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, particularly those of robotics, 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 which are obviously irrelevant to a human solving the same task. In order to leverage knowledge of irrelevant actions in a stochastic state space we formalize the notion of *affordances*: knowledge added to a Markov Decision Process (MDP) which prunes actions in such a way that is neither state space nor reward specific. This action pruning reduces the number of state-action pairs the agent must evaluate in order to behave nearly optimally. Furthermore, we show that an agent can learn affordances through unsupervised experience, and that learned affordances can equal or surpass the performance of those which are provided by experts. We demonstrate our approach in the state-abundant Minecraft domain, showing significant increases in speed E: make sure we actually get an increase in speed (CPU time) after results are gathered and reductions in state-space exploration during planning. Additionally, we demonstrate the immediately practical robotics applications of affordance-aware planning by employing it in a real-world robotic cooking assistant domain.**

I. Introduction

Robots operating in unstructured, stochastic environments face a highly difficult planning problem [? ?]. **E: I don't think it's actually necessary to cite examples of tasks.** Robotics planning problems are classically formalized as a stochastic sequential decision making problem in which 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. **E: Isn't this just an MDP? Shouldn't we mention that here?** However, many robotics problems are of such exceeding complexity that formalizing them as mentioned results in an immense state-action space. This large state-action space, in turn, restricts the classes 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.**E: we need to be careful about state space vs state-action space...**

To address this state-action space explosion, prior work has explored adding knowledge to the planner, such as options [?] and macroactions [? ?]. However, while these methods allow the agent to search more deeply in the state space they add high-level 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 [?]. In deterministic domains, hierarchical task networks (HTNs) add knowledge that greatly increases planning speed [?], but HTNs do not apply to nondeterministic domains**D: Need a citation for this?** and learning this knowledge remains difficult.**D: And this? E: I don't think we need to mention HTNs in the intro since they solve a very different problem.**

To address these issues, we propose augmenting a Markov Decision Process (MDP) with a formalization of *affordances*. In its most abstract form, an affordance simply specifies which actions an agent should consider in different states in order to achieve its goal[?]. By applying formalized affordances to planning, we are able to limit the agent's action set, enabling an agent to focus on aspects of the environment that are most relevant toward solving its current goal and avoid exploration of irrelevant parts of the state-action space, which leads to dramatic speedups in planning. Moreover, our formalization allows generalization across specific tasks, so a single agent can autonomously learn affordances through experience, minimizing the agent's dependence on expert knowledge. Affordances are not specific to a particular reward function **E: aren't they though – like if we suddenly made lava good wouldn't that mess up our agentLookingAtLava affordance? G: Your example would affect the planner (and the affordances would make planning take longer) but they can be learned independent of the reward function.** or state space, and provide the agent with transferable knowledge that is effective in a wide variety of problems.

Our experiments demonstrate that affordances provide dramatic speedups for a variety of planning tasks compared to baselines, may be learned from experience, and can transfer across different tasks. We conduct experiments in the game Minecraft, as well as on a robotic cooking assistant.

II. Affordances

A. *Affordances and OO-MDPs*

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 prefers to maximize 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.

An Object-Oriented Markov Decision Process (OO-MDP) [**?**] can be used to represent an MDP. Unlike the vectorized state of an MDP, 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. The collection of attribute values of a given object is termed that object's state, $o.state$. A vectorized MDP state rcan be equivalently understood as the set of all the object states, $s \in \mathcal{S} = \cup_{i=1}^{o}\{o_i.state\}$, in an OO-MDP.

E: we need to fix this formalism as per what we talked about Dave – i.e. it's an object or a mapping... We define an affordance, Δ, as the mapping $\langle p, g\rangle \longmapsto \mathcal{A}'$, where:

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

The precondition and lifted goal description **E: Can we maybe just call this a goal type?** refer to predicates that are defined in the OO-MDP definition. An affordance is *activated* when its predicate is true in a given state and its lifted goal description g matches the agent's current goal.

Our definition of affordances builds on OO-MDPs. Using OO-MDP predicates for affordance preconditions and goal descriptions allows for state space independence **E: I think a sentence explaining/emphasizing this would be good since it's not entirely obvious at a first pass why OO-MDPs give you state-space independence. E: also here's state space independence but what about reward independence?** Thus, a planner equipped with affordances can be used in any number of different environments. For instance, the affordances defined for navigation problems can be used in any task regardless of the spatial size of the world, number of objects in the world, and specific goal the agent is trying to satisfy.

B. Affordance-Aware Planning

Affordances may be used to restrict the action set of any planner in a particular state to a subset of the action set. Namely, the union of all action sets provided by all active affordances in any particular state, $\mathcal{A}_\Delta$, is a subset of the full action set $\mathcal{A}$.

$$\mathcal{A}_\Delta = \left(\bigcup_{\Delta_{active}} \Delta.getActions(s)\right) \subseteq \mathcal{A} \qquad (1)$$

Our goal for a given state is that the set of affordance actions, $\mathcal{A}_\Delta$, equals the set of optimal actions, $\mathcal{A}^o$. Otherwise stated, we would like the probability that our affordance actions are optimal to be 1:

$$\Pr(\mathcal{A}_\Delta = \mathcal{A}^o \mid s, \Delta_1 \ldots \Delta_K) = 1 \qquad (2)$$

G: The previous sentence seem unnecessary given that we say we want $A_\Delta = A^o$ and have the equation.

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

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

Since our affordance actions are defined as the union of the actions of all active affordances, we can combine equations (1) and (2), where each affordance contributes a set $\mathcal{A}' \subseteq A_\Delta$:

$$\Pr(\mathcal{A}'_1 \cup \ldots \cup \mathcal{A}'_K = \mathcal{A}^o \mid s, \Delta_1 \ldots \Delta_K) \qquad (3)$$

We model the distributions of actions in A' using a Dirichlet-multinomial distribution. When an affordance is active it can simply sample a multinomial from its Dirichelet-multinomial distribution and then return sampled actions from the multinomial. However, it is not obvious exactly how many actions the affordance should sample. To address this ambiguity, the affordance also maintains a Dirichelet distribution over the number of actions it samples from its sampled multinomial. The affordance's action sampling process is specified in 2. Our use of a Dirichlet-multinomial ensures that in the limit, it is possible to apply each action in each state, retaining any optimality guarantees of the planner. Thus, our final expanded probability is:

E: INSERT EQUATION OF D-MULTI PROBABILITY

Provided that the Dirichelet-multinomial and Dirichelet associated with each affordance are properly specified, the probability of sampling the optimal action set, $\mathcal{A}^o$, approaches 1 as the counts of the hyperparameters for the Dirichlet-multinomial and dirichlet distributions increase. We will see exactly how these counts increase in the next section.

Algorithm 2 Δ_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 expert provisions of affordances as specified, any OO-MDP solver can be made *affordance-aware*. Namely, we require that an expert provide a set $\mathcal{P}$ of predicates for the domain of relevance (e.g. Minecraft, Cooking**E: is there a reason Cooking is capitalized or can we smush that?**) and a set $\mathcal{G} \subset \mathcal{P}$ that indicates which predicates may serve as goal conditions. Additonally, they must specify the Dirichlet parameters α and β for each affordance. **E: I feel like we should say what they are – just obfuscating them as "parameters" makes it sound like specifying them would be hard when in reality they're fairly intuitive pseudo-counts**

E: With expert aren't they really specifying pairings of ps and gs not just the entire set of both – there's an implicit pruning by the expert similar to what we do in learning

Note that in the limit**E: limit of what?**, affordances become deterministic. In this way, 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 provided for all experiments. **E: isn't this just hard affordances? shouldn't we mention them by name – also why are we talking about our experiments before our experiments section. I'm a bit confused by the purpose of the above sentence**

C. Learning Affordances

Not only is it arduous to specify α and β but even with expert domain knowledge it is often unclear how to set them. A great deal of the a burden of supplied knowledge could be lessened, then, if these parameters could be inferred without expert intervention. We suggest such a methodology. **E: This is as far as I got on my first pass (I added the above paragraph)**

To learn affordances, we require that a domain expert supply a set of predicates $\mathcal{P}$ and possible goals $\mathcal{G} \subset \mathcal{P}$. Additionally, a domain expert must provide a means of generating candidate state spaces in which each goal $g \in \mathcal{G}$ may be satisfied (i.e. the function $createTestWorld(g)$ at line 5 in Algorithm 3).

The agent forms a set of candidate affordances Δ with every combination of $\langle p, g \rangle$, for $p \in \mathcal{P}$ and $g \in \mathcal{G}$, as seen in line 1-3 of Algorithm 3. To learn the action set for each of these candidate affordances, we developed a 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 example, the agent learns to build towers of blocks in small state spaces that can be solved exactly (i.e. a state space of several thousand states), but generalizes its knowledge to worlds that are too large to solve with exact algorithms (state spaces of tens of thousand to hundreds of thousands of states).

Algorithm 3 $learn(\mathcal{P}, \mathcal{G})$

```
1: for $(p, g) \in \mathcal{P} \times \mathcal{G}$ do
2:     $knowledgeBase.add(\Delta(p, g))$
3: end for
4: for $g \in \mathcal{G}$ do
5:     $w_i = createTestWorld(g)$
6:     $\pi_i = planner.solve(w_i, g)$
7:     $updateParameters(knowledgeBase, \pi_i)$
8: end for
9: $removeLowInfoAffordances(knowledgeBase)$
```

For each optimal policy, we count the number of states in which an action was optimal, when each affordance was activated, as seen in Algorithm 4. α is set to this count. Additionally, we define β as a vector representing counts of integers 1 to $|\mathcal{A}|$. Then, for each optimal policy, we count

Algorithm 4 $updateParameters(knowledgeBase, \pi)$

```
1: for $state \in \pi.reachableStates()$ do
2:     for $\Delta \in knowledgeBase$ do
3:         if $\Delta.p(s) \wedge \Delta.g \models s.g$ then
4:             $\Delta(\pi_i.getOptimalAction(s)).\alpha$++
5:         end if
6:     end for
7: end for
```

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. **D: Need to add beta counts to algorithm 4. (a bit tricky to do concisely so I'm taking a bit of time on it.**

For experiments, we introduce a simplified version of the affordance where the action set $\mathcal{A}$ associated with each affordance is defined as the set of actions whose probability of being optimal was greater than 1% of the probability mass of the sampled multinomial.

III. Experiments

ST: Need to introduce minecraft, robot, and other baselines.

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 complex planning tasks involving assembly, crafting, 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]. 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.

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. Additionally, we ran our full learning process and learned 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 provided the agent with a knowledge base of 17 affordances, which are listed in Figure **??**. Our experiments consisted of a variety of common tasks (state spaces 1-6 in Table **??**) in Minecraft, including constructing bridges over trenches, smelting gold, tunneling through walls, and constructing towers. We tested on worlds

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

of varying size and difficulty to demonstrate the scalability and flexibility of the affordance formalism.

For the learning process, the training data consisted of 150 simple state spaces, each approximately a 100-2000 state 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 twenty consecutive policy rollouts, or the planner failed to converge after 2500 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, movement actions (move, rotate, jump) had a small probability (0.05) of incorrectly applying a different movement action.

We conducted experiments in which we varied the number of training worlds used in the learning process from 0-100. As in Table **??**, we generated 0 to 100 simple state spaces, each a small world (several thousand states) with randomized features that mirrored the agent's actual state space. We then solved the OO-MDP with training data of 0 to 100 simple state spaces to demonstrate the effectiveness of added training data.

Additionally, we compared our approach to Temporally Extended Actions: Macroactions and Options. We compared RTDP 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 randomly generated Minecraft worlds of the same Minecraft tasks as those discussed above (path planning, bridge building, gold smelting, etc.). The option policies and macro actions provided were hand coded by domain experts.

Finally, we deployed an affordance-aware planner onto Baxter for use in an assistive cooking task. **D: Need to fill in more details here**

IV. Results

A. Baxter

Fig. 1. Placeholder for baxter results/image

B. 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 I
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 II
LEARNED AFFORDANCE RESULTS: AVG. CPU TIME PER CONVERGED POLICY (AVERAGE OVER 5 WORLDS PER GOAL TYPE)

C. Minecraft: Learning rate

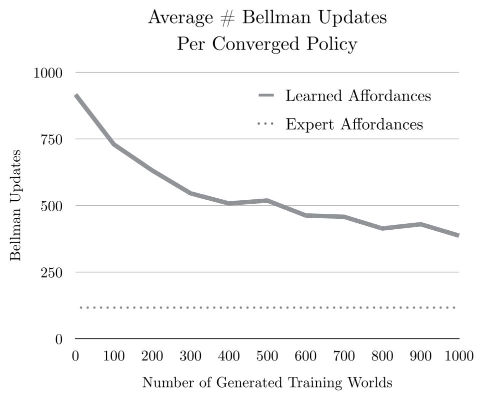

Fig. 2. Placeholder - will recollect this data given recent updates

D. Options

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

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

V. 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* [**?**] and *options* [**?**]. 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 [**? ? ? ? ? ?**].

Given the potential for unhelpful temporally extended actions to negatively impact planning time [**?**], 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. 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 necessitates a plethora of knowledge while affordances solve a planning problem that is stochastic, reward-aware and requires only relatively basic knowledge about the domain. **E: Need citations for HTNs**

C. Action Pruning

Sherstov and Stone [**?**] 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 [**?**] 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 [**? ?**] 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 [**?**].

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 [**?**].

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.

VI. CONCLUSION

D: Conclusion could use some work/rewriting We proposed a novel approach to representing transferable knowledge in terms of *affordances* [**?**] 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. The results support 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 plug into high-level subgoal planning, which will reduce the size of the explored state-action space and improve 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.