Affordance-Aware Planning

David Abel & Gabriel Barth-Maron, James MacGlashan, Stefanie Tellex
Department of Computer Science, Brown University
{dabel,gabrielbm,jmacglashan,stefie10}@cs.brown.edu

***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 address this problem fail to prevent autonomous agents from considering many actions which would be obviously irrelevant to a human solving the same problem. To address this problem, we introduce a new type of knowledge that can be added to a Markov Decision Process (MDP) for pruning actions in a state- and reward-general way, which we term *affordances*. This pruning reduces the number of state-action pairs the agent needs to evaluate in order to behave in a near optimal way. Further, we show that an agent can learn affordances through experience, and that learned affordance knowledge can equal or surparpass the performance of expert-provided affordances. 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, as well as applying it to the problem of a real-world robotic cooking assistant.**

I. Introduction

Robots operating in unstructured environments face a very large stochastic planning problem when carrying out complex tasks such as cooking [4] or assembling objects [13]. 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 [9].

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 [22] and macroactions [5, 17]. 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 [12]. Other approaches fall short of learning useful, transferable knowledge, either due to complexity or lack of generalizability.

Instead, we propose a formalization of *affordances* [8] for Markov Decision Processes (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 created 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.

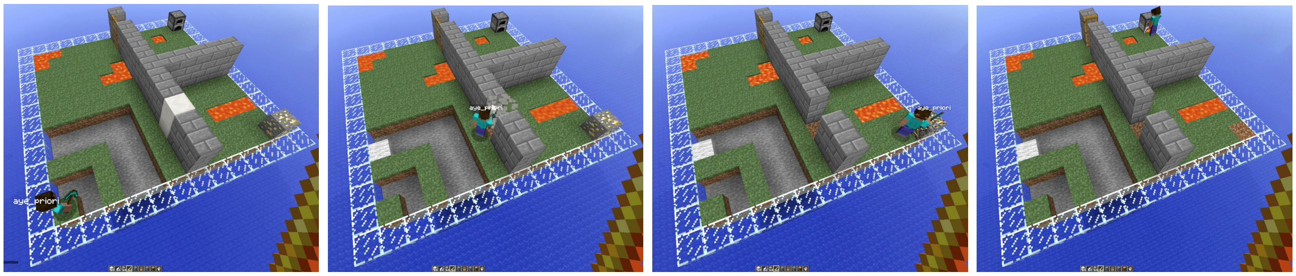
Because affordances define the *kind* of goals for which actions are useful, affordances also enable high-level reasoning that can be combined with approaches like subgoal planning for even greater performance gains. In our current model, ideal subgoals are sometimes given directly to planning agents by an expert - however, we are interested in automatically discovering subgoals in an online way, a problem which has already enjoyed some success [**? ?**].

II. Background

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 early iteration 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

[1]https://www.youtube.com/watch?v=wgJfVRhotlQ
[2]Watch at: https://vimeo.com/88689171



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

Fig. 1. Affordance-aware RTDP tasked with a gold-smelting task with a variety of obstacles (only solved by an affordance-aware planner)

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.

D: Should put an estimate of the state space size (or in experiments for test worlds)

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).

A. *OO-MDPs*

We define affordances in terms of propositional functions on states. Our definition builds on the Object-Oriented Markov Decision Process (OO-MDP) [7]. OO-MDPs are an extension of the classic Markov Decision Process (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: $s \in \mathcal{S} = \cup_{i=1}^{o} \{o_i.state\}$.

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 [**?**] 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.

III. 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.

A. *Temporally 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 succession. Two common forms of temporally extended actions are *macro-actions* [11] and *options* [22]. 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 [16, 14, 19, 6, 1, 15].

Given the potential for unhelpful temporally extended actions to negatively impact planning time [12], we believe

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

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. We conducted a set of experiments to investigate this intuition.

B. Hierarchical Task Networks

D: I think we should have a shoutout to Branavan's Learning High Level Plans from Text paper in this section (and include subgoal planning as part of this section

E: I've been writing traditional as I expect we'll discover some HTNs that grapple with the issues stated below – which we should probably cite Traditional Hierarchical Task Networks (HTNs) employ *task decompositions* to aid in planning. The goal at hand is decomposed into smaller tasks which are in turn decomposed into smaller tasks. This decomposition continues until primitive tasks that are immediately achievable 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 both HTNs and affordances fulfill the same role: both achieve action pruning by exploiting some form of supplied knowledge. HTNs do so with the use of information regarding both the task decomposition of the goal at hand and the sorts constraints that said decomposition imposes upon the planner. Similarly, affordances require knowledge as to how to extract values for propositional functions of interest by querying the state.

However there are three of essential distinctions between affordances and traditional HTNs. (1) HTNs deal exclusively with deterministic domains as opposed to the stochastic spaces with which affordances grapple. As a result they produce plans and not policies. (2) Moreover, HTNs do not incorporate reward into their planning. Consequently, they lack mathematical guarantees of optimal planning. **E: I think.. We should double check this.** (3) On a qualitative level, the degree of supplied knowledge in HTNs surpasses that of affordances: whereas affordances simply require relevant propositional functions, HTNs require not only constraints for sub-tasks but a hierarchical framework of arbitrary complexity. Thus, despite a superficial similarity between affordances and HTNs wherein both employ supplied knowledge, the two deal with disparate forms of planning problems; HTN's planning problem is deterministic, reward-agnostic and allows for a plethora of knowledge while affordances solve a planning problem that is stochastic, reward-aware and permits only relatively basic knowledge about the domain.

C. Action Pruning

Sherstov and Stone [21] 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 [20] 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.

D. Temporal Logic

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, 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.

E. 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 [10].

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 [18].

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.

IV. 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 the agent is trying to satisfy.

A. 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 place a Dirichlet Multinomial prior on the action selection for a given state to maximize this probability. 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.

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

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

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

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** $\mathcal{A}^*$

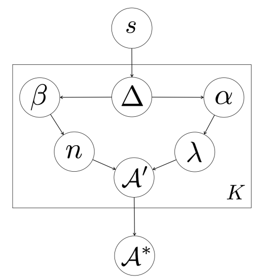


Fig. 2. The full graphical model approximating a distribution over $\mathcal{A}^*$, the pruned action set for a given state s

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:

$$\Pr(\mathcal{A}'_i \mid s, \Delta_i) = \Pr(\mathcal{A}'_i \mid N_i, \lambda_i) = \Pr(\lambda_i \mid \alpha_i)\Pr(N_i \mid \beta_i) \tag{4}$$

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 - this type of expert affordance was given for all experiments.

B. Learning Affordances

A strength of our affordance formalism is that it is simple to learn useful affordances directly. 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, 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. Additionally, we provide two types of affordances that may be learned through this process.

The first prunes actions in a probabilistic way, maintaining the optimality guarantees of each planner in the limit, while the other prunes actions in a deterministic way, sacrificing the optimality guarantees for a boost in planning time. We call the probabilistic affordances 'soft', and the deterministic affordances, 'hard'. Hard affordances define their action set by pruning away actions provided by the learning process whose probability mass given by the prior was lower than 1%, meaning that the action was optimal less than 1% of the time. All expert affordances were defined to be 'hard'.

V. 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 a set of expert affordances from our background knowledge of the domain. Each expert affordances was defined to be hard. Additionally, we ran our full learning process and learned soft and hard affordances for each task. We compared standard paradigm planners (Real Time Dynamic Programming and Value Iteration) with their expert-affordance-aware counterparts and with their learned-affordance-aware counterparts.

For the expert affordances, we gave the agent a knowledge base of 17 affordances, which are listed in Figure **??**. Our experiments consisted of a variety of common tasks (state spaces 1-5 in Table **??**) in Minecraft, ranging from constructing bridges over trenches, to smelting gold, to tunneling through walls, and constructing towers. We also tested each planner on worlds of varying size and difficulty to demonstrate the scalability and flexibility of the affordance formalism.

For the learning process, the training data consisted of 125 simple state spaces, each a $2 \times 3 \times 4$ 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, as well as the CPU time taken to find a plan. Value Iteration 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 ten consecutive policy rollouts. We set the reward function to -1 for all transitions, except transitions to states in which the agent was on lava, which returned -10. 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, block destruction, etc.), had a small probability (0.3) of applying in another random direction.

Further, 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 as the agent learns more. As in Table **??**, 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.

Lastly, we compared our approach to Temporally Extended Actions: Macroactions and Options. We tested RTDP and VI with just expert affordances, expert Macroactions, and expert Options, as well as the combination of affordance, macro actions, and options. We conducted these experiments in a variety of domains, both in randomly generated grid-worlds, and in randomly generated Minecraft problems similar to those problem types discussed above (path planning, bridge building, gold smelting, etc.). The option policies and macro actions provided were hand coded by domain experts.

VI. Results

A. HTN/TLPlan Comparison

State Space	JSHOP2	Affordances
10 blocks	-	-
100 blocks	-	-
200 blocks	-	-
300 blocks	-	-
400 blocks	-	-
500 blocks	-	-

TABLE I
Blocksworld Results: Number of states explored to find optimal plan

B. Baxter

Fig. 3. Placeholder for baxter results/image

C. Options

State Space	None	Options	Affordances	Both
`4rooms`	-	-	-	-
`Doors`	-	-	-	-
`Small`	-	-	-	-
`Medium`	-	-	-	-
`Large`	-	-	-	-

TABLE II
OPTIONS VS. AFFORDANCES: CPU TIME PER CONVERGED POLICY

D. Minecraft: Expert vs Learned vs None

D: These are preliminary results and will not be included in the final. I will run experiments on more and larger worlds (currently showing average after planning in 5 worlds per task type - worlds were 2x3x4, I'll run on 8x8x8).

State Space	RTDP	Learned Soft	Learned Hard	Expert
`Trench`	2502	2804	2263	**1437**
`Mining`	1063	1428	**724**	894
`Smelting`	2657	3149	**2174**	2575
`Wall`	3004	3409	**2192**	2420
`Tower`	4191	3617	**3485**	4402

TABLE III
LEARNED AFFORDANCE RESULTS: AVG. NUMBER OF BELLMAN UPDATES PER CONVERGED POLICY (AVERAGE OVER 5 WORLDS PER GOAL TYPE)

State Space	RTDP	Learned Soft	Learned Hard	Expert
`Trench`	0.96s	1.17s	0.77s	**0.47s**
`Mining`	0.34s	0.54s	**0.21s**	0.26s
`Smelting`	0.91s	1.25s	**0.70s**	0.81s
`Wall`	1.12s	1.49s	**0.78s**	0.85s
`Tower`	0.95s	1.04s	**0.78s**	0.88s

TABLE IV
LEARNED AFFORDANCE RESULTS: AVG. CPU TIME PER CONVERGED POLICY (AVERAGE OVER 5 WORLDS PER GOAL TYPE)

E. Minecraft: Learning rate

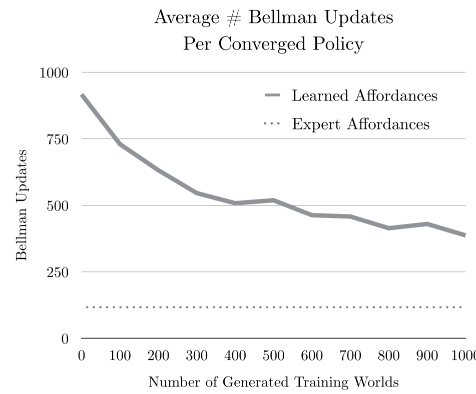


Fig. 4. Placeholder - will recollect this data given recent updates

VII. CONCLUSION

We proposed a novel approach to representing transferable knowledge in terms of *affordances* [8] 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 optimally. We demonstrated the effectiveness of the affordance model by comparing standard MDP solvers to their affordance-aware equivalent in a series of challenging planning tasks in the Minecraft. domain. 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 results indicating the effectiveness of the learned affordances, suggesting that the agent may be able to discover novel affordance types and learn to tackle new types of problems on its own. Lastly, we compared the effectiveness of augmenting planners with affordances to augmenting with temporally extended actions, as well as providing both to a planner. The results suggest that affordances, when combined with temporally extended actions, provide substantial reduction in the portion of the state-action space that needs to be explored.

In the future, we hope to automatically discover useful subgoals - a topic of some active research [**?** **?**]. This will allow for affordances to reduce the size of the explored state-action space without requiring knowledge from an expert and increasing transferability across task types. Additionally, we hope to decrease the amount of knowledge given to the planner by implementing Incremental Feature Dependency Discovery [**?**], which will allow our affordance learning algorithm to discover novel preconditions that will further enhance action pruning.

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] 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.

[5] 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.

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

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

[] Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 881–888, New York, NY, USA, 2011. ACM. URL http://www.icml-2011.org/papers/473_icmlpaper.pdf.

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

[9] Matthew Grounds and Daniel Kudenko. 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, 2005.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[] Andrew William Moore and Trinity Hall. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.