Symbol acquisition for task-level planning

Symbol Acquisition for Task-Level Planning

George Konidaris, Leslie Pack Kaelbling and **Tomas Lozano-Perez**
MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Cambridge MA 02139 USA
{gdk, lpk, tlp}@csail.mit.edu

Abstract

We consider the problem of how to plan efficiently in low-level, continuous state spaces with temporally abstract actions (or skills), by constructing abstract representations of the problem suitable for task-level planning. The central question this effort poses is which abstract representations are required to express and evaluate plans composed of sequences of skills. We show that classifiers can be used as a symbolic representation system, and that the ability to represent the preconditions and effects of an agent's skills is both necessary and sufficient for task-level planning. The resulting representations allow a reinforcement learning agent to acquire a symbolic representation appropriate for planning from experience.

Introduction

A core challenge of AI is designing intelligent agents that can perform high-level learning and planning, while ultimately affecting control using low-level sensors and actuators. Hierarchical reinforcement learning approaches (Barto and Mahadevan 2003) attempt to address this problem by providing a framework for learning and planning using high-level skills. One motivation behind such approaches is that an agent that has learned a set of high-level skills should be able to plan using them to quickly solve new problems, without further learning. However, planning directly in such high-dimensional, continuous state spaces remains difficult, even when using high-level skills.

By contrast, task-level planning techniques (Ghallab, Nau, and Traverso 2004) perform planning using prespecified symbolic state descriptors and operators that describe the effects of actions on those descriptors. Although these methods are usually used for planning in discrete state spaces, they have sometimes been combined with low-level motion planners or closed-loop controllers to construct robot systems that combined high-level task planning with low-level skills (Nilsson 1984; Malcolm and Smithers 1990; Cambon, Alami, and Gravot 2009; Choi and Amir 2009; Dornhege et al. 2009; Wolfe, Marthi, and Russell 2010; Kaelbling and Lozano-Pérez 2011). Here, a symbolic state at the task level refers to (and abstracts over) an infinite collection of low-level states. However, in all of these cases, significant effort was required to design symbolic descriptions of the task, and carefully specify the interface between the low-level state space and the high-level abstract task space.

Our aim is to understand how to construct task-level plans given a hierarchical reinforcement learning problem posed in a low-level, continuous state space. For simplicity, our goal is to meet the minimal set of requirements for a conformant task-planning system—finding a sequence of actions guaranteed to reach a goal state, irrespective of the low-level stochasticity of the task, and without reference to cost.

Since we aim to avoid planning in the low-level state space, the central question this effort poses is how to create the high-level abstract representations required to evaluate plans composed of sequences of skills. We show that predicate symbols can be considered classifiers, and that symbols describing low-level representations of the preconditions and effects of an agent's skills are both sufficient and necessary for task-level planning; we discuss two classes of skills for which the appropriate symbols can be concisely described. Since the appropriate predicates represent well-defined, testable conditions in the low-level state space, they are directly amenable to learning, allowing an agent to construct its own symbolic representation directly from experience. We build a representation of the continuous playroom domain (Konidaris and Barto 2009a) and use it to perform planning, and then briefly demonstrate how such a representation can be learned from experience.

Background and Setting

Semi-Markov Decision Processes

We assume that the low-level sensor and actuator space of the agent can be described as a fully observable, continuous-state semi-Markov decision process (SMDP), described by a tuple $M = (S, O, R, P, \gamma)$, where $S \subseteq \mathcal{R}^n$ is the n-dimensional state space; $O(s)$ returns a finite set of temporally extended *options* (Sutton, Precup, and Singh 1999) available in state $s \in S$; $R(s', \tau | s, o)$ is the reward received when executing action $o \in O(s)$ at state $s \in S$, and arriving in state $s' \in S$ after τ time steps; $P(s', \tau | s, o)$ is a PDF describing the probability of arriving in state $s' \in S$, τ time steps after executing action $o \in O(s)$ in state $s \in S$; and

$\gamma \in (0, 1]$ is a discount factor.

An option o consists of three components: an *option policy*, π_o, which is executed when the option is invoked; an *initiation set*, $I_o = \{s|o \in O(s)\}$, which describes the states in which the option may be executed; and a *termination condition*, $\beta_o(s) \rightarrow [0, 1]$, which describes the probability that an option will terminate upon reaching state s. The combination of reward model, $R(s', \tau|s, o)$, and transition model, $P(s', \tau|s, o)$, for an option o is known as an *option model*.

An agent that possesses option models for all of its options is capable of planning, using, for example, a sample-based SMDP planner (Sutton, Precup, and Singh 1999; Kocsis and Szepesvári 2006). However, although options can speed up planning by shortening the effective search depth (Sorg and Singh 2010; Powley, Whitehouse, and Cowling 2012), SMDP-style planning in large, continuous state spaces is still extremely difficult. Value function methods are faced with the curse of dimensionality, and sample-based methods cannot guarantee that a plan is feasible in the presence of low-level stochasticity, especially when abstracting over start states.

Task-Level Planning

While SMDPs enable temporal abstraction, task-level approaches operate using symbolic descriptions of state; consequently, they are often able to find plans that are much longer-range than could be effectively discovered by planning directly in state space. The best known task-level planning language is probably STRIPS (Fikes and Nilsson 1971), where a planning domain is described using a set of predicates and operator descriptions. A state in STRIPS is a conjunction of positive literal predicates. Each operator description describes the preconditions (a conjunction of positive predicates) that must hold in a state to take an action, and that action's effects (a list of predicates to be added to or deleted from the state). Operators may also have parameters, which can be used as arguments for their precondition and effect predicates. When applied to a robot domain, each predicate refers to an infinite set of low-level robot states; abstracting in this way (and restricting the form of the state and operator descriptors) enables efficient high-level abstract planning. However, in all such work so far, the set of predicates has been supplied by a human designer, who must also carefully specify the interface between the low-level state space and the high-level abstract task space.

Symbols and Plans

Given an SMDP, our goal is to build an abstract representation of the task for use in symbolic planning. In this section we define a predicate symbol as a *test* that is true in some states. We then define a plan as a sequence of option executions from a set of start states, and the plan space as the set of all possible plans. This allows us to reason about the symbols required to evaluate the feasibility of any plan in the plan space. We begin with a working definition of the notion of a symbol:

Definition 1. *A propositional symbol σ_Z is the name associated with a test τ_Z, and the corresponding set of states $Z = \{s \in S \mid \tau_Z(s) = 1\}$.*

Each symbol σ_Z is defined as both a test (or classifier) τ_Z defined over the state space, and an associated set $Z \subseteq S$ for which the test returns true. The set of states is an *extensional representation* of the symbol: it lists the states that the symbol refers to. The classifier is an *intensional representation*: a function that determines membership in the set of states.

The power of task-based planning is due to the ability to manipulate and reason with intensional representations. Thus, the class of classifiers we choose to serve as symbols must support the ability to efficiently compute the union and intersection of two classifiers, the complement of a classifier, and to determine whether a classifier refers to the empty set (which gives us the ability to determine whether one classifier is a subset of another). This gives us a *symbol algebra* (technically a concrete boolean algebra) defined over classifiers, which allows us to compute classifiers representing the results of boolean operations (or, and, not, and implication) applied to our symbols. Examples of classifier types that support these operations include decision trees and classifiers with linear decision boundaries. For abstract subgoal options, we will additionally require the ability to perform a projection (we define both abstract subgoal options and the projection operator in a later section).

We next define the *plan space* of a domain.

Definition 2. *A* plan $p = \{o_1, ..., o_{p_n}\}$ *from a state set $Z \subseteq S$ is a sequence of actions $o_i \in O$, $1 \leq i \leq p_n$, that is to be executed from some state in Z. A plan is* feasible *when the probability of the agent being able to execute it is* 1*, i.e., $\nexists \tilde{S} = \{s_1, ..., s_j\}$ such that $s_1 \in Z$, $o_i \in O(s_i)$ and $P(s_{i+1}|s_i, o_i) > 0, \forall i < j$, and $o_j \notin O(s_j)$ for any $j < p_n$.*

Definition 3. *The* plan space *for an SMDP is the set of all tuples (Z, p), where $Z \subseteq S$ is a set of states (equivalently, a* start symbol*) in the SMDP, and p is a plan.*

For convenience, we treat the agent's goal set g (or each goal g, if the agent has multiple alternatives)—which is itself a propositional symbol—as an option that does nothing but can only be executed when the agent has reached g. Given goal g, option o_g has initiation set $I_g = g$, termination condition $\beta_g(s) = 1, \forall s$, and a null policy.

Definition 4. *A plan tuple is* satisficing *for goal g if it is feasible, and its final action is goal option o_g.*

The role of a symbolic representation is therefore to be able to determine whether any given plan tuple (Z, p) is satisficing for goal g, which we can achieve by testing whether it is feasible (since then determining whether it is satisficing follows trivially). In the following section, we define a set of propositional symbols and show that the ability to represent them is sufficient to do so in any SMDP.

Symbols for Propositional Planning

In order to test the feasibility of a plan, we begin with a set of possible start states, and repeatedly compute the set of states that can be reached by each option, checking in turn that that set is a subset of the initiation set of its successor option. We will therefore construct abstract representations

that correspond to the initiation set of each option, as well as allow us to represent the image of an option from a set of states (the set of states reachable by executing an option from any state in the starting set). We begin by defining a class of symbols that represent the initiation set of an option:

Definition 5. *The* precondition *of option o is the symbol referring to its initiation set: Pre$(o) = \sigma_{I_o}$.*

We now define a symbolic operator that computes the consequences of taking an action in a set of states:

Definition 6. *Given an option o and a set of states $X \subseteq S$, we define the image of o from X (denoted Im(X, o)) as: Im$(X, o) = \{s' | \exists s \in X, P(s'|s, o) > 0\}$.*

The image operator $\text{Im}(X, o)$ computes the set of states that might result from executing option o from some state in X. The ability to express the image operator symbolically (in addition to represent each option precondition) is sufficient to evaluate the feasibility of any plan:

Theorem 1. *Given an SMDP, the ability to represent the preconditions of each option and to symbolically compute the image operator is sufficient for determining whether any plan tuple (Z, p) is feasible.*

Proof. Consider an arbitrary plan tuple (Z, p), with plan length n. To determine whether it is feasible, we can set $z_0 = Z$ and repeatedly compute $z_{j+1} = \text{Im}(z_j, p_j)$, for $j \in \{1, ..., n\}$. The plan tuple is feasible if and only if $z_i \subseteq \text{Pre}(p_{i+1}), \forall i \in \{0, ..., n-1\}$. □

Since the feasibility test in the above proof is biconditional, any other feasibility test must express exactly those condition for each pair of successive options in a plan. In some SMDPs, we may be able to execute a different test that happens to evaluate to the same value everywhere (e.g., if every option conveniently flips a set of bits indicating which options can be run next), but we can employ an adversarial argument to construct an SMDP in which any other test is incorrect. Representing the image operator and precondition sets are therefore also necessary for abstract planning.

We call the symbols required to name an option's initiation set, and express its image operator that option's *characterizing symbols*. These symbols are analogous to an operator pre-condition and effect in STRIPS. The most important difference is that the characterizing symbols have intensional definitions in the low-level state space—they are each represented as atomic symbols, rather than as a conjunction of predefined abstract symbols. This distinction is important because the characterizing symbols are precisely defined in terms of the domain and the available options only—they do not depend on anything else; in particular, they do not depend on any other symbols. Nevertheless, they allow us to compute the results of symbolic operations by operating on the intensional definitions directly.

The ability to perform symbolic planning thus hinges on the ability to symbolically represent the image operator. Although it might seem that we can simply use it as defined—via a low-level model of the option—this is an extensional definition that outputs a distribution for a single start state; it does not allow us to compute the set of possible next states for an uncountably infinite number of start states in finite time. Therefore, we must attempt to define the intensional symbols necessary to compute it symbolically. However, doing this for an arbitrary option can be arbitrarily hard. Consider an option that maps each state in its initiation set to a single (but arbitrary and unique) state in S. For this option, $\text{Im}(\{s\}, o) = \{s'\}$, for some arbitrary pair of $s, s' \in S$. In this case we can do no better than than expressing $\text{Im}(Z, o)$ as a union of an infinite number of singleton symbols. Fortunately, however, we can concisely represent the image operator for at least two classes of options in common use.

Subgoal Options

One common type of option—especially prevalent in research on skill discovery methods—is the *subgoal option* (Precup 2000). Such an option reaches a set of states (referred to as its subgoal) before terminating, and the state it terminates in can be considered independent of the state from which it is executed. This results in a particularly simple way to express the image operator.

Definition 7. *The* effect set *of subgoal option o is the symbol representing the set of all states that an agent can possibly find itself in after executing o: Eff$(o) = \{s' | \exists s \in S, \tau, P(s', \tau|s, o) > 0\}$.*

For a subgoal option, $\text{Im}(Z, o) = \text{Eff}(o), \forall Z \subseteq S$.

A common generalization of a subgoal option is the *partitioned subgoal option*. Here, the option's initiation set can be partitioned into two or more subsets, such that the option behaves as a subgoal option from each subset. For example, consider an option that we might describe as *walk through the door you are facing*; if there are a small number of such doors, then the agent can be considered to execute a separate subgoal option when standing in front of each. In such cases, we must explicitly represent each partition of the initiation set separately; the image operator can then be computed using the union of all applicable effect sets.

Unfortunately, using subgoal options severely restricts the potential goals an agent may plan for: all feasible goals must be a superset of one of the effect sets. However, they lead to a particularly simple planning mechanism. Since a subset test between the precondition of one option and the effects set of another is the *only* type of expression that need ever be evaluated, we can test all such subsets in time $O(n^2)$ for $O(n)$ characterizing sets, and build a directed plan graph G with a vertex for each characterizing set, and an edge present between vertices i and j if and only if $\text{Eff}(o_i) \subseteq \text{Pre}(o_j)$. A plan tuple (p, Z), where $p = \{o_1, ..., o_{p_n}\}$ is feasible if and only if $Z \subseteq I_{o_1}$, there exists a path in G from o_1 to o_{p_n}, and the goal is a superset of the effects set of o_{p_n}. This condition suggests that options should be constructed so that their termination conditions lie inside the initiation sets for potential successor options, and is the underlying principle behind pre-image backchaining (Lozano-Perez, Mason, and Taylor 1984; Burridge, Rizzi, and Koditschek 1999; Tedrake 2009) and skill chaining (Konidaris and Barto 2009b).

Abstract Subgoal Options

A more general type of option is the *abstract subgoal option*, which results in some of the features of the state vector taking on a particular set of values, and others remaining unchanged. (As above, we may generalize abstract subgoal options to *partitioned abstract subgoal options*.) The subgoal is said to be abstract because it is satisfied for any value of the unchanged variables. For example, an option to grasp an object might terminate when that object has been grasped; the room the robot is in, the position of its other hand, and its location in the building are all irrelevant and unaffected.

An abstract subgoal option is analogous to a STRIPS operator, which specifies the symbolic values that change and assumes the rest remain unchanged. However, STRIPS makes the assumption that a state is specified by a conjunction of predicates, that those predicates are known in advance, that the same predicates both describe the state and operator preconditions and effects, and that the only effects permissible are deleting or adding a conjunct. These conditions are difficult to enforce in our case because we may have states that are more complicated (e.g., include some but not all of, or more than, a particular state set), or are not conjunctions, and it is difficult to guarantee that we can express option preconditions and effects using the same predicates.

Without loss of generality, we can write a state vector $s \in S$ given o as $s = [a, b]$, where a is the part of the state vector that changes when executing o (o's *mask*), and b is the part that does not. This allows us to define the following operator:

Definition 8. *Given an option o and a set of states $Z \subseteq S$, we define the projection of Z with respect to o (denoted Project(Z, o)) as: Project*$(Z, o) = \{[a, b] | \exists a', [a', b] \in Z\}$.

Projection expands the set of states named by Z by projecting away the restrictions on the value of the changeable portion of the state descriptor. The projection operator cannot be obtained using set operations, and must instead be obtained by directly altering the classifier that represents the test associated with the symbol, because its definition involves operations on the low-level state representation.

Since the $\text{Eff}(o)$ set specifies all possible effects of the option execution, we can compute the image operator as $\text{Im}(Z, o) = \text{Project}(Z, o) \cap \text{Eff}(o)$. This is process depicted in Figure 1, for an abstract subgoal option that achieves a subgoal in one dimension and leaves the other unchanged.

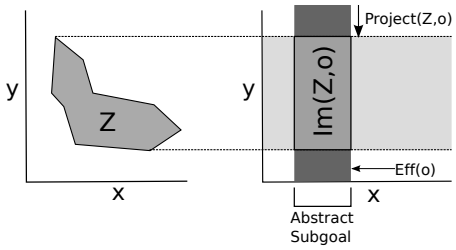


Figure 1: $\text{Im}(Z, o)$ for abstract subgoal option o can be expressed as the intersection of the projection operator and o's effects set. Here, o has a subgoal in the x dimension, and leaves the y dimension unchanged. Note that, in general, the projection operator may output multiple disjoint sets.

Parametrized Symbols

So far we have defined only propositional symbols. However, in order to obtain compact symbolic descriptions, and in order to generalize, it may be advantageous to use *lifted* symbols, where the test represented by the symbol is parametrized by some aspect of the problem.

There are two sources of parameters. The first allows us to obtain compact descriptions by combining the classifiers used to represent the characterizing sets of a partitioned (potentially abstract) subgoal option into a smaller number of characterizing symbols by conditioning the effects set on some feature of the state space. For example, as before we might have an option for passing through the door in front of the agent, modeled as a partitioned subgoal option by having a separate precondition and effect for each individual door. We can generalize this option by parametrizing it with the location of each door.

The second source of parametrizing variables is when the options themselves are parametrized. For example, a robot might have a controller for grasping, parametrized by a deictic variable *GraspTarget*. In such cases, the characterizing set must be parametrized by the option parameters.

In both cases the parametrizing variables could be indexes, which select among several precondition and effect sets. However, if there are regularities in the characterizing sets, we may be able to represent them compactly, rather than as an explicit list. In this way, an agent that builds its symbols through learning can generalize experience obtained using one instantiation of the variables to all others.

Symbolic Planning in the Continuous Playroom Domain

In the continuous playroom domain (Konidaris and Barto 2009a), an agent with three effectors (an eye, a hand, and a marker) is placed in a room with five objects (a light switch, a bell, a ball, and red and green buttons) and a monkey; the room also has a light (initially off) and music (also initially off). The agent is given options that allow it to move a specified effector over a specified object (always executable, resulting in the effector coming to rest uniformly at random within a fixed radius of the object), plus an "interact" option for each object (for a total of 20 options). The effectors and objects are arranged randomly at the start of every episode; one arrangement is depicted in Figure 2.

Interacting with the buttons or the bell requires the light to be on, and the agent's hand and eye to be placed over the relevant object. The ball and the light switch are brightly colored and can be seen in the dark, so they do not require the light to be on. Interacting with the green button turns the music on; the red button turns it off. Interacting with the light switch turns the lights on or off. Finally, if the agent's marker is on the bell and it interacts with the ball, the ball is thrown at the bell and makes a noise. If this happens when the lights are off and the music is on, the monkey cries out, and the episode ends. The agent's state consists of 33 continuous state variables describing the x and y distance between

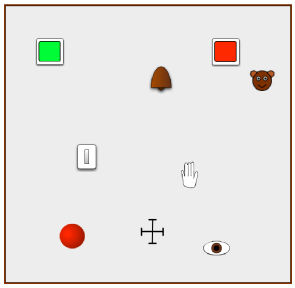


Figure 2: The continuous playroom domain.

each of the agent's effectors and each object, the light level (which is at 0 when the light is off, and is 1 at the center of the room when the light is on, dropping off with the square of the distance from that location), the music volume (selected at random from the range $[0.3, 1.0]$ when the green button is pressed), and whether the monkey has cried out.

Symbolic Planning

We first demonstrate planning in the continuous playroom using a symbolic representation that models the options as partitioned abstract subgoal options. This representation was designed by hand, guided by the theory presented above. We implemented a very simple planner in Java using breadth-first search. Timing results are given in Table 1.

Goal	Depth	Nodes	Time (s)
Lights On	3	199	1.35
Lights & Music On	6	362	2.12
Monkey Cries	13	667	3.15

Table 1: Timing results for a few example plans, obtained using a MacBook Pro with a 2.5Ghz Intel Core i5 processor and 8GB of RAM.

Each plan starts from the set of all states where the monkey is not crying out, and the lights and music are off, and is guaranteed to be able to reach the specified goal with probability 1 from any state in the start set, for any random configuration of the domain. We know of no existing sample-based SMDP planner that can provide similar guarantees.

Acquiring Symbols from Experience

To demonstrate symbol acquisition from experience, we gathered $5,000$ positive and negative examples[1] of each option's initiation set and effects set by repeatedly creating a new instance of the playroom domain and sequentially executing options at random. For the effects set, we used option termination states as positive examples, and states encountered during option execution but before termination as negative examples. We used the WEKA toolkit (Hall et al. 2009) C4.5 decision tree (Quinlan 1993) for symbol learning. A representative learned initiation set, for interacting with the green button, is given in Figure 3.

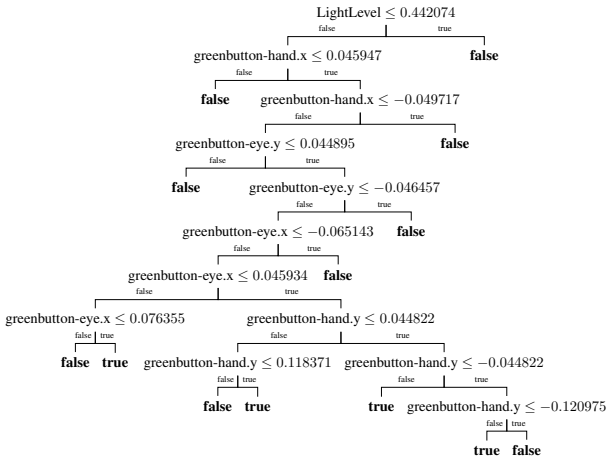


Figure 3: The decision tree representing the learned initiation set symbol for interacting with the green button. It approximately expresses the set of states we might label *the hand and eye are over the green button, and the light is on.*

An example effect set for moving an effector over an object is shown in Figure 4. The effects sets for the interaction options were slightly less straightforward, because in most cases executing the option did not change the state. For example, interacting with the green button only changed the state when the music was not already on. However, learning this partition is trivial. Given that the music at time t was off, the resulting learned decision tree simply describes the set where the music is turned on at time $t + 1$. A similar pair of effects sets is learned for interacting with the light switch, depending on whether or not the light was on before executing the option.

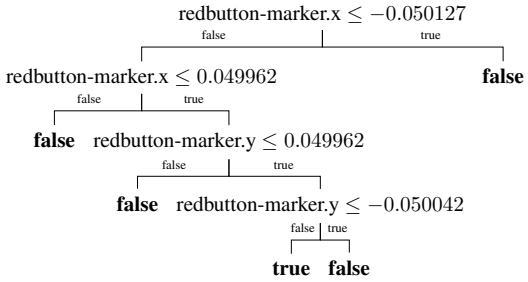


Figure 4: The decision tree representing the learned effects set of moving the marker to the red button. It approximately expresses the set of states we might label *the marker is over the red button.*

Related Work

Several researchers have learned symbolic models of the preconditions and effects of pre-existing controllers for later use in planning (Drescher 1991; Schmill, Oates, and Cohen

[1]This number was chosen at random, and does not reflect the difficulty of learning the relevant sets.

2000; Pasula, Zettlemoyer, and Kaelbling 2007; Amir and Chang 2008; Kruger et al. 2011; Lang, Toussaint, and Kersting 2012; Mourão et al. 2012); similar approaches have been applied under the heading of relational reinforcement learning (Džeroski, De Raedt, and Driessens 2001). However, in all of these cases, the high-level symbolic predicates (or attributes) used to learn the model were pre-specified; our work shows how to specify them.

We know of very few systems where a high-level, abstract state space is formed using low-level descriptions of skills. Huber (2000) uses an approach based on a discrete event dynamic system, where the state of the system is described by the (discrete) state of a set of controllers. For example, if a robot has three controllers that can be in one of four states (cannot execute, can execute, executing, completed), then we can form a discrete state space made up of three attributes, each of which can take on four possible values. Hart (2009) used this formulation to learn hierarchical manipulation schema.

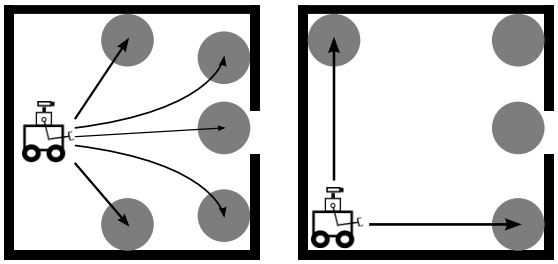
Modayil and Kuipers (2008) show that a robot can learn to distinguish objects in its environment via unsupervised learning, and then use a learned model of the motion of the object given an action to perform task-level planning. However, the learned models are still in the original state space. Later work by Mugan and Kuipers (2012) use *qualitative distinctions* to adaptively discretize a continuous state space in order to acquire a discrete model suitable for planning; here, discretization is based on the ability to predict the outcome of executing an action.

Other approaches to MDP abstraction have focused on discretizing large continuous MDPs into abstract discrete MDPs (Munos and Moore 1999) or minimizing the size of a discrete MDP model (Dean and Givan 1997).

Discussion and Conclusions

Although we have not addressed the question of how skills should be designed to enable effective planning, our work does suggest some guidelines. First, the initiation set of an option should be as large as possible, to increase the likelihood of a feasible plan; conversely, an option's image should should be compact, for the same reason. This argument provides support for the broadly accepted idea that skills should be "funnel-shaped" (Lozano-Perez, Mason, and Taylor 1984; Burridge, Rizzi, and Koditschek 1999; Tedrake 2009), in that they reliably move the agent from a large number of start states to a small number of end states. Similarly, option policies should largely subsume the stochasticity of the underlying task, in order to reduce the size of each option's image.

The most interesting implication of the ideas presented here is that motor skills play a central role in determining the representational requirements of an intelligent agent. In order to plan symbolically, an agent's representation must therefore directly model properties of its skills; *a suitable symbolic description of a domain depends on the skills available to the agent.* For example, Figure 5 shows two cases where the same robot must navigate the same room, but requires different symbolic representations because it uses different navigation subgoal options. In Figure 5(a), the robot uses options that drive it to the centroid of each wall. As a consequence, the symbols required to represent the effect of executing its options might be described as $AtLeftWall$, $AtRightWall$, $AtEntrance$, etc. In Figure 5(b), the robot uses options that perform wall-following to reach a corner or a door. As a consequence, the symbols required to represent the effect of executing its options could be described as $LowerLeftCorner$, $LowerRightCorner$, etc.



(a) (b)

Figure 5: A robot navigating a room using a) subgoal options for moving to distinct walls, or b) wall-following subgoal options. The effect set for each option is shown in gray.

This approach contrasts with the standard way of merging task-level planning and low-level control (first employed in Shakey, the earliest such robot system (Nilsson 1984)), where human designers specify a symbolic description of the world and then write the perceptual and motor control programs necessary to ground and execute plans formed using that description. Such approaches can be considered *symbol-first*. By contrast, our approach can be considered *skill-first*, where we use an existing set of skills as a basis for building a representation appropriate for planning.

Although our formalism is motivated by the problem of hierarchical robot control, much remains to be done before it could be employed on a robot—at the minimum, it should be extended to include cost and afford probabilistic planning. Actively learning preconditions and effects may also prove necessary; Nguyen and Kemp (2011) describe promising early work in this direction. Another critical question is that of *transfer*. Our framework has so far been defined on top of a single base SMDP. However, an agent should be able to use skills gained in one problem setting to more effectively solve another. While there has been some work on defining portable skills (Konidaris and Barto 2007), portable symbols will have to be either parametrized by details of the environment, or use an intermediate function to translate agent-centric characterizing sets to problem-specific ones.

Our attempt to merge a low-level SMDP with a high-level classical planning formulation imposes some quite restrictive conditions on the agent's actions. A simpler approach is to use low-level models of those actions to plan in the original state space; this has the advantage of retaining all of the original SMDP guarantees, and imposing no skill restrictions. However, it does not afford the powerful state abstraction mechanisms that we believe will ultimately prove necessary for hierarchical planning in sophisticated robots.

References

Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research* 33:349–402.

Barto, A., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13:41–77.

Burridge, R.; Rizzi, A.; and Koditschek, D. 1999. Sequential composition of dynamically dextrous robot behaviors. *International Journal of Robotics Research* 18(6):534–555.

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

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

Dean, T., and Givan, R. 1997. Model minimization in Markov decision processes. In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 106–111.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security and Rescue Robotics*.

Drescher, G. 1991. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine learning* 43(1):7–52.

Fikes, R., and Nilsson, N. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory and practice*. Morgan Kaufmann.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. 2009. The WEKA data mining software: An update. *SIGKDD Explorations* 11(1):10–18.

Hart, S. 2009. *The Development of Hierarchical Knowledge in Robot Systems*. Ph.D. Dissertation, University of Massachusetts Amherst.

Huber, M. 2000. A hybrid architecture for hierarchical reinforcement learning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, 3290–3295.

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

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, 282–293.

Konidaris, G., and Barto, A. 2007. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*.

Konidaris, G., and Barto, A. 2009a. Efficient skill learning using abstraction selection. In *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*.

Konidaris, G., and Barto, A. 2009b. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22*, 1015–1023.

Kruger, N.; Geib, C.; Piater, J.; Petrick, R.; Steedman, M.; Wörgötter, F.; Ude, A.; Asfour, T.; Kraft, D.; Omrčen, D.; Agostini, A.; and Dillmann, R. 2011. Object-action complexes: Grounded abstractions of sensory-motor processes. *Robotics and Autonomous Systems* 59:740–757.

Lang, T.; Toussaint, M.; and Kersting, K. 2012. Exploration in relational domains for model-based reinforcement learning. *Journal of Machine Learning Research* 13:3691–3734.

Lozano-Perez, T.; Mason, M.; and Taylor, R. 1984. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research* 3(1):3–24.

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

Modayil, J., and Kuipers, B. 2008. The initial development of object knowledge by a learning robot. *Robotics and Autonomous Systems* 56(11):879–890.

Mourão, K.; Zettlemoyer, L.; Patrick, R.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. In *Proceedings of Conference on Uncertainty in Articial Intelligence*.

Mugan, J., and Kuipers, B. 2012. Autonomous learning of high-level states and actions in continuous environments. *IEEE Transactions on Autonomous Mental Development* 4(1):70–86.

Munos, R., and Moore, A. 1999. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1348–1355.

Nguyen, H., and Kemp, C. 2011. Autonomous active learning of task-relevant features for mobile manipulation. In *The RSS 2011 Workshop on Mobile Manipulation: Learning to Manipulate*.

Nilsson, N. 1984. Shakey the robot. Technical report, SRI International.

Pasula, H.; Zettlemoyer, L.; and Kaelbling, L. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29:309–352.

Powley, E.; Whitehouse, D.; and Cowling, P. 2012. Monte carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games*, 234–241.

Precup, D. 2000. *Temporal Abstraction in Reinforcement Learning*. Ph.D. Dissertation, Department of Computer Science, University of Massachusetts Amherst.

Quinlan, J. 1993. *C4.5: programs for machine learning*, volume 1. Morgan Kaufmann.

Schmill, M.; Oates, T.; and Cohen, P. 2000. Learning planning operators in real-world, partially observable environments. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 245–253.

Sorg, J., and Singh, S. 2010. Linear options. In *Proceedings of the Ninth International Conference on Autonomous Systems and Multiagent Systems*, 31–38.

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

Tedrake, R. 2009. LQR-Trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*, 18–24.

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined Task and Motion Planning for Mobile Manipulation. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, 254–258.