Affordance-Aware Planning

Abstract

Planning algorithms for non-deterministic domains are often intractable in large state spaces due to the well-known curse of dimensionality. Existing approaches to planning in large stochastic state spaces fail to prevent autonomous agents from considering many actions that are obviously irrelevant to a human solving the same task. To prevent agents from applying irrelevant actions we formalize the notion of *affordances* as state space independent, goal-oriented knowledge added to an Object Oriented Markov Decision Process (OO-MDP). Affordances prune irrelevant actions based on the agent's goal and the current state, reducing the number of state-action pairs the planner must evaluate in order to formulate a near optimal policy. Affordances may be provided by an expert or may be learned without supervision. We demonstrate our approach by planning in the state-rich Minecraft domain, showing significant increases in speed, reductions in state space exploration, and improvements in the quality of the synthesized policy. Additionally, we show that learned affordances often surpass the performance of those provided by experts. Finally, we demonstrate that affordance-aware planning enables a robot to assist a person performing a cooking task.

Introduction

Robots operating in unstructured, stochastic environments such as a factory floor or a kitchen face a difficult planning problem due to the large state space and inherent uncertainty due to unreliable perception and actuation [5, 16]. Robotic planning tasks are often formalized as a stochastic sequential decision making problem, modeled as a Markov Decision Process (MDP) [28]. In these problems, the agent must find a mapping from states to actions for some subset of the state space that enables the agent to achieve a goal while minimizing costs along the way. However, many robotics tasks are so complex that modeling them as an MDP results in a massive state-action space, which in turn restricts the types of robotics problems that are computationally tractable. For example, when a robot is manipulating objects in an environment, an object can be placed anywhere in a large set of locations. The size of the state space increases exponentially with the number of objects, which bounds the placement problems that the robot is able to expediently solve. The difficulty of the task is compounded by the fact that most of these objects and locations are irrelevant given a specific goal. For instance, when making brownies, the oven and flour are important, while the soy sauce and sauté pan are not. For a different task, such as stir-frying broccoli, a different set of objects and actions must be inferred.

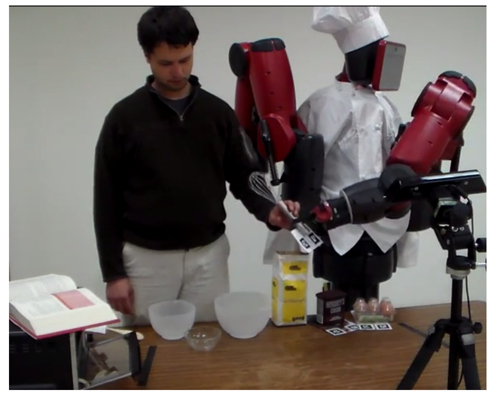

Figure 1: Affordances enable a robot to efficiently infer helpful actions in very large state spaces, such as a kitchen.

To address this state-action space explosion, prior work has explored adding knowledge to the planner, such as options [27] and macro-actions [6, 22]. However, while these methods can allow the agent to search more deeply in the state space, they add non-primitive actions to the planner which *increase* the size of the state-action space. The resulting augmented space is even larger, which can have the paradoxical effect of increasing the search time for a good policy [15]. Deterministic forward-search algorithms like hierarchical task networks (HTNs) [21], and temporal logical planning (TLPlan) [2, 3], add knowledge to the planner that greatly increases planning speed, but do not generalize to stochastic domains. Additionally, the knowledge provided to the planner by these methods is quite extensive, reducing the agent's autonomy.

To address these issues, we augment an Object Oriented Markov Decision Process (OO-MDP) with a formalization of *affordances*. Affordances were originally proposed by Gibson [12] as action possibilities prescribed by an agent's capabilities in an environment. We rigorously formalize the notion of an affordance as knowledge added to an OO-MDP that prunes irrelevant actions on a state-by-state basis based on the agent's current goal. Affordances can be specified by hand and also learned through experience, making them a concise, transferable, and learnable means of representing useful planning knowledge. Our experiments demonstrate that affordances provide dramatic improvements for a variety of planning tasks compared to baselines, and are applicable across different state spaces. Moreover, while manually provided affordances outperform baselines, affordances learned through experience yield even greater improvements. We conduct experiments in the game Minecraft, which has a very large state-action space, and on a real-world robotic cooking assistant. All associated code with this paper may be found at (redacted for review).

Related Work

In this section, we discuss the differences between affordance-aware planning and other forms of knowledge engineering that have been used to accelerate planning. This paper builds on previous work published at two workshops (redacted for review).

Stochastic Approaches

Here, we compare other approaches of action pruning and knowledge engineering that provide speedups to planners in stochastic domains.

Temporally extended actions are actions that the agent can select like any other action of the domain, except executing them results in multiple primitive actions being executed in succession. Two common forms of temporally extended actions are *macro-actions* [14] and *options* [27]. Macro-actions are actions that always execute the same sequence of primitive actions. Options are defined with high-level policies that accomplish specific sub tasks. For instance, when an agent is near a door, the agent can engage the 'door-opening-option-policy', which switches from the standard high-level planner to running a policy that is crafted to open doors. Although the classic options framework is not generalizable to different state spaces, creating *portable* options is a topic of active research [19, 17, 24, 8, 1, 18].

Since temporally extended actions may negatively impact planning time [15] by adding to the number of actions the agent can choose from in a given state, combining affordances with temporally extended actions allows for even further speedups in planning, as demonstrated in Table 3. In other words, affordances are complementary knowledge to options and macro-actions.

Sherstov and Stone [26] considered MDPs for which the action set of the optimal policy of a source task could be transferred to a new, but similar, target task to reduce the learning time required to find the optimal policy in the target task. Aaffordances prune away actions on a state-by-state basis, enabling more aggressive pruning whereas the learned action pruning is on per-task level.

Rosman and Ramamoorthy [25] provide a method for learning action priors over a set of related tasks. Specifically, they compute a Dirichlet distribution over actions by extracting the frequency that each action was optimal in each state for each previously solved task. Action priors can only be used with planning/learning algorithms that work well with an ϵ-greedy rollout policy, while affordances can be applied to almost any MDP solver. Action priors are only active for a fraction ϵ of the time steps, which is quite small, limiting the improvement they can make to the planning speed. Finally, as variance in tasks explored increases, the priors will become more uniform. In contrast, affordance-aware planning can handle a wide variety of tasks in a single knowledge base, as demonstrated by Table 2.

Heuristics in MDPs are used to convey information about the value of a given state-action pair with respect to the task being solved and typically take the form of either value function initialization [13], or reward shaping[23]. However, heuristics are highly dependent on the reward function and state space of the task being solved, whereas affordances are state space independent and may be learned easily for different reward functions. If a heuristic can be provided, the combination of heuristics and affordances may even more greatly accelerate planning algorithms than either approach alone.

Deterministic Approaches

There have been several attempts at engineering knowledge to decrease planning time for deterministic planners. These are fundamentally solving a different problem from what we are interested in since they deal with non-stochastic problems, but there are interesting parallels nonetheless.

Hierarchical Task Networks (HTNs) employ *task decompositions* to aid in planning [11]. The agent decomposes the goal into smaller tasks which are in turn decomposed into smaller tasks. This decomposition continues until immediately achievable primitive tasks are derived. The current state of the task decomposition, in turn, informs constraints which reduce the space over which the planner searches. At a high level HTNs and affordances both achieve action pruning by exploiting some form of supplied knowledge.

However HTNs do not incorporate reward into their planning. Consequently, they lack any guarantees of the quality of any induced plan. Additionally, the degree of supplied knowledge in HTNs far exceeds that of affordances: HTNs require not only constraints for sub-tasks but a hierarchical framework of arbitrary complexity. Affordances require either simple symbolic knowledge, as illustrated in Table 1, or a set of predicates for use as features and a means of generating candidate state spaces.

Bacchus and Kabanza [2, 3] provided planners with domain dependent knowledge in the form of a first-order version of linear temporal logic (LTL), which they used for control of a forward-chaining planner. With this methodology, a STRIPS style planner may be guided through the search space by pruning candidate plans that falsify the given knowledge base of LTL formulas, often achieving polynomial time planning in exponential space. LTL formulas are difficult to learn, placing dependence on an expert, while we demonstrate that affordances can be automatically learned from experience.

Technical Approach

We define affordances as formal knowledge added to a Markov Decision Process (MDP). An MDP is a five-tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a state space; $\mathcal{A}$ is the agent's set of actions; $\mathcal{T}$ denotes $\mathcal{T}(s' \mid s, a)$, the transition probability of an agent applying action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ and arriving in $s' \in \mathcal{S}$; $\mathcal{R}(s, a, s')$ denotes the reward received by the agent for applying action a in state s and transitioning to state s'; and $\gamma \in [0, 1)$ is a discount factor that defines how much the agent prefers immediate rewards over future rewards (the agent prefers to maximize immediate rewards as γ decreases).

Our representation of affordances builds on Object-Oriented MDPs (OO-MDPs) [10]. An OO-MDP efficiently represents the state of an MDP through the use of objects and predicates. An OO-MDP state is a collection of objects, $O = \{o_1, \ldots, o_o\}$. Each object o_i belongs to a class, $c_j \in \{c_1, \ldots, c_c\}$. Every class has a set of attributes, $Att(c) = \{c.a_1, \ldots, c.a_a\}$, each of which has a domain, $Dom(c.a)$, of possible values. OO-MDPs enable planners to use predicates over classes of objects. That is, the OO-MDP definition also includes a set of predicates $\mathcal{P}$ that operate on the state of objects to provide additional high-level information about the MDP state.

OO-MDP predicates provide state space independence. For a given planning domain, OO-MDP objects often appear across tasks. Since predicates operate on collections of objects, they generalize beyond specific state spaces within the domain. For instance, in Minecraft, a predicate checking the contents of the agent's inventory generalizes beyond any particular Minecraft task. We capitalize on this state space independence by using OO-MDP predicates as features for action pruning.

Modeling the Optimal Actions

Our goal is to formalize affordances in a way that enables a planning algorithm to prune away suboptimal actions in each state. We define the optimal action set, $\mathcal{A}^*$, for a given state s and goal G as:

$$\mathcal{A}^* = \{a \mid Q^*_G(s, a) = V^*_G(s)\}, \tag{1}$$

where $Q^*_G(s, a)$ and $V^*_G(s)$ represent the optimal Q function and value function, respectively.

We aim to learn a probability distribution over the optimality of each action for a given state (s), goal (G), and knowledge base (K) by which action pruning may be informed. Thus, we want to infer a Bernouli distribution for each action's optimality:

$$\Pr(a_i \in \mathcal{A}^* \mid s, G, K) \tag{2}$$

for $i \in \{1, \ldots, |\mathcal{A}|\}$, where $\mathcal{A}$ is the OO-MDP action space.

We formalize our knowledge base, K, as a set of n paired preconditions and goal types, $\{(p_1, g_1) \ldots (p_n, g_n)\}$, along with a parameter vector, θ. We abbreviate each pair (p_j, g_j) to δ_j for simplicity. Each precondition $p \in \mathcal{P}$ is a *predicate* in predicate space, $\mathcal{P}$, defined by the OO-MDP, and g is a *goal type* which is a logical expression that defines a subset of goals. For example, a predicate might be $nearTrench(agent)$ which is true when the agent is standing near a trench. A goal type specifies the sort of problem the agent is trying to solve. In the context of Minecraft, a goal type might refer to the agent retrieving an object of a certain type from the environment, reaching a particular location, or crafting an object or structure. Depending on the agent's current goal, the relevance of each action changes dramatically.

We rewrite Equation 2 replacing K with its constituents:

$$\Pr(a_i \mid s, G, K) = \Pr(a_i \in \mathcal{A}^* \mid s, G, \delta_1 \ldots \delta_n, \theta_i) \tag{3}$$

where θ_i represents the set of parameters relevant to modeling the probability of action $a_i \in \mathcal{A}^*$.

We introduce the indicator function f, which returns 1 if and only if δ_j's predicate is true in the provided state s, and δ_j's goal type is entailed by the agent's current goal, G:

$$f(\delta, s, G) = \begin{cases} 1 & \delta.p(s) \land \delta.g(G) \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Evaluating f for each δ_j given the current state and goal gives rise to a set of binary features, $\phi_j = f(\delta_j, s, G)$, which we use to reformulate our probability distribution:

$$\begin{aligned} &\Pr(a_i \in \mathcal{A}^* \mid s, G, \delta_1 \ldots \delta_n, \theta_i) \\ &\qquad = \Pr(a_i \in \mathcal{A}^* \mid \phi_1, \ldots, \phi_n, \theta_i) \end{aligned} \tag{5}$$

This distribution may be modeled in a number of ways making this approach quite flexible. To enable efficient learning, we model our distribution using Naive Bayes. First we factor using Bayes' rule:

$$= \frac{\Pr(\phi_1, \ldots, \phi_n, \mid a_i \in \mathcal{A}^*, \theta_i) \Pr(a_i \in \mathcal{A}^* \mid \theta_i)}{\Pr(\phi_1, \ldots, \phi_n | \theta_i)} \tag{6}$$

Next we assume that each feature is conditionally independent of the others, given whether the action is optimal:

$$= \frac{\prod_{j=1}^{n} \Pr(\phi_j \mid a_i \in \mathcal{A}^*, \theta_i) \Pr(a_i \in \mathcal{A}^* \mid \theta_i)}{\Pr(\phi_1, \ldots, \phi_n | \theta_i)} \tag{7}$$

Finally, we define the prior on the optimality of each action to be the fraction of the time each action was optimal during training.

Learning the Optimal Actions

Our approach to modeling the optimality of each action allows affordances to be learned through unsupervised experience. To learn affordances, we provide a set of training worlds (W) for which the optimal policy, π, may be tractably computed. Then, we compute the maximum likelihood estimate of the parameter vector θ_i for each action.

Under our Bernouli Naive Bayes model, we estimate the parameters $\theta_{i,0} = \Pr(a_i)$ and $\theta_{i,j} = \Pr(\phi_j | a_i)$, for $j \in \{1, \ldots, n\}$, where the maximum likelihood estimates are:

$$\theta_{i,0} = \frac{C(a_i)}{C(a_i) + C(\bar{a}_i)} \tag{8}$$

$$\theta_{i,j} = \frac{C(\phi_j, a_i)}{C(a_i)} \tag{9}$$

Here, $C(a_i)$ is the number of observed occurrences where a_i was optimal across all worlds W, $C(\bar{a_i})$ is the number of observed occurrences where a_i was not optimal, and $C(\phi_j, a_i)$ is the number of occurrences where $\phi_j = 1$ and a_i was optimal. More formally:

$$C(a_i) = \sum_{w \in W} \sum_{s \in w} (a_i \in \pi(s)) \tag{10}$$

$$C(\bar{a_i}) = \sum_{w \in W} \sum_{s \in w} (a_i \notin \pi(s)) \tag{11}$$

$$C(\phi_j, a_i) = \sum_{w \in W} \sum_{s \in w} (a_i \in \pi(s) \wedge \phi_j == 1) \tag{12}$$

We determined optimality using the inferred policy, π.

During the learning phase, the agent learns which actions are useful under different conditions. More formally, it learns the model parameter θ for the distribution on the optimality of each action (i.e. Equation 5), given the features $\phi_1, \ldots, \phi_n$. For example, consider the smelting task pictured in Figure 2. During training, we observe that the `destroyBlock` action is often optimal when the agent is looking at a block of gold and the agent was trying to smelt gold. Likewise, when the agent was *not* looking at a block of gold or was attempting to satisfy a different goal, then the `destroyBlock` action was generally not optimal. This information informs the distribution over the optimality of the `destroyBlock` action given the set of features, which is used at test time to suggest that the agent ought to destroy gold blocks (when trying to smelt gold), but not arbitrary other blocks.

Affordance-Aware Planning

A planner uses affordances to prune actions in a state- and goal-specific way.

First, affordances may be specified by a domain expert. The expert specifies a set of actions associated with a precondition-goal type pair. When the affordance is active, the actions suggested by the affordance are included in the agent's action set. For instance, if an agent is standing above a block of buried gold and is trying to smelt a block of gold, then an expert may indicate that the agent should consider the actions of looking down and digging. All actions contributed by active affordances are grouped to yield the set of actions to consider for each state. Table 1 shows several examples of expert-defined affordances.

Figure 2: A gold smelting task in the Minecraft domain. The agent's goal is to mine a block of gold, move to the forge and then smelt the gold in the forge to produce gold ingots.

Precondition	Goal Type	Actions
`lookingTowardGoal`	`atLocation`	`{move}`
`lavaInFront`	`atLocation`	`{rotate}`
`lookingAtGold`	`hasGoldOre`	`{destroy}`

Table 1: Examples of expert-provided affordances.

Second, actions may be pruned by thresholding the posterior in Equation 5. In this method, the affordances remove any actions whose probability of being optimal is below the provided threshold for each state. The threshold was determined empirically, and was set to $\frac{0.2}{|\mathcal{A}|}$, where $|\mathcal{A}|$ is the size of the full action space of the OO-MDP. This threshold is quite conservative, and means that our approach only prunes actions which are extremely unlikely to be optimal.

Through the use of any of the above methods, an affordance-aware planner prunes actions on a state-by-state basis, focusing the agent on relevant action possibilities of the environment, consequently reducing planning time. Any planner operating in an OO-MDP may be made affordance-aware with this approach.

In a recent review on the theory of affordances, Chemero [7] suggests that an affordance is a relation between the features of an environment and an agent's abilities. Our approach grounds this interpretation, where the features of the environment correspond to the goal-dependent state features, ϕ, and the agent's abilities correspond to the OO-MDP action set. In our model, there is an affordance for each δ_j, with preconditions $\delta_j.p$, goal type $\delta_j.g$ and action distribution $\Pr(a_i \in \mathcal{A}^* | \phi_j, \theta)$, which is computed in our Naive Bayes model by marginalizing over all the features not associated with ϕ_j.

Results

We evaluate our approach using the game Minecraft and a collaborative robotic cooking task. Minecraft is a 3-D blocks game in which the user can place, craft, and destroy blocks of different types. Minecraft's physics and action space allow users to create complex systems, including logic gates and functional scientific graphing calculators[1]. Minecraft serves as a model for robotic tasks such as cooking assistance, assembling items in a factory, object retrieval, and complex terrain traversal. As in these tasks, the agent operates in a very large state-action space in an uncertain environment. Figure 2 shows a scene from one of our Minecraft problems. Additionally, we implemented an affordance based planner to enable a manipulator robot to infer helpful actions in response to a person working on a kitchen task, shown in Figure 1.

[1]https://www.youtube.com/watch?v=wgJfVRhotlQ

Minecraft Tests

Our experiments consisted of five common tasks in Minecraft, including constructing bridges over trenches, smelting gold, tunneling through walls, basic path planning, and digging to find an object. We tested on randomized worlds of varying size and difficulty. The generated test worlds varied in size from tens of thousands of states to hundreds of thousands of states. The agent learned affordances from a training set consisting of 20 simple state spaces of each map type (100 total maps), each approximately a 1,000-10,000 state world. We conducted all tests with a single knowledge base.

We use Real Time Dynamic Programming (RTDP) [4] as our baseline planner, a sampling-based algorithm that does not require the planner to visit all states. We compare RTDP with learned affordance-aware RTDP (LRTDP), and expert-defined affordance-aware RTDP (ERTDP). We terminated each planner when the maximum change in the value function was less than 0.01 for 100 consecutive policy rollouts, or the planner failed to converge after 1000 rollouts. The reward function was -1 for all transitions, except transitions to states in which the agent was in lava, where we set the reward to -10. The goal was set to be terminal. The discount factor was $\lambda = 0.99$. To introduce nondeterminism into our problem, movement actions (move, rotate, jump) in all experiments had a small probability (0.05) of incorrectly applying a different movement action. This noise factor approximates noise faced by a physical robot that attempts to execute actions in a real-world domain.

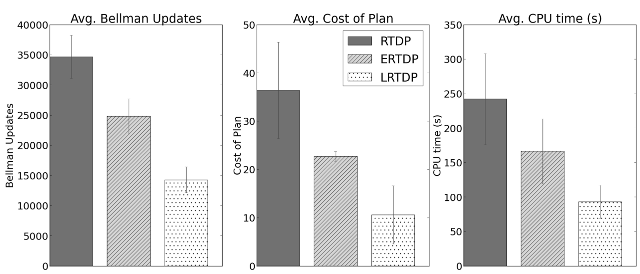

Figure 3: Average results from all maps.

We report the number of Bellman updates executed by each planning algorithm, the accumulated reward of the average plan, and the CPU time taken to find a plan. Table 2 shows the average Bellman updates, accumulated reward, and CPU time for RTDP, LRTDP and ERTDP after planning in 20 different maps of each goal type (100 total). Figure 3 shows the results averaged across all maps.

Because the planners were forced to terminate after only 1000 rollouts, they did not always converge to the optimal policy. LRTDP on average found a comparably better plan (10.6 cost) than ERTDP (22.7 cost) and RTDP (36.4 cost), found the plan in significantly fewer Bellman updates (14287.5 to ERTDP's 24804.1 and RTDP's 34694.3) and in less CPU time (93.1s to ERTDP's 166.4s and RTDP's 242.0s). These results indicate that while learned affordances gave the largest improvements, expert-provided affordances can also significantly enhance performance, and, depending on the domain, could add significant value in making large state-spaces tractable without the overhead of supplying training worlds.

Planner	Bellman	Reward	CPU
Mining Task			
`RTDP`	17142.1 (±3843)	**-6.5** (±1)	**17.6s** (±4)
`ERTDP`	14357.4 (±3275)	**-6.5** (±1)	31.9s (±8)
`LRTDP`	**12664.0** (±9340)	-12.7 (±5)	33.1s (±23)
Smelting Task			
`RTDP`	30995.0 (±6730)	**-8.6** (±1)	45.1s (±14)
`ERTDP`	28544.0 (±5909)	**-8.6** (±1)	72.6s (±19)
`LRTDP`	**2821.9** (±662)	-9.8 (±2)	**7.5s** (±2)
Wall Traversal Task			
`RTDP`	45041.7 (±11816)	-56.0 (±51)	**68.7s** (±22)
`ERTDP`	32552.0 (±10794)	-34.5 (±25)	96.5s (±39)
`LRTDP`	**24020.8** (±9239)	**-15.8** (±5)	80.5s (±34)
Trench Traversal Task			
`RTDP`	16183.5 (±4509)	**-8.1** (±2)	53.1s (±22)
`ERTDP`	**8674.8** (±2700)	-8.2 (±2)	**35.9s** (±15)
`LRTDP`	11758.4 (±2815)	-8.7 (±1)	57.9s (±20)
Plane Traversal Task			
`RTDP`	52407 (±18432)	-82.6 (±42)	877.0s (±381)
`ERTDP`	32928 (±14997)	-44.9 (±34)	505.3s (±304)
`LRTDP`	**19090** (±9158)	**-7.8** (±1)	**246s** (±159)

Table 2: RTDP vs. affordance-aware alternatives.

For some task types, LRTDP found a slightly worse plan on average than vanilla RTDP (e.g. the mining task). This is likely due to the fact that LRTDP occasionally prunes away actions that are essential for solving a particular problem. To fix this, we could lower the threshold to allow for even more conservative action pruning at the cost of performance. In future work, we plan on investigating more principled approaches to dynamically adjusting the threshold based on planning feedback.

Temporally Extended Actions and Affordances

We compared our approach to Temporally Extended Actions: macro-actions and options. We conducted these experiments with the same configurations as our Minecraft experiments. Domain experts provided the option policies and macro-actions.

Planner	Bellman	Reward	CPU
`RTDP`	27439 (±2348)	-22.6 (±9)	107 (±33)
`LRTDP`	**9935** (±1031)	**-12.4** (±1)	**53** (±5)
`RTDP+Opt`	26663 (±2298)	-17.4 (±4)	129(±35)
`LRTDP+Opt`	**9675** (±953)	**-11.5** (±1)	**93** (±10)
`RTDP+MA`	31083 (±2468)	-21.7 (±5)	336 (±28)
`LRTDP+MA`	**9854** (±1034)	**-11.7** (±1)	**162** (±17)

Table 3: Affordances vs. Temporally Extended Actions

Table 3 indicates the results of comparing RTDP equipped with macro actions, options, and affordances across 100 different executions in the same randomly generated Minecraft worlds. The results are averaged across tasks of each type presented in Table 2. Both macro-actions and options add a significant amount of time to planning. This increase is because it is computationally expensive to predict the expected reward associated with applying an option or a macro-action. Furthermore, the branching factor of the state-action space significantly increases when augmented with additional actions, causing the planner to run for longer and perform more Bellman updates. With affordances, the planner found a better plan in less CPU time, and with fewer Bellman updates. These results support the claim that affordances can handle the augmented action space provided by temporally extended actions by pruning away unnecessary actions, and that options and affordances provide complementary information.

ERTDP and Baxter

To assess affordances applied to a real-world robotic task, we devised a cooking domain that requires the robot to choose helpful actions for a person following a recipe. The robot reasons over ingredients, containers, appliances, working spaces and utensils. The action set consists of pouring containers, mixing ingredients and moving containers and tools, encapsulating a large set of grounded actions for the objects in a kitchen.

To help a person cook, a robot needs to plan across possible actions a person might take as well as possible actions the robot could take. Coupled with the large number of items in a kitchen, a planner in this complicated world must take advantage of knowledge inherent to the domain and to the recipe with which it is working. Using four ingredients required for brownies and the containers necessary for the ingredients, our cooking state space has 4.73×10^7 states. To remove a large number of self-transitions, we provided affordances for each action. For example, one affordance specified that mixing should only occur in containers that contain ingredients. An affordance for pouring required the container being poured to contain ingredients.

We divided a brownie recipe into three subgoals: combining and mixing the dry ingredients, combining and mixing the wet ingredients, and combining these two mixtures into a batter. For each subgoal, we provided more affordances specific to the objects used in that subgoal, like a whisk should only be used to mix wet ingredients. We used ERTDP to search for the least-cost plan to complete the recipe. The robot inferred actions such as handing off the whisk to the person to mix the wet ingredients.

In table 4 we show the comparison between standard RTDP and ERTDP for the three subgoals. Because the number of states and actions is reduced significantly, ERTDP can plan successfully in a short amount of time. Standard RTDP always encountered the maximum number of rollouts specified at the maximum depth each time.

We provide a video [2] showing how this affordance-based planner running on a robot can help a person cook by dynamically replanning through constant observations. After observing the placement of a cocoa container in the robot's workspace, the robot fetches a wooden spoon to allow the person to mix. After observing an egg container, the robot fetches a whisk to help beat the eggs. Because we modeled the culinary world as an MDP and replanned at every observation, the robot dynamically resolves failures and accounts for unpredictable user actions. In the video, the robot fails to grasp the wooden spoon on the first attempt and must retry the grasp after it observed no state change.

[2]See at https://vimeo.com/106226282

Planner	Bellman	Reward	CPU
Dry Ingredients			
`RTDP`	20000 ($\pm$0)	-123.1 ($\pm$0)	56.0s ($\pm$2.9)
`ERTDP`	**2457.2** ($\pm$53.2)	**-6.5** ($\pm$0)	**10.1s** ($\pm$0.3)
Wet Ingredients			
`RTDP`	19964 ($\pm$14.1)	-123.0 ($\pm$0)	66.6s ($\pm$9.9)
`ERTDP`	**5873.5** ($\pm$53.7)	**-6.5** ($\pm$0)	**15.6s** ($\pm$1.2)
Brownie Batter			
`RTDP`	20000 ($\pm$0)	-123.4 ($\pm$0.7)	53.3s ($\pm$2.4)
`ERTDP`	**6642.4** ($\pm$36.4)	**-7.0** ($\pm$0)	**31.9s** ($\pm$0.4)

Table 4: RTDP vs. Affordance-aware ERTDP for robotic kitchen tasks

Conclusion

We proposed a novel approach to represent transferable planning knowledge in terms of *affordances* [12]. Affordances allow an agent to efficiently prune actions based on learned or expert provided knowledge, significantly reducing the number of state-action pairs the agent needs to evaluate in order to act near optimally. We demonstrated the effectiveness of the affordance model by comparing RTDP to its affordance-aware equivalents in a series of challenging planning tasks in the Minecraft domain. Further, we designed a learning process that allows an agent to autonomously learn useful affordances that may be used across a variety of task types, reward functions, and state spaces, allowing for convenient extensions to robotic applications. Additionally, we compared the effectiveness of augmenting planners with affordances with temporally extended actions, and the combination of the two. The results suggest that affordances may be combined with temporally extended actions to provide improvements in planning. Lastly, we deployed an affordance-aware planner on a robot in a collaborative cooking task.

In the future, we hope to automatically discover useful state space specific subgoals online - a topic of some active research [20, 9]. Automatic discovery of subgoals would allow affordance-aware planners to take advantage of the goal-oriented focus of affordances, and would further reduce the size of the explored state-action space by improving the effectiveness of action pruning.

Additionally, we hope to explore additional methods that capitalize on the distribution over optimal actions, such as incorporating affordances with a forward search sparse sampling algorithm [29], or replacing the Naive Bayes model with a more sophisticated model, such as Logistic Regression or a Noisy-OR. We are also investigating methods of learning the threshold value in a more principled way - one such approach is to initialize the planner with a strict threshold, and slowly relax the threshold until a near optimal policy is found. These methods will enable agents to acquire and leverage general planning knowledge to infer actions in very large state spaces.

References

[1] D. Andre and S.J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth national conference on Artificial intelligence*, pages 119–125. American Association for Artificial Intelligence, 2002.

[2] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to control search in a forward chaining planner. In *In Proceedings of the 3rd European Workshop on Planning*, pages 141–153. Press, 1995.

[3] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:2000, 1999.

[4] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[5] Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. Interpreting and executing recipes with a cooking robot. In *Proceedings of International Symposium on Experimental Robotics (ISER)*, 2012.

[6] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005.

[7] Anthony Chemero. An outline of a theory of affordances. *Ecological psychology*, 15(2):181–195, 2003.

[8] T. Croonenborghs, K. Driessens, and M. Bruynooghe. Learning relational options for inductive transfer in relational reinforcement learning. *Inductive Logic Programming*, pages 88–97, 2008.

[9] Özgür Şimşek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 816–823, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102454. URL `http://doi.acm.org/10.1145/1102351.1102454`.

[10] C. Diuk, A. Cohen, and M.L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, 2008.

[11] Kutluhan Erol, James Hendler, and Dana S Nau. Htn planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128, 1994.

[12] JJ Gibson. The concept of affordances. *Perceiving, acting, and knowing*, pages 67–82, 1977.

[13] Eric A Hansen and Shlomo Zilberstein. Solving markov decision problems using heuristic search. In *Proceedings of AAAI Spring Symposium on Search Techniques from Problem Solving under Uncertainty and Incomplete Information*, 1999.

[14] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229. Morgan Kaufmann Publishers Inc., 1998.

[15] Nicholas K. Jong. The utility of temporal abstraction in reinforcement learning. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[16] Ross A. Knepper, Stefanie Tellex, Adrian Li, Nicholas Roy, and Daniela Rus. Single assembly robot in search of human partner: Versatile grounded language generation. In *Proceedings of the HRI 2013 Workshop on Collaborative Manipulation*, 2013.

[17] G. Konidaris and A. Barto. Efficient skill learning using abstraction selection. In *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*, pages 1107–1112, 2009.

[18] G. Konidaris, I. Scheidwasser, and A. Barto. Transfer in reinforcement learning via shared features. *The Journal of Machine Learning Research*, 98888:1333–1371, 2012.

[19] George Konidaris and Andrew Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI '07, pages 895–900, January 2007.

[20] Amy Mcgovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *In Proceedings of the eighteenth international conference on machine learning*, pages 361–368. Morgan Kaufmann, 2001.

[21] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 968–973, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1624312.1624357`.

[22] M Newton, John Levine, and Maria Fox. Genetically evolved macro-actions in ai planning problems. *Proceedings of the 24th UK Planning and Scheduling SIG*, pages 163–172, 2005.

[23] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[24] Balaraman Ravindran and Andrew Barto. An algebraic approach to abstraction in reinforcement learning. In *Twelfth Yale Workshop on Adaptive and Learning Systems*, pages 109–144, 2003.

[25] Benjamin Rosman and Subramanian Ramamoorthy. What good are actions? accelerating learning using learned action priors. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–6. IEEE, 2012.

[26] A.A. Sherstov and P. Stone. Improving action selection in mdp's via knowledge transfer. In *Proceedings of the 20th national conference on Artificial Intelligence*, pages 1024–1029. AAAI Press, 2005.

[27] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[28] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, 2008.

[29] Thomas Walsh, Sergiu Goschin, and Michael Littman. Integrating sample-based planning and model-based reinforcement learning, 2010. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1880`.