halting in state s.

2.1. Portable Skills

The primary goal of *transfer learning* (Taylor & Stone, 2009) is to create an agent capable of leveraging knowledge gained in one task to improve its performance in a different but related task. We are interested in a collection of tasks, modelled by a family of SMDPs.

We first consider the most basic definition of an agent, which is anything that can perceive its environment through sensors, and act upon it with effectors (Russell & Norvig, 2009). In practice, a human designer will usually build upon the observations produced by the agent's sensors to construct the Markov state space for the problem at hand, while discarding unnecessary sensor information. Instead we will seek to effect transfer by using both the agent's sensor information—which is typically egocentric, since the agent carries its own sensors—in addition to the Markov state space.

We assume that tasks are related because they are faced by the same agent (Konidaris et al., 2012). For example, consider a robot equipped with various sensors that is required to perform a number of as yet unspecified tasks. The only aspect that remains constant across all these tasks is the presence of the robot, and more importantly its sensors, which map the state space $\mathcal{S}$ to a portable, lossy, and egocentric observation space $\mathcal{D}$. We define an observation function $\phi : \mathcal{S} \to \mathcal{D}$ that maps states to observations and depends on the sensors available to the agent. We assume the sensors may be noisy, but that the noise has mean 0 in expectation, so that if $s, t \in \mathcal{S}$, then $s = t \implies \mathbb{E}[\phi(s)] = \mathbb{E}[\phi(t)]$. To differentiate, we refer to $\mathcal{S}$ as *problem space* (Konidaris & Barto, 2007).

Augmenting an SMDP with this egocentric data produces the tuple $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{O}_i, \mathcal{T}_i, \mathcal{R}_i, \mathcal{D} \rangle$ for each task i, where the egocentric observation space $\mathcal{D}$ remains constant across all tasks. We can use $\mathcal{D}$ to define portable options, whose option policies, initiation sets and termination conditions are all defined egocentrically. Because $\mathcal{D}$ remains constant regardless of the underlying SMDP, these options can be transferred across tasks (Konidaris & Barto, 2007).

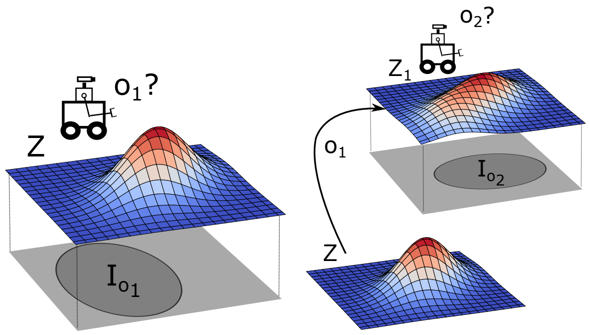
2.2. Abstract Representations

We wish to learn an abstract representation to facilitate planning. A *probabilistic plan* $p_Z = \{o_1, \ldots, o_n\}$ is defined to be the sequence of options to be executed, starting from some state drawn from distribution Z. It is useful to introduce the notion of a *goal option*, which can only be executed when the agent has reached its goal. Appending this option to a plan means that the probability of successfully executing a plan is equivalent to the probability of reaching some goal.

A representation suitable for planning must allow us to calculate the probability of a given plan successfully executing to completion. As a plan is simply a chain of options, it is therefore necessary (and sufficient) to learn when an option can be executed, as well as the outcome of doing so (Konidaris et al., 2018). This corresponds to learning the *precondition* $\text{Pre}(o) = \Pr(s \in I_o)$, which expresses the probability that option o can be executed at state $s \in \mathcal{S}$, and the *image* $\text{Im}(Z, o)$, which represents the distribution of states an agent may find itself in after executing o from states drawn from distribution Z. Figure 2 illustrates how the precondition and image are used to calculate the probability of executing a two-step plan.

In general, we cannot model the image for an arbitrary option; however, we can do so for a subclass known as

reinforcement learning frameworks, such as *VizDoom* (Kempka et al., 2016), *Minecraft* (Johnson et al., 2016) and *Deepmind Lab* (Beattie et al., 2016).



(a) The agent begins at distribution Z, and must determine the probability with which it can execute the first option o_1.

(b) The agent estimates the effect of executing o_1, given by Z_1. It must then determine the probability of executing o_2 from Z_1.

Figure 2: (a–b) An agent attempting to calculate the probability of executing the plan $p_Z = \{o_1, o_2\}$, which requires knowledge of the conditions under which o_1 and o_2 can be executed, as well as the effect of executing o_1 (Konidaris et al., 2018).

subgoal options (Precup, 2000), whose terminating states are independent of their starting states (Konidaris et al., 2018). That is, for any subgoal option o, $\Pr(s' \mid s, o) = \Pr(s' \mid o)$. We can thus substitute the option's image for its *effect*: $\text{Eff}(o) = \text{Im}(Z, o)\ \forall Z$.

Subgoal options are not overly restrictive, since they refer to options that drive an agent to some set of states with high reliability, which is a common occurrence in robotics owing to the use of closed-loop controllers. Nonetheless, it is likely an option may not be subgoal. It is often possible, however, to *partition* an option's initiation set into a finite number of subsets, so that it possesses the subgoal property when initiated from each of the individual subsets. That is, we partition an option's start states into classes $\mathcal{C}$ such that $\Pr(s' \mid s, c) \approx \Pr(s' \mid c), c \in \mathcal{C}$ (see Figure 3). In practice, the agent achieves this by clustering state transition samples based on effect states, and assigning each cluster to a partition. For each pair of partitions we then check whether their start states overlap significantly, and if so merge them, which accounts for probabilistic effects (Andersen & Konidaris, 2017; Konidaris et al., 2018; Ames et al., 2018).

2.2.1. Abstract Options

We may also assume that the option is *abstract*—that is, it obeys the frame and action outcomes assumptions (Pasula et al., 2004). Thus for each abstract option, we can decompose the state into two sets of variables $s = [a, b]$ such that executing the option results in state $s' = [a, b']$, where a is the subset of variables that remain unchanged. We refer to the variables that are modified by an option as its *mask*. Whereas subgoal options induce an abstract MDP or planning graph, *abstract* subgoal options allow us to construct a propositional model corresponding to a *factored* abstract MDP or STRIPS representation (Fikes & Nilsson, 1971).

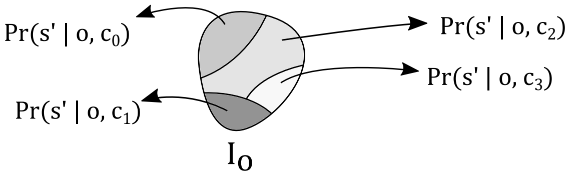


Figure 3: Option o is not subgoal, but we can partition the initiation set I_o into 4 subsets $c_0, \ldots, c_3$, such that the option is subgoal when initiated from each of these sets.

In order to construct a symbolic representation, we first partition options to ensure they are (abstract) subgoal options. We then estimate the precondition and effect for each of the partitioned options. Estimating the precondition is a classification problem, while the effect is one of density estimation. Finally, for all valid combinations of effect distributions, we construct a forward model by computing the probability that states drawn from their grounding lies within the learned precondition of each option, discarding operators with low probability of occurring. This procedure is illustrated by Figure 4, but for more detail see Konidaris et al. (2018).

3. Learning Portable Representations

To aid in explanation, we make use of a simple continuous task where a robot navigates the building illustrated in Figure 5a. The problem space is the xy-coordinates of the robot, while we use an egocentric view of the environment (nearby walls and windows) around the agent for transfer. These observations are illustrated in Figures 5b–d.

The robot is equipped with options to move between different regions of the building, halting when it reaches the start or end of a corridor. It possesses the following four options: (a) `Clockwise` and `Anticlockwise`, which move the agent in a clockwise or anticlockwise direction respectively, (b) `Outward`, which moves the agent down a corridor away from the centre of the building, and (c) `Inward`, which moves it towards the centre.

We could adopt the approach of Konidaris et al. (2018) to learn an abstract representation using transition data in $\mathcal{S}$. However, that procedure generates symbols that are distributions over xy-coordinates, and are thus tied directly to the particular problem configuration. If we were to simply translate the environment along the plane, the xy-coordinates would be completely different, and our learned representation would be useless.

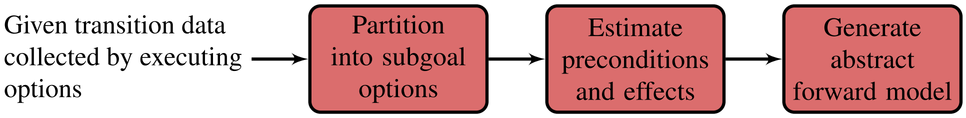

Given transition data collected by executing options
Partition into subgoal options
Estimate preconditions and effects
Generate abstract forward model

Figure 4: The process of learning symbolic representations (Konidaris et al., 2018). The abstract model can take various forms, such as a factored MDP or a PPDDL description (Younes & Littman, 2004). The shaded nodes are learned from data by the agent autonomously.

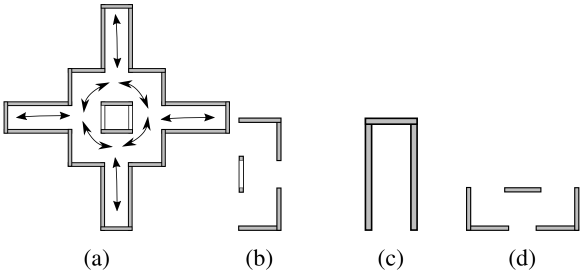


Figure 5: (a) A continuous navigation task where an agent navigates between different regions in xy-space. Walls are represented by grey lines, while the two white bars represent windows. Arrows describe the agent's options. (b–d) Local egocentric observations. We name these window-junction, dead-end and wall-junction respectively.

To overcome that limitation, we propose learning a symbolic representation over $\mathcal{D}$ instead of $\mathcal{S}$. Transfer can be achieved in this manner (provided ϕ is *non-injective*[2]) because $\mathcal{D}$ remains consistent across SMDPs, even if the state space or transition function do not.

Given only data produced by sensors, the agent proceeds to learn an abstract representation, and identifies three portable symbols. These symbols are exactly those illustrated by Figure 5. The learned operators are listed in Table 1, where it is clear that naïvely considering egocentric observations alone is insufficient for planning purposes: the agent does not possess an option with probabilistic outcomes, but the Inward option appears to have probabilistic effects due to aliasing.

A further challenge appears when the goal of the task is defined in $\mathcal{S}$. If we have goal $\mathcal{G} \subseteq \mathcal{S}$, then given information from $\mathcal{D}$, *we cannot determine whether we have achieved the goal*. This follows naturally from the property that ϕ is non-injective: consider two states $s, t \in \mathcal{S}$ such that $s \neq t$ and $\phi(s) = \phi(t) = d \in \mathcal{D}$. If $s \in \mathcal{G}$, but $t \notin \mathcal{G}$, then knowledge of d alone is insufficient to determine whether we have entered a state in $\mathcal{G}$. We therefore require additional information to disambiguate such situations, allowing us to map from egocentric observations back into $\mathcal{S}$.

We can accomplish this by partitioning our portable options based on their effects in $\mathcal{S}$, resulting in options that are subgoal in both $\mathcal{D}$ and $\mathcal{S}$. Recall that options are partitioned to ensure the subgoal property holds, and so each partition defines its own unique image distribution. If we label each problem-space partition, then each label refers to a unique distribution in $\mathcal{S}$ and is sufficient for disambiguating our egocentric symbols. Figure 6 annotates the domain with labels according to their problem-space partitions. Note that the partition numbers are completely arbitrary.

Generating agent-space symbols results in lifted symbols such as dead-end(X), where dead-end is the name for a distribution over $\mathcal{D}$, and X is a partition number that must be determined on a per-task basis. Note that the *only* time problem-specific information is required is to determine the values of X, which grounds the portable symbol in the current task.

[2] We require that ϕ be non-injective, since the ability to transfer depends on two distinct states in $\mathcal{S}$ being identical in $\mathcal{D}$. If this is not the case, we can do no better than learning using only $\mathcal{S}$.

Table 1: A list of the six subgoal options, specifying their preconditions and effects in agent space only.

Option	**Precondition**	**Effect**
Clockwise1	wall-junction	window-junction
Clockwise2	window-junction	wall-junction
Anticlockwise1	wall-junction	window-junction
Anticlockwise2	window-junction	wall-junction
Outward	wall-junction $\vee$ window-junction	dead-end
Inward	dead-end	$\begin{cases} \texttt{window-junction} \text{ w.p. } 0.5 \\ \texttt{wall-junction} \text{ w.p. } 0.5 \end{cases}$

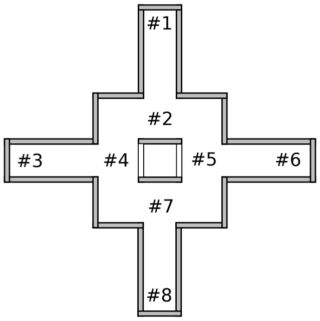


Figure 6: Each number refers to the initiation set of an option partitioned in problem space. For readability, we merge identical partitions. For instance, #2 refers to the initiation sets of a single problem space partition of `Outward`, `Clockwise` and `Anticlockwise`.

The following result shows that the combination of agent-space symbols with problem-space partition numbers provides a sufficient symbolic vocabulary for planning. (The proof is given in the supplementary material.)

Theorem 1 *The ability to represent the preconditions and image of each option in agent space, together with the partitioning in $\mathcal{S}$, is sufficient for determining the probability of being able to execute any probabilistic plan p from starting distribution Z.*

4. Generating a Task-Specific Model

Our approach can be viewed as a two-step process. The first phase learns portable symbolic operators using egocentric transition data from possibly several tasks, while the second phase uses problem-space transitions from the current task to partition options in $\mathcal{S}$. The partition labels are then used as parameters to ground the previously-learned portable operators in the current task. We use these labels to learn *linking functions* that connect precondition and effect parameters. For example, when the parameter of `Anticlockwise2` is #5, then its effect should take parameter #2. Figure 7 illustrates this grounding process.

These linking functions are learned by simply executing options and recording the start and end partition labels of each transition. We use a simple count-based approach that, for each option, records the fraction of transitions from one partition label to another. A more precise description of this approach is specified in the supplementary material.

A combination of portable operators and partition numbers reduces planning to a search over the space $\Sigma \times \mathbb{N}$, where Σ is the set of generated symbols. Alternatively (and equivalently), we can generate either a factored MDP or a PPDDL representation (Younes & Littman, 2004). To generate the latter, we use a *function* named `partition` to store the current partition number and specify predicates for the three symbols derived in the previous sections: `window-junction`, `dead-end` and `wall-junction`. The full domain description is provided in the supplementary material.

5. Inter-Task Transfer

In our example, it is not clear why one would want to learn portable symbolic representations—we perform symbol acquisition in $\mathcal{D}$ *and* instantiate the operators for the given task, which requires more computation than directly learning symbols in $\mathcal{S}$. We now demonstrate the advantage of doing so by learning portable models of two different domains, both of which feature continuous state spaces and probabilistic transition dynamics.

5.1. Rod-and-Block

We construct a domain we term *Rod-and-Block* in which a rod is constrained to move along a track. The rod can be rotated into an upward or downward position, and a number of blocks are arranged to impede the rod's movement. Two walls are also placed at either end of the track. One such task configuration is illustrated by Figure 8.

The problem space consists of the rod's angle and its x position along the track. Egocentric observations return the types of objects that are in close proximity to the rod, as well as its angle. In Figure 8, for example, there is a block to the left of the rod, which has an angle of π. The high-level options given to the agent are `GoLeft`, `GoRight`, `RotateUp`, and `RotateDown`. The first two translate the rod along the rail until it encounters a block or wall while maintaining its angle. The remaining options rotate the rod into an upward or downward position, provided it does not collide with an object. These rotations can be done in both a clockwise and anti-clockwise direction.

We learn a symbolic representation using egocentric transitions only, following the same procedure as prior work (Konidaris et al., 2018): first, we collect agent-space transitions by interacting with the environment. We then partition the options in agent space using the DBSCAN clustering algorithm (Ester et al., 1996) so that the subgoal property approximately holds. This produces partitioned agent-space options. Finally, we estimate the options' preconditions using a support vector machine with Platt scaling (Cortes & Vapnik, 1995; Platt, 1999), and use kernel density estimation (Rosenblatt, 1956; Parzen, 1962) to model effect distributions.[3]

The above procedure results in portable high-level operators, one of which is illustrated by Figure 9. These operators

[3] We provide more details in the supplementary material.

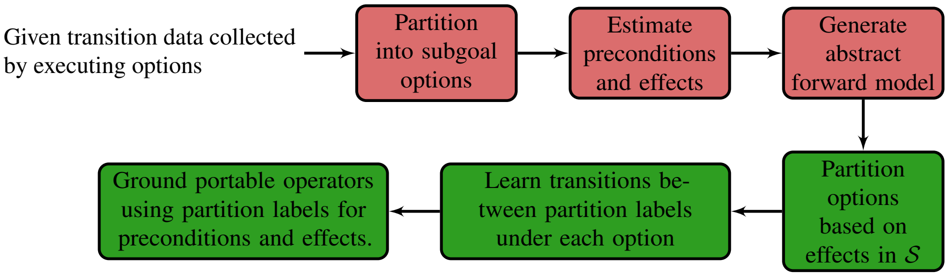

Given transition data collected by executing options
Partition into subgoal options
Estimate preconditions and effects
Generate abstract forward model
Partition options based on effects in $\mathcal{S}$
Learn transitions between partition labels under each option
Ground portable operators using partition labels for preconditions and effects.

Figure 7: The full process of learning portable representations from data. In nodes coloured red, the agent learns representations using egocentric data from all previously encountered tasks, while the green nodes denote where the agent learns using problem-space data from the current task only.

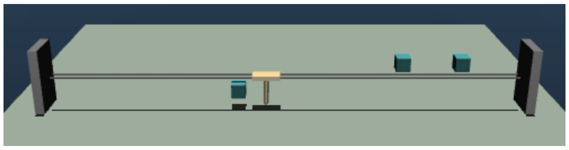


Figure 8: The *Rod-and-Block* domain. This particular task consists of three obstacles that prevent the rod from moving along the track when the rod is in either the upward or downward position. Different tasks are characterised by different block placements.

can be reused for new tasks or configurations of the *Rod-and-Block* domain—we need not relearn them when we encounter a new task, though we can always use data from a new task to improve them. More portable operators are given in the supplementary material.

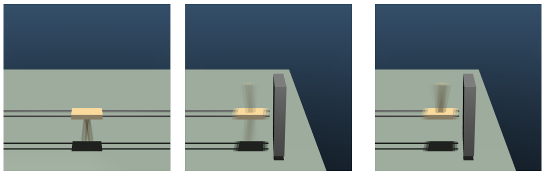
Once we have learned sufficiently accurate portable operators, they need only be instantiated for the given task by learning the linking between partitions. This requires far fewer samples than classification and density estimation over the state space $\mathcal{S}$, which is required to learn a task-specific representation.

To illustrate this, we construct a set of ten tasks $\rho_1, \dots, \rho_{10}$ by randomly selecting the number of blocks, and then randomly positioning them along the track. Because tasks have different configurations, constructing a symbolic representation in problem space requires relearning a model of each task from scratch. However, when constructing an egocentric representation, symbols learned in one task can immediately be used in subsequent tasks. We gather k transition samples from each task by executing options uniformly at random, and use these samples to build both task-specific and egocentric (portable) models.

In order to evaluate a model's accuracy, we randomly select 100 goal states for each task, as well as the optimal plans for reaching each from some start state. Each plan consists of two options, and we denote a single plan by the tuple $\langle s_1, o_1, s_2, o_2 \rangle$. Let $\mathcal{M}_k^{\rho_i}$ be the forward model consisting of high-level preconditions and effects constructed for task ρ_i using k samples. We calculate the likelihood of each optimal plan under the model: $\Pr(s_1 \in I_{o_1} \mid \mathcal{M}_k^{\rho_i}) \times \Pr(s' \in I_{o_2} \mid \mathcal{M}_k^{\rho_i})$, where $s' \sim \text{Eff}(o_1)$. We build models using increasing numbers of samples, varying the number of samples in steps of 250, until the likelihood averaged over all plans is greater than some acceptable threshold (we use a value of 0.75), at which point we continue to the next task. The results are given by Figure 11a.

5.2. Treasure Game

We next apply our approach to the *Treasure Game*, where an agent navigates a continuous maze in search of treasure. The domain contains ladders and doors which impede the agent. Some doors can be opened and closed with levers, while others require a key to unlock.



(a) (b)

```
(:action Up Clockwise_1
 :parameters()
 :precondition (and (sym_18) (sym_11))
 :effect (and (sym_12) (not sym_18))
)
```

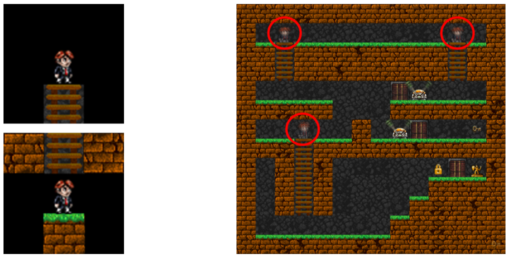
(c)

Figure 9: (a) The precondition for RotateUpClockwise1, which states that in order to execute the option, the rod must be left of a wall facing down. The precondition is a conjunction of these two symbols—the first (sym_18) is a distribution over the rod's angle only, while the second (sym_11) is independent of it. (b) The effect of the option, with the rod adjacent to the wall in an upward position. (c) PDDL description of the above operator, which is used for planning.

The problem space consists of the xy-position of the agent, key and treasure, the angle of the levers (which determines whether a door is open) and the state of the lock.

The egocentric space is a vector of length 9, the elements of which are the type of sprites in each of the nine directions around the agent, plus the "bag" of items collected by the agent. The agent possesses a number of high-level options, such as `GoLeft` and `DownLadder`. More details are given by Konidaris et al. (2018).

We construct a set of ten tasks $\rho_1, \ldots, \rho_{10}$ corresponding to different levels of the *Treasure Game*,[4] and learn portable models. We test their sample efficiency as in Section 5.1. An example of a portable operator, as well as its problem-space partitioning, is given by Figure 10, while the number of samples required to learn a good model of all 10 levels is given by Figure 11b.



(a) (b)

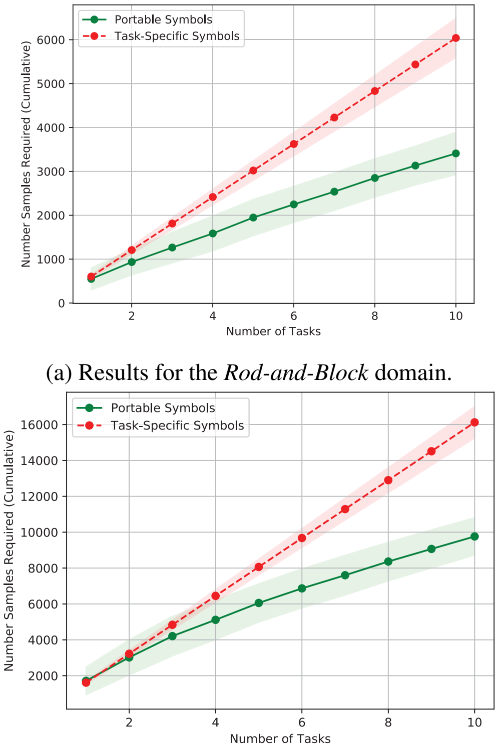
```
(:action DownLadder_1
 :parameters()
 :precondition (and (sym_80))
 :effect (and (sym_622) (not sym_80))
)
```

(c)

Figure 10: (a) The precondition (top) and positive effect (bottom) for the `DownLadder` operator, which states that in order to execute the option, the agent must be standing above the ladder. The option results in the agent standing on the ground below it. The black spaces refer to unchanged low-level state variables. (b) Three problem-space partitions for the `DownLadder` operator. Each of the circled partitions is assigned a unique label and combined with the portable rule in (a) to produce a grounded operator. (c) The PDDL representation of the operator specified in (a).

5.3. Discussion

Naturally, learning problem-space symbols results in a sample complexity that scales linearly with the number of tasks, since we must learn a model for each new task from scratch. Conversely, by learning and reusing portable symbols, we can reduce the number of samples we require as we encounter more tasks, leading to a *sublinear* increase. The agent initially requires about 600 samples to learn a task-specific model of each *Rod-and-Block* configuration, but decreases to roughly 330 after only two tasks. Similarly, 1600 samples are initially needed for each level of the *Treasure Game*, but only 900 after four levels, and about 700 after seven.



(a) Results for the *Rod-and-Block* domain.

(b) Results for the *Treasure Game* domain.

Figure 11: Cumulative number of samples required to learn sufficiently accurate models as a function of the number of tasks encountered. Results are averaged over 100 random permutations of the task order. Standard errors are specified by the shaded areas.

Intuitively, one might expect the number of samples to plateau as the agent observes more tasks. That we do not is as a result of the exploration policy—the agent must observe all relevant partitions at least once, and selecting actions uniformly at random is naturally suboptimal. Nonetheless, we still require far fewer samples to learn the links between partitions than learning a full model from scratch.

In both of our experiments, we construct a set of 10 domain configurations and then test our approach by sampling 100 goals for each, for a total of 1000 tasks per domain. Our model-based approach learns 10 forward models, and then uses them to plan a sequence of actions to achieve each goal. By contrast, a model-free approach would be required

[4] We made no effort to design tasks in a curriculum-like fashion. The levels are given in the supplementary material.

to learn all 1000 policies, since every goal defines another unique SMDP that must be solved. Furthermore, it is unclear how to extend these techniques to deal with tasks whose state space dimensionality differ.

Our approach treats the problem as a two-step procedure: we first learn representations using only $\mathcal{D}$, and then use only $\mathcal{S}$ to ground the representations to our current task. A naïve alternative would be to simply combine $\mathcal{S}$ and $\mathcal{D}$ and learn representations over the combined state space. However, doing so would result in models that are not wholly transferable. For example, in the *Treasure Game*, the agent would learn that to climb down a ladder, it must be standing on top of the ladder *and* at some xy-position. In a new task, the agent would recognise that it is standing on a ladder, but its coordinates would likely be different and so the precondition would not apply.

We depart from most model-based approaches in that we rely on the portable observation space $\mathcal{D}$ for transfer. This raises questions regarding how hard it is to specify $\mathcal{D}$, and the sensitivity of the egocentric observation space to the resulting representations. Fortunately, it is not too hard to provide an egocentric view of the agent: as mentioned, for many real-world problems with embodied agents, this amounts to the agent carrying its own sensors, while for simulated problems (such as those presented here) one can simply centre the input observation on the agent's reference frame. We note, too, that there has been work on autonomously discovering portable observation spaces (Snel & Whiteson, 2010), but this is orthogonal to our work.

Finally, we remark that transfer will naturally depend on, and be sensitive to, the characteristics of $\mathcal{D}$. The question of sensitivity has been extensively studied in the context of learning a single policy (Konidaris et al., 2012, Section 4.3.4), where results indicate that policy learning erodes gradually with the usefulness of $\mathcal{D}$. Practically, this is a concern for learning the option policies, and so we will only remark that if $\mathcal{D}$ is sufficient to learn the options (which we assume has already taken place), then it is sufficient to learn the corresponding representations.

6. Related Work

There has been some work in autonomously learning parameterised representations of skills, particularly in the field of relational reinforcement learning. Finney et al. (2002), Pasula et al. (2004) and Zettlemoyer et al. (2005), for instance, learn operators that transfer across tasks. However, the high-level symbolic vocabulary is given; we show how to learn it. Ames et al. (2018) adopt a similar approach to Konidaris et al. (2018) to learn symbolic representations for parameterised actions. However, the representation learned is fully propositional (even if the actions are not) and cannot be transferred across tasks. Ugur & Piater (2015) are able to discover parameterised symbols for robotic manipulation tasks, but discrete relations between object properties such as width and height are given.

Relocatable action models (Sherstov & Stone, 2005; Leffler et al., 2007) assume states can be aggregated into "types" which determine the transition behaviour. State-independent representations of the outcomes from different types are learned and improve the learning rate in a single task. However, the mapping from lossy observations to states is provided to the agent, since learning this mapping is as hard as learning the full MDP.

There is a large body of literature in the fields of meta-learning and lifelong learning devoted to methods that learn an internal or latent representation that generalises across a distribution of tasks (Jonschkowski & Brock, 2015; Higgins et al., 2017; Kirkpatrick et al., 2017; Finn et al., 2017; de Bruin et al., 2018). When presented with a new task, agents subsequently learn a policy based on its internal representation in a model-free manner. In contrast, our approach learns an explicit model which supports forward planning, and is independent of the task or reward structure.

More recently, Zhang et al. (2018) propose a method for constructing portable representations for planning. However, the mapping to abstract states is provided, and planning is restricted solely to the equivalent of an egocentric space. Similarly, Srinivas et al. (2018) learn a goal-directed latent space in which planning can occur. However, the goal must be known upfront and be expressible in the latent space. We do not compare to either, since both are unsuited to tasks with goals defined in problem space, and neither provides soundness guarantees.

7. Summary

We have introduced a framework for autonomously learning portable representations for planning. Previous work (Konidaris et al., 2018; Ames et al., 2018) has shown how to learn a high-level representation suitable for planning, but these representations are directly tied to the task in which they were learned. Ultimately, this is a fatal flaw—should any of the environments change even slightly, the entire representation would need to be relearned from scratch. Conversely, we demonstrate that an agent is able to learn a portable representation given only data gathered from option execution. We also show that the addition of particular problem-specific information results in a representation that is provably sufficient for learning a sound representation for planning. This allows us to leverage experience in solving new unseen tasks—an important step towards creating adaptable, long-lived agents.

Acknowledgements

This research was supported in part by the National Research Foundation of South Africa (Grant Number 117808), and by NSF CAREER Award 1844960, DARPA under agreement numbers W911NF1820268 and (YFA) D15AP00104, and AFOSR YIP agreement number FA9550-17-1-0124. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NSF, DARPA, or the AFOSR.

References

Agre, P. and Chapman, D. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, volume 87, pp. 286–272, 1987.

Ames, B., Thackston, A., and Konidaris, G. Learning symbolic representations for planning with parameterized skills. In *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018.

Andersen, G. and Konidaris, G. Active exploration for learning symbolic representations. In *Advances in Neural Information Processing Systems*, pp. 5016–5026, 2017.

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

Beattie, C., Leibo, J., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. DeepMind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3): 1394–1401, 2018.

Ester, M., Kriegel, H., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd International Conference on Knowledge Discovery and Data Mining*, volume 96, pp. 226–231, 1996.

Fikes, R. and Nilsson, N. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135, 2017.

Finney, S., Gardiol, N., Kaelbling, L., and Oates, T. The thing that we tried didn't work very well: deictic representation in reinforcement learning. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pp. 154–161, 2002.

Guazzelli, A., Bota, M., Corbacho, F., and Arbib, M. Affordances, motivations, and the world graph theory. *Adaptive Behavior*, 6(3-4):435–471, 1998.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. DARLA: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pp. 1480–1490, 2017.

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. The Malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 4246–4247, 2016.

Jonschkowski, R. and Brock, O. Learning state representations with robotic priors. *Autonomous Robots*, 39(3): 407–428, 2015.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. Vizdoom: A Doom-based AI research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games*, pp. 1–8. IEEE, 2016.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, pp. 201611835, 2017.

Konidaris, G. and Barto, A. Building portable options: skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, volume 7, pp. 895–900, 2007.

Konidaris, G., Scheidwasser, I., and Barto, A. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(May):1333–1371, 2012.

Konidaris, G., Kaelbling, L., and Lozano-Pérez, T. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61(January):215–289, 2018.

Leffler, B. R., Littman, M. L., and Edmunds, T. Efficient reinforcement learning with relocatable action models. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, volume 7, pp. 572–577, 2007.

Parzen, E. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3): 1065–1076, 1962.

Pasula, H., Zettlemoyer, L., and Kaelbling, L. Learning probabilistic relational planning rules. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 73–81, 2004.

Platt, J. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 10(3):61–74, 1999.

Precup, D. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.

Rosenblatt, N. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pp. 832–837, 1956.

Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

Sherstov, A. and Stone, R. Improving action selection in MDPs via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, volume 5, pp. 1024–1029, 2005.

Snel, M. and Whiteson, S. Multi-task evolutionary shaping without pre-specified representations. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pp. 1031–1038. ACM, 2010.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pp. 4732–4741, 2018.

Taylor, M. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Ugur, E. and Piater, J. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation*, pp. 2627–2633, 2015.

Younes, H. and Littman, M. PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, 2004.

Zettlemoyer, L., Pasula, H., and Kaelbling, L. Learning planning rules in noisy stochastic worlds. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pp. 911–918, 2005.

Zhang, A., Lerer, A., Sukhbaatar, S., Fergus, R., and Szlam, A. Composable planning with attributes. In *International Conference on Machine Learning*, pp. 5842–5851, 2018.