Affordance-Aware Planning

Paper-ID [add your ID here]

***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 the planner from considering many actions which would be obviously irrelevant to a human solving the same problem. We introduce a novel, state- and reward-general approach to pruning actions while solving an MDP by encoding knowledge about the domain in terms of *affordances* [?]. This pruning significantly reduces the number of state-action pairs the agent needs to evaluate in order to act optimally. We demonstrate our approach in the Minecraft domain, showing significant increase in speed and reduction in state-space exploration compared to the standard versions of these algorithms. Further, we provide a learning framework based on simulation through scaffolding that enables an agent to learn affordances through experience, removing the dependence on the expert. We provide preliminary results indicating that the learning process effectively produces affordances that help solve an MDP faster.**

I. 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. 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 [**?**].

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 [**?**] and macroactions [**? ?**]. 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. Further, other approaches fall short of learning useful, transferable knowledge, either due to complexity or lack of generalizability (cite? where is this stated? George?).

Instead, we propose a formalization of *affordances* [**?**] that enables an agent to focus on problem-specific aspects of the environment. Our approach avoids exploration of irrelevant parts of the state-action space, which leads to dramatic speedups in planning.

We formalize the notion of an affordance as a piece of planning knowledge provided to an agent operating in a Markov Decision Process (MDP). Affordances are not specific to a particular reward function or state space, and thus, provide the agent with transferable knowledge that is effective in a wide variety of problems. 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 Figure **??**, we demonstrate the effectiveness of affordance-aware subgoal planning on a complicated task in the Minecraft domain.[1] We let other standard planners try to solve this task for several hours, but they all failed to converge on a policy (while affordance-aware subgoal planner found a near-optimal policy in less than 5 minutes).

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. Minecraft's physics and action space is expressive enough to allow very complex worlds to be created by users, such as a functional scientific graphing calculator[2]; simple scenes from a Minecraft world appear in Figure **??**.

Minecraft serves as 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 real world agent. For instance, robotic agents are prone to uncertainty all 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 a pit of lava. We chose to give the Minecraft agent perfect sensor data about the Minecraft world, as that is outside the focus of this work.

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) [**?**]. 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

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

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

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. For example, in `BRIDGEWORLD`, a `nearTrench(STATE)` predicate evaluates to true when the singular instance of class `AGENT` is directly adjacent to an empty location at floor level (i.e. the cell beneath the agent in some direction does not contain a block). In the original OO-MDP work, these predicates were used to model and learn an MDP's transition dynamics. In the next section, we use the predicates to define affordances that enable planning algorithms to prune irrelevant actions.

III. Hard-Affordances

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

> $\mathcal{A}'$ a subset of the action space, $\mathcal{A}$, 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, g, representing a *lifted goal description*.

The precondition and goal description predicates refer to predicates that are defined in the OO-MDP definition. Using OO-MDP predicates for affordance preconditions and goal descriptions allows for state space independent preconditions and goal conditions to be defined and is why the affordances provided to an affordance-aware planner can be used in any number of different tasks. 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.

Given a set of n domain affordances $Z = \{\Delta_1, ..., \Delta_n\}$ and a current agent goal condition defined with an OO-MDP predicate G, the action set that a planning algorithm considers may be pruned on a state by state basis as shown in Algorithm 2.

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:         $\mathcal{A}^* \leftarrow \mathcal{A}^* \cup \Delta.\mathcal{A}'$
5:     **end if**
6: **end for**
7: **return** $\mathcal{A}^*$

Specifically, the algorithm starts 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 ($\Delta.\mathcal{A}'$) are added to the action set $\mathcal{A}^*$ (line 4). Finally, $\mathcal{A}^*$ is returned (line 7).

A. Experiments

We conducted a series of experiments in the Minecraft domain that compared the performance of Real Time Dynamic Programming (RTDP) without affordances, to RTDP when provided with an expert set of affordances. We selected the affordances provided from our background knowledge of the domain. We gave the agent a single knowledge base of 5 types of affordances, which are listed in Figure Our experiments consisted of a variety of common tasks in Minecraft, ranging from basic path planning, to smelting gold, to opening doors and jumping over trenches. We also tested each planner on worlds of varying size and difficulty to demonstrate the scalability and flexibility of the affordance formalism. The evaluation metric for each trial was the number of state backups that were executed by each planning algorithm. RTDP terminated when the maximum change in the value function was less than 0.01 for five 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 -200. The goal was set to be terminal. The discount factor was set to $\lambda = 0.99$. For all experiments, the agent was given stochastic actions. Specifically, actions associated with a direction (e.g. movement, block placement, jumping, etc.), had a small probability (0.3) of moving in another random direction.

TABLE I

EXPERT AFFORDANCE RESULTS

	RTDP	A-RTDP
4BRIDGE	836	**152**
6BRIDGE	4561	**392**
8BRIDGE	18833	**788**
DOORB	12207	**1945**
LAVAB	4425	**993**
TUNNEL	26624	**145**
GOLD	7738	**809**

B. Results

Table I shows the number of bellman updates required when solving the OO-MDP with RTDP (left column) compared to solving the OO-MDP with an Affordance-Aware RTDP (right column). The affordance aware planner significantly outperformed its unaugmented counterpart in all of these experiments. This result demonstrates that affordances prune away many useless action in these block building, block destruction, and gold smelting types of tasks.

IV. LEARNING-AFFORDANCES

We have demonstrated that providing an (OO-)MDP solver with a knowledge base of affordances can lead to dramatic speed ups in planning. However, relying on experts to hand craft affordances is not satisfying. Instead, we would like to have the agent learn these affordances through experience to remove this strict dependence on experts. Here, we propose a methodology for learning affordances directly, with some preliminary results indicating the effectiveness of the system.

A. Learning Process

First, we require a slightly different formalism. If we kept the same formalism as in the expert case, then our learned affordances would often completely get rid of many actions. Since the learned affordances are likely more prone to make mistakes, we can't prune actions in this harsh way, as we will lose optimality guarantees of the OO-MDP solver.

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: $\quad \Delta_i.\mathcal{A}' \leftarrow \lambda$
5: **end for**
6: **return** $\Delta_i.\mathcal{A}'$

Instead, for a given state, we solve for the probability of getting a particular action set $\mathcal{A}^*$, and sample from this distribution.

$$\Pr(\mathcal{A}^*|s, \Delta_0 \ldots \Delta_N) \tag{1}$$

We know each affordance contributes a set $\mathcal{A}'$ in each state:

$$\Pr(\mathcal{A}'_0 \cup \mathcal{A}'_N|s, \Delta_0 \ldots \Delta_N) \tag{2}$$

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

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

For each affordance, to get an action set $\mathcal{A}'$, we form a Dirichlet Multinomial over actions (λ), and a Dirichlet over the size (N) of each action set:

$$\Pr(\lambda \mid \alpha) = DirMult(\alpha) \tag{4}$$

$$\Pr(N \mid \beta) = Dir(\beta) \tag{5}$$

Then, for each affordance we sample from our distribution over N to get a candidate action set size, n, and then take n samples from our distribution over λ to get a candidate action set $\mathcal{A}'$.

$$\Pr(\mathcal{A}_i \mid s, \Delta_i) = \Pr(\mathcal{A}'_i \mid N, \lambda) = \Pr(\lambda \mid \alpha)\Pr(N \mid \beta) \tag{6}$$

B. Computing α and β

We require that an expert provide a set $\mathcal{P}$ of propositional functions for the domain of relevance (i.e. Minecraft). Additionally, they must specify a set $\mathcal{G} \subset \mathcal{P}$, that indicates which PropositionalFunctions may serve as goals. We form a set of candidate affordances Δ with every combination of $\langle p, g \rangle$, for $p \in \mathcal{P}$ and $g \in \mathcal{G}$.

Then, we randomly generate a large number of small state spaces (typically on the order of several thousand), annotated with their lifted goal description $g \in \mathcal{G}$. We solve the OO-MDP in each state space and get an optimal policy π_j. For each optimal policy, we count the number of policies that used each action when each affordance was activated[3]. These counts represent α. Then, we count the number of unique actions used by each policy, representing β.

C. Experiments

We tested our learning procedure on several simple worlds of varying size. We compared the performance of RTDP solving the OO-MDP in each of these worlds with (1) No affordances, (2) Expert provided affordances, and (3) Learned affordances.

D. Results

Table II 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 still a substantial gap between the learned affordance performance and that of the expert affordances. This indicates that there is likely a lot to improve in the learning process.

[3] An affordance is 'activated' when its predicate is true and the lifted goal description g matches the agent's current goal

TABLE II
LEARNED AFFORDANCE RESULTS

	No Affordances	Learned	Expert
`Tiny World`	879	576	94
`Small World`	1460	1159	321
`Medium World`	3993	2412	693
`Large World`	8344	5100	1458

V. CONCLUSION

VI. RSS CITATIONS

Please make sure to include `natbib.sty` and to use the `plainnat.bst` bibliography style. `natbib` provides additional citation commands, most usefully `\citet`. For example, rather than the awkward construction

```
\cite{kalman1960new} demonstrated...
```

rendered as "[**?**] demonstrated...," or the inconvenient

```
Kalman \cite{kalman1960new}
demonstrated...
```

rendered as "Kalman [**?**] demonstrated...", one can write

```
\citet{kalman1960new} demonstrated...
```

which renders as "**?**] demonstrated..." and is both easy to write and much easier to read.

A. RSS Hyperlinks

This year, we would like to use the ability of PDF viewers to interpret hyperlinks, specifically to allow each reference in the bibliography to be a link to an online version of the reference. As an example, if you were to cite "Passive Dynamic Walking" [**?**], the entry in the bibtex would read:

```
@article{McGeer01041990,
  author = {McGeer, Tad},
  title = {\href{http://ijr.sagepub.com/content/9/2/62.abstract}{Passive Dynamic Walking}},
  volume = {9},
  number = {2},
  pages = {62-82},
  year = {1990},
  doi = {10.1177/027836499000900206},
  URL = {http://ijr.sagepub.com/content/9/2/62.abstract},
  eprint = {http://ijr.sagepub.com/content/9/2/62.full.pdf+html},
  journal = {The International Journal of Robotics Research}
}
```

and the entry in the compiled PDF would look like:

[1] Tad McGeer. Passive Dynamic Walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

where the title of the article is a link that takes you to the article on IJRR's website.

Linking cited articles will not always be possible, especially for older articles. There are also often several versions of papers online: authors are free to decide what to use as the link destination yet we strongly encourage to link to archival or publisher sites (such as IEEE Xplore or Sage Journals). We encourage all authors to use this feature to the extent possible.

ACKNOWLEDGMENTS

REFERENCES