Affordances as Transferable Knowledge for Planning Agents

Gabriel Barth-Maron* and **David Abel*** and **James MacGlashan** and **Stefanie Tellex**
{gabriel_barth-maron, david_abel, james_macglashan, stefanie_tellex}@brown.edu
Brown University, Computer Science Dept.
115 Waterman Street, 4th floor
Providence, RI 02912-1910

Abstract

Robotic agents often map perceptual input to simplified representations that do not reflect the complexity and richness of the world. This simplification is due in large part to the limitations of planning algorithms, which fail in large stochastic state spaces on account of the well-known "curse of dimensionality." Existing approaches to address this problem fail to prevent autonomous agents from considering many actions which would be obviously irrelevant to a human solving the same problem. We formalize the notion of affordances as knowledge added to an Markov Decision Process (MDP) that prunes actions in a state- and reward- general way. This pruning significantly reduces the number of state-action pairs the agent needs to evaluate in order to act near-optimally. We demonstrate our approach in the Minecraft domain as a model for robotic tasks, showing significant increase in speed and reduction in state-space exploration during planning. Further, we provide a learning framework that enables an agent to learn affordances through experience, opening the door for agents to learn to adapt and plan through new situations. We provide preliminary results indicating that the learning process effectively produces affordances that help solve an MDP faster, suggesting that affordances serve as an effective, transferable piece of knowledge for planning agents in large state spaces.

Introduction

As robots move out of the lab and into the real world, planning algorithms need to scale to domains of increased noise, size, and complexity. A classic formalization of this problem is a stochastic sequential decision making problem in which the agent must find a policy (a mapping from states to actions) for some subset of the state space that enables the agent to achieve a goal from some initial state, while minimizing any costs along the way. Increases in planning problem size and complexity directly correspond to an explosion in the state-action space, restricting extensions to large state-spaces such as robotic applications. Current approaches to solving sequential decision making problems in the face of uncertainty cannot tackle these problems as the state-action space becomes too large (Grounds and Kudenko 2005).

To address this state-space explosion, prior work has explored adding knowledge to the planner to solve problems in these massive domains, such as options (Sutton, Precup, and Singh 1999) and macroactions (Botea et al. 2005; Newton, Levine, and Fox 2005). However, these approaches add knowledge in the form of additional high-level actions to the agent, which *increases* the size of the state-action space (while also allowing the agent to search more deeply within the space). The resulting augmented space is even larger, which can have the paradoxical effect of increasing the search time for a good policy (Jong 2008). Further, other approaches fall short of learning useful, transferable knowledge, either due to complexity or lack of generalizability.

Instead, we propose a formalization of *affordances* (Gibson 1977) for MDPs that specifies which actions an agent should consider in different kinds of states to achieve a certain kind of goal. Our approach enables an agent to focus on aspects of the environment that are most relevant toward solving its current goal and avoids exploration of irrelevant parts of the state-action space, which leads to dramatic speedups in planning.

Further, we propose a learning process that enables agents to autonomously learn affordances through experience, lessening the agent's dependence on expert knowledge. Affordances are not specific to a particular reward function or state space, and provide the agent with transferable knowledge that is effective in a wide variety of problems. We call any planner that uses affordances an *affordance-aware* planner.

Because affordances define the *kind* of goals for which actions are useful, affordances also enable high-level reasoning that can be combined with approaches like high-level subgoal planning for even greater performance gains. In our current model, ideal subgoals are given directly to planning agents by an expert - however, we are interested in automatically discovering subgoals in an online way, a task which has already enjoyed some success (Mcgovern and Barto 2001; Şimşek, Wolfe, and Barto 2005).

This paper contains significant material from our previously published workshop paper (Abel et al. 2014). It focuses more on the robotics applications of our framework.

*The first two authors contributed equally to this paper.

Minecraft

We use Minecraft as our planning and evaluation domain. Minecraft is a 3-D blocks world game in which the user can place and destroy blocks of different types. It serves as a model for a variety of robotic tasks involving assembly and construction. Minecraft's physics and action space are expressive enough to allow very complex systems to be created by users, including logic gates and functional scientific graphing calculators[1]; simple scenes from a Minecraft world appear in Figure 1 - a video demonstration of an affordance-aware planner solving this task may be seen online[2]. Minecraft serves as a model for robotic tasks such as cooking assistance, assembling items in a factory, and object retrieval. As in these tasks, the agent operates in a very large state-action space in an uncertain environment.

Minecraft is also an effective parallel for the actual world, both in terms of approximating the complexity and scope of planning problems, as well as modeling the uncertainty and noise presented to a robotic agent. For instance, robotic agents are prone to uncertainty throughout their system, including noise in their sensors (cameras, LIDAR, microphones, etc.), odometry, control, and actuation. In order to accurately capture some of the inherent difficulties of planning under uncertainty, the Minecraft agent's actions were modified to have stochastic outcomes. These stochastic outcomes may require important changes in the optimal policy in contrast to deterministic actions, such as keeping the agent's distance from high cost areas of the state-space, such as lava or cliffs.

We chose to give the Minecraft agent perfect sensor data about the Minecraft world. However, affordances typically relate to the agent's immediate surroundings, so limiting the perceptual scope should not impede the performance gains of affordances. We have considered extensions to Partially Observable domains, though at a distance solving a POMDP is effectively unchanged by the presence of affordances (beyond the performance gains provided by pruning actions).

Object-Oriented Markov Decision Processes (OO-MDPs)

We define affordances in terms of propositional functions on states. Our definition builds on the OO-MDP (Diuk, Cohen, and Littman 2008). OO-MDPs are an extension of the classic MDP. A classic 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 and transitioning to state s'; and $\gamma \in [0, 1)$ is a discount factor that defines how much the agent prefers immediate rewards over distant rewards (the agent more greatly prefers to maximize more immediate rewards as γ decreases).

A classic way to provide a factored representation of an MDP state is to represent each MDP state as a single feature vector. By contrast, an OO-MDP represents the state space as 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. Upon instantiation of an object class, its attributes are given a state $o.state$ (an assignment of values to its attributes). The underlying MDP state is the set of all the object states: $s \in \mathcal{S} = \cup_{i=1}^{o} \{o_i.state\}$.

There are two advantages to using an object-oriented factored state representation instead of a single feature vector. First, different states in the same state space may contain different numbers of objects of varying classes, which is useful in domains like Minecraft in which the agent can dynamically add and remove blocks to the world. Second, MDP states can be defined invariantly to the specific object references. For instance, consider a Minecraft world with two block objects, b_1 and b_2. If the agent picked up and swapped the position of b_1 and b_2, the MDP state before the swap and after the swap would be the same, because the MDP state definition is invariant to which object holds which object state. This object reference invariance results in a smaller state space compared to representations like feature vectors in which changes to value assignments always result in a different state.

While the OO-MDP state definition is a good fit for the Minecraft domain, our motivation for using an OO-MDP lies in the ability to formulate 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.

While an OO-MDP reduces the size of the Minecraft state space by a significant factor, the resulting state space is still far too large to solve with any existing (OO)-MDP solver. This is the primary motivator for incorporating affordances - to reduce the amount of the state space that an OO-MDP agent will have to explore.

The Brown UMBC Reinforcement Learning And Planning framework (BURLAP[3]) is working toward integrating planning and reinforcement learning algorithms with a variety of planning domains, represented as an OO-MDP, including ROS. In this way, transferable knowledge like affordances can be quickly deployed to domains like Mountain Car (Moore and Hall 1990) and Minecraft, but also to a variety of Robots that utilize ROS. Our group is also working to deploy affordances as a means of knowledge representation and reasoning for collaborative cooking with ReThink's Baxter.

Related Work

In this section, we discuss the differences between affordance-aware planning and other forms of knowledge that have been used to accelerate planning.

Temporarily Extended Actions

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

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

[2] Watch at: https://vimeo.com/88689171

[3] http://burlap.cs.brown.edu/

succession. Two common forms of temporally extended actions are *macro-actions* (Hauskrecht et al. 1998) and *options* (Sutton, Precup, and Singh 1999). 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 hand 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 (Konidaris and Barto 2007; 2009; Ravindran and Barto 2003; Croonenborghs, Driessens, and Bruynooghe 2008; Andre and Russell 2002; Konidaris, Scheidwasser, and Barto 2012).

Given the potential for unhelpful temporally extended actions to negatively impact planning time (Jong 2008), we believe combing affordances with temporally extended actions may be especially valuable because it will restrict the set of temporally extended actions to those useful for a task. In the future, we plan to explore the benefit from combining these approaches.

Action Pruning

Sherstov and Stone (Sherstov and Stone 2005) considered MDPs with a very large action set and 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. The main difference between our affordance-based action set pruning and this action transfer work is that affordances prune away actions on a state by state basis, where as the learned action pruning is on per task level. Further, with lifted goal descriptions, affordances may be attached to subgoal planning for a significant benefit in planning tasks where complete subgoal knowledge is known.

Rosman and Ramamoorthy (Rosman and Ramamoorthy 2012) 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.

There are a few limitations of the actions priors work that affordance-aware planning does not possess: (1) the action priors can only be used with planning/learning algorithms that work well with an ϵ-greedy rollout policy; (2) the priors are only utilized for fraction ϵ of the time steps, which is typically quite small; and (3) as variance in tasks explored increases, the priors will become more uniform. In contrast, affordance-aware planning can be used in a wide range of planning algorithms, benefits from the pruned action set in every time step, and the affordance defined lifted goal-description enables higher-level reasoning such as subgoal planning.

Temporal Logic

Bacchus and Kabanza (Bacchus and Kabanza 1995; 1999) 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, STRIPS style planner may be guided through the search space by checking whether candidate plans do not falsify a given knowledge base of LTL formulas, often achieving polynomial time planning in exponential space.

The primary difference between this body of work and affordance-aware planning is that affordances may be learned (increasing autonomy of the agent), while LTL formulas are far too complicated to learn effectively, placing dependence on an expert.

Heuristics

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*, or *reward shaping*. Initializing the value function to an admissible close approximation of the optimal value function has been shown to be effective for LAO* and RTDP (Hansen and Zilberstein 1999).

Reward shaping is an alternative approach to providing heuristics. The planning algorithm uses a modified version of the reward function that returns larger rewards for state-action pairs that are expected to be useful, but does not guarantee convergence to an optimal policy unless certain properties of the shaped reward are satisfied (Ng, Harada, and Russell 1999).

A critical difference between heuristics and affordances is that heuristics are highly dependent on the reward function and state space of the task being solved, whereas affordances are state space independent and transferable between different reward functions. However, if a heuristic can be provided, the combination of heuristics and affordances may even more greatly accelerate planning algorithms than either approach alone.

Affordances

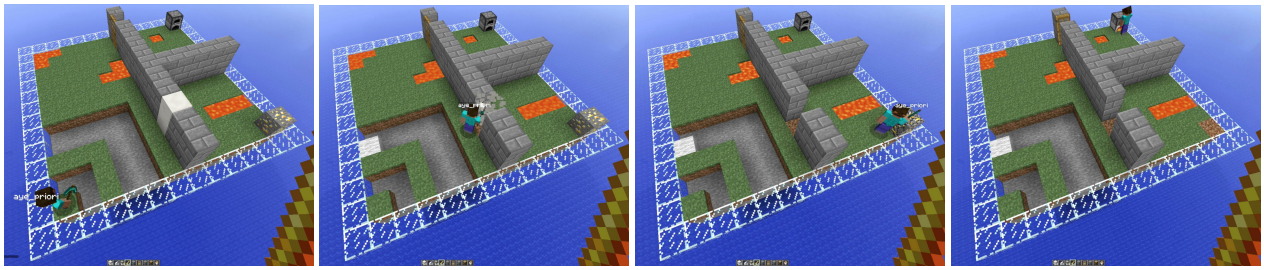
We define an affordance Δ as the mapping $\langle p, g \rangle \longmapsto \mathcal{A}'$, where:

$\mathcal{A}' \subseteq \mathcal{A}$, a subset of the action space, representing the relevant *action-possibilities* of the environment.

p is a predicate on states, $s \longrightarrow \{0, 1\}$ representing the *precondition* for the affordance.

g is an ungrounded predicate on states representing a *lifted goal description*.

The precondition and goal description refer to predicates that are defined in the OO-MDP definition. We call an affordance *activated* when its predicate is true and its lifted goal description g matches the agent's current goal. Using OO-MDP predicates for affordance preconditions and goal descriptions allows for state space independence. Thus, a planner equipped with affordances can be used in any number of different environments. For instance, the affordances defined for Minecraft navigation problems can be used in any task regardless of the spatial size of the world, number of blocks in the world, and specific goal location that needs to be reached.



(a) Start (b) Destroy Wall (c) Collect Ore (d) Smelt Ore

Figure 1: Affordance-aware Real-Time Dynamic Programming (RTDP) tasked with a gold-smelting task with a variety of obstacles (only solved by an affordance-aware planner)

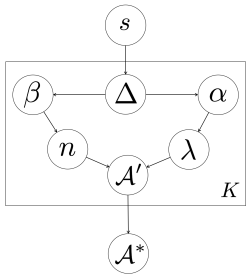


Figure 2: The full graphical model approximating a distribution over $\mathcal{A}^*$, the pruned action set for a given state s

Affordance-Aware Planning

We call any planner that uses affordances an *affordance-aware* planner. For a given state, our goal is to solve for the probability of getting a particular action set $\mathcal{A}^*$, and approximate sampling from this distribution. This ensures that in the limit, it is possible to apply each action in each state. $\mathcal{A}^*$ represents a drawn action subset from the OO-MDP action set that is likely to contain the optimal action(s) for a given state, but not suboptimal actions.

$$\Pr(\mathcal{A}^* \mid s, \Delta_1 \dots \Delta_K) \tag{1}$$

We let each affordance contribute a set $\mathcal{A}' \subseteq A^*$ in each state:

$$\Pr(\mathcal{A}'_1 \cup \dots \cup \mathcal{A}'_K \mid s, \Delta_1 \dots \Delta_K) \tag{2}$$

We approximate this term assuming the sets $\mathcal{A}'_i$ are disjoint:

$$\sum_i^K \Pr(\mathcal{A}'_i \mid s, \Delta_i) \tag{3}$$

Given a set of K domain affordances $Z = \{\Delta_1, ..., \Delta_K\}$ and a current agent goal condition defined with an OO-MDP predicate G, the action set that a planning algorithm considers is pruned on a state by state basis as shown in Algorithm 1. Each activated affordance contributes a suggested action set, determined by Algorithm 2.

Specifically, we prune actions on a state by state basis by initializing an empty set of actions $\mathcal{A}^*$ (line 1). The algorithm then iterates through each of the domain affordances (lines 2-6). If the affordance precondition ($\Delta.p$) is satisfied by some set of objects in the current state and the affordance goal condition ($\Delta.g$) is defined with the same predicate as the current goal (line 3), then the actions associated with the affordance ($\Delta.\mathcal{A}' = \Delta.getActions(s)$) are added to the action set $\mathcal{A}^*$ (line 4). Finally, $\mathcal{A}^*$ is returned (line 7).

Algorithm 1 getActionsForState($state$, Z, G)

1: $\mathcal{A}^* \leftarrow \{\}$
2: **for** $\Delta \in Z$ **do**
3: **if** $\Delta.p(state)$ and $\Delta.g = G$ **then**
4: $\mathcal{A}^* \leftarrow \mathcal{A}^* \cup \Delta.getActions(s)$
5: **end if**
6: **end for**
7: **return** $\mathcal{A}^*$

For each affordance, we get an action set $\mathcal{A}'$. This process is outlined by Algorithm 2. To compute $\mathcal{A}'$, we form a Dirichlet-multinomial distribution over actions (λ), and a Dirichlet distribution over the size (N) of each action set. Therefore, the probability of getting an action set from affordance i in state s is:

$$\Pr(\mathcal{A}'_i \mid s, \Delta_i) = \Pr(\mathcal{A}'_i \mid N, \lambda) = \Pr(\lambda \mid \alpha)\Pr(N \mid \beta) \tag{4}$$

For a given affordance Δ_i, first we sample from our distribution over action set size to get a candidate action set size (lines 1-2). We then take that many samples from our distribution over actions to get a candidate action set $\mathcal{A}'$ (lines 3-5).

$$\Pr(\lambda \mid \alpha) = DirMult(\alpha) \tag{5}$$
$$\Pr(N \mid \beta) = Dir(\beta) \tag{6}$$

Algorithm 2 $\Delta_i.getActions(s)$

1: $\lambda \leftarrow DirMult(\Delta_i.\alpha)$
2: $N \leftarrow Dir(\Delta_i.\beta)$
3: **for** 1 to N **do**
4: $\Delta_i.\mathcal{A}' \leftarrow \lambda$
5: **end for**
6: **return** $\Delta_i.\mathcal{A}'$

Through the use of Algorithms 1 & 2, any OO-MDP solver can be made *affordance-aware*. For a planner to be made affordance-aware, we require that an expert provide a set $\mathcal{P}$ of predicates for the domain of relevance (i.e. Minecraft). Additionally, the expert must specify a set $\mathcal{G} \subset \mathcal{P}$ that indicates which predicates may serve as goal conditions. If the expert wishes to provide the affordances directly, they must specify the Dirichlet parameters α and β. Note that in the limit, the expert may fix α and β in a way that forces a given affordance to always suggest a specific set of actions - all expert affordances given during experiments were provided in this form.

Learning Affordances

A strength of our affordance formalism is that it is simple to learn useful affordances directly. Furthermore, since affordances are not dependent on a specific reward function or state-space, affordances learned during training may be transferred to a wide variety of different tasks. This means that a single set of learned affordances could aid a robot in many different applications. Given the set of predicates $\mathcal{P}$ and possible goals $\mathcal{G} \subset \mathcal{P}$, we form a set of candidate affordances Δ with every combination of $\langle p, g \rangle$, for $p \in \mathcal{P}$ and $g \in \mathcal{G}$. To learn the action set for each of these candidate affordances, we propose a scaffolded learning process that computes α and β from the solved policy of m goal-annotated OO-MDPs that have small state spaces (small enough to be solved using tabular method), but still present similar sorts of features to the state spaces the agent might expect to see in more complex environments.

For each optimal policy, we count the number of policies that used each action when each affordance was activated. α is set to this count. Additionally, we define β as a vector of the integers 1 to $|\mathcal{A}|$. Then, for each optimal policy, we count the number of different actions that were optimal for each activated affordance Δ_i, and increment that value for $\Delta_i.\beta$. This captures how large or small optimal action sets are expected to be for each affordance.

Experiments

We conducted a series of experiments in the Minecraft domain that compared the performance of several OO-MDP solvers without affordances to their affordance-aware counterparts. We selected the expert affordances from our background knowledge of the domain and specified them so that each affordance always mapped to the same set of actions.

For the expert affordances, we gave the agent a knowledge base of 5 types of affordances, which are listed in Figure 3. Our experiments consisted of a variety of common tasks (state spaces 1-7 in Table 1) in Minecraft, ranging from basic path planning, to smelting gold, to opening doors and tunneling through walls. We also tested each planner on worlds of varying size and difficulty to demonstrate the scalability and flexibility of the affordance formalism.

Additionally, we tested our learning procedure with a set of 17 predicates[4], and compared the performance of

[4]Examples of which are: *is there a plane in front of the agent* and *is there a destructible wall behind the agent*.

$$\Delta_1 = \langle nearTrench, reachGoal \rangle \longmapsto \{place, jump\}$$
$$\Delta_2 = \langle onPlane, reachGoal \rangle \longmapsto \{move\}$$
$$\Delta_3 = \langle nearWall, reachGoal \rangle \longmapsto \{destroy\}$$
$$\Delta_4 = \langle nearFurnace, makeGold \rangle \longmapsto \{place\}$$
$$\Delta_5 = \langle nearOre, makeGold \rangle \longmapsto \{destroy\}$$

Figure 3: The five affordance types used in expert experiments.

RTDP solving the OO-MDP with (1) No affordances, (2) Learned affordances, and (3) Expert provided affordances. The training data consisted of 1000 simple state spaces, each a $3 \times 3 \times 3$ world with randomized features that mirrored the agent's actual state space. The same training data was used for each test state space.

The evaluation metric for each trial was the number of Bellman updates that were executed by each planning algorithm. Value Iteration (VI) was terminated when the maximum change in the value function was less than 0.01. RTDP terminated when the maximum change in the value function was less than 0.01 for five consecutive policy rollouts. In subgoal planning, we gave the agent a set of candidate subgoals for each task. From these subgoals, the high-level subgoal plan was solved using breadth-first search, which only took a small fraction of the time compared to the total low-level planning and therefore is not reported. In future work, we hope to explore automatic discovery of useful subgoals to further reduce the amount of knowledge given by an expert and to increase the effectiveness of affordances.

We set the reward function to -1 for all transitions, except transitions to states in which the agent was on lava, which returned -200. The goal was set to be terminal. The discount factor was set to $\lambda = 0.99$. For all experiments, actions associated with a direction (e.g. movement, block placement, jumping, etc.), had a small probability (0.3) of moving in another random direction.

Lastly, we conducted experiments in which we varied the number of training worlds used in the learning process from 0-1000 to demonstrate that planning performance improves with more training data. As in Table 2, we generated 0 to 1000 simple state spaces, each a $3 \times 3 \times 3$ world with randomized features that mirrored the agent's actual state space. We then solved the OO-MDP with training data of 0 to 1000 simple state spaces to demonstrate the effectiveness of added training data.

Results

Table 1 shows the number of Bellman updates required when solving the OO-MDP with conventional methods (left column) compared to solving the OO-MDP with an affordance-aware method (right column). The affordance-aware methods significantly outperformed their unaugmented counterparts in all of these experiments. These results, while unsurprising, concretely demonstrate that a small set of affordances prune away many useless actions across many different types of Minecraft tasks.

State Space	VI	A-VI	RTDP	A-RTDP	SG	A-SG
1	71604	**100**	836	**152**	1373	**141**
2	413559	**366**	4561	**392**	28185	**547**
3	1439883	**904**	18833	**788**	15583	**1001**
4	861084	**4368**	12207	**1945**	6368	**1381**
5	413559	**366**	4425	**993**	25792	**597**
6	203796	**105**	26624	**145**	5404	**182**
7	16406	**962**	7738	**809**	7412	**578**

Table 1: Expert Affordance Results: Avg. Number of Bellman Updates per converged policy. Where SG refers to the subgoal planner, and the "A-" prefix indicates the use of an affordance-aware planner.

State Space	No Affordances	Learned	Expert
`Tiny`	879	414	94
`Small`	1460	802	321
`Medium`	3993	2412	693
`Large`	8344	5100	1458

Table 2: Learned Affordance Results: Avg. Number of Bellman Updates per converged policy

Table 2 indicates the average number of Bellman updates required by RTDP to solve the OO-MDP in each of the four candidate worlds. The learned affordances clearly improved on standard RTDP by a significant margin, though there is clearly still room to improve the learning process to approach learned affordances that are near-expert level.

Figure 4 demonstrates the added effect of more training data. In these experiments, we tested on map types that mirrored the features of the worlds generated during training, but this process could be extended to allow for scaffolded learning and more complicated maps (such as the gold smelting task in Figure 1). The averages reported are from solving the OO-MDP 20 times for each world, with each knowledge base. There was a negligible difference in the quality of the policies generated.

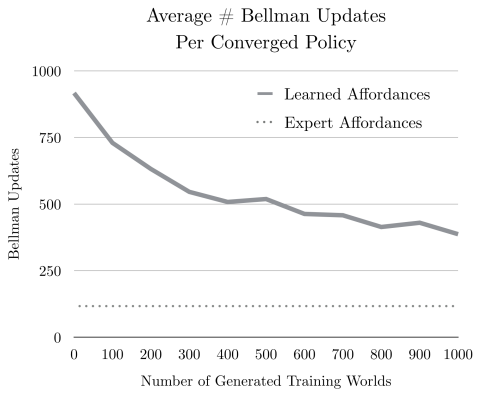


Figure 4: The effectiveness of added training data on planning performance.

Conclusion

We proposed a novel approach to representing transferable knowledge in terms of *affordances* (Gibson 1977) that allows an agent to efficiently prune its action space based on domain knowledge, providing a significant reduction in 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 standard paradigm planners to their affordance-aware equivalents in a series of challenging planning tasks in the Minecraft domain, a domain that is an apt model for many robotic tasks. Further, we designed a full 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. We provided preliminary results indicating the effectiveness of the learned affordances, suggesting that the agent may be able to learn to tackle new types of problems on its own.

In the future, we hope to increase our coverage of Minecraft to solve even more difficult planning problems, and apply this approach beyond the Minecraft domain and onto actual robots and other domains of interest. Additionally, in certain cases, we provide the agent with extensive state-space-specific subgoal knowledge that is essential in solving some difficult planning problems. We hope to automatically discover useful subgoals - a topic of some active research (Mcgovern and Barto 2001; Şimşek, Wolfe, and Barto 2005). This will allow for affordances to reduce the size of the explored state-action space without requiring knowledge from an expert, increasing transferability across task types.

References

Abel, D.; Barth-Maron, G.; MacGlashan, J.; and Tellex, S. 2014. Toward affordance-aware planning. In *First Workshop on Affordances: Affordances in Vision for Cognitive Robotics*.

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

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

Bacchus, F., and Kabanza, F. 1999. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:2000.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24:581–621.

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

Şimşek, O.; Wolfe, A. P.; and Barto, A. G. 2005. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, 816–823. New York, NY, USA: ACM.

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

Gibson, J. 1977. The concept of affordances. *Perceiving, acting, and knowing* 67–82.

Grounds, M., and Kudenko, D. 2005. Combining reinforcement learning with symbolic planning. In *Proceedings of the 5th, 6th and 7th European conference on Adaptive and learning agents and multi-agent systems: adaptation and multi-agent learning*, ALAS '05.

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

Hauskrecht, M.; Meuleau, N.; Kaelbling, L. P.; Dean, T.; and Boutilier, C. 1998. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, 220–229. Morgan Kaufmann Publishers Inc.

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

Konidaris, G., and Barto, A. 2007. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI '07, 895–900.

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

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

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

Moore, A. W., and Hall, T. 1990. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory.

Newton, M.; Levine, J.; and Fox, M. 2005. Genetically evolved macro-actions in ai planning problems. *Proceedings of the 24th UK Planning and Scheduling SIG* 163–172.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, 278–287.

Ravindran, B., and Barto, A. 2003. An algebraic approach to abstraction in reinforcement learning. In *Twelfth Yale Workshop on Adaptive and Learning Systems*, 109–144.

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

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

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