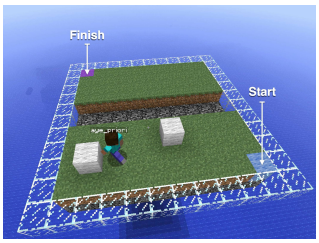
Affordance-Aware Planning

Abstract

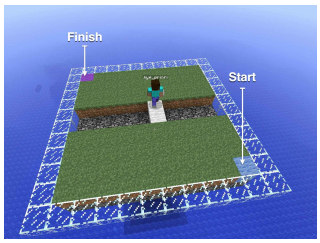
Planning algorithms for non-deterministic domains are often intractable in large state spaces due to the well-known "curse of dimensionality." Existing approaches to address this problem fail to prevent the system from considering many actions which would be obviously irrelevant to a human solving the same problem. We introduce a novel, state- and reward- general approach to limiting the branching factor in large domains by encoding knowledge about the domain in terms of *affordances* (Gibson, 1977). Our affordance formalism can be coupled with a variety of planning frameworks to create "affordance-aware planning," allowing an agent to efficiently prune its action space based on domain knowledge and its current subgoal. 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.

1 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 (Grounds and Kudenko, 2005).



(a) Planning with VI.



(b) Planning with affordances.

Figure 1: Scenes from a Minecraft agent planning using Value Iteration (VI) compared to affordance-aware VI in a bridge building task. We were forced to cut off VI after several hours due to the large nature of the Minecraft state space, while the Affordance-Aware VI converged to a policy in under a minute.

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 et al., 1999) and macroactions (Botea et al., 2005; Newton et al., 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.

Instead, we propose a formalism that enables an agent to focus on problem-specific aspects of the environment, guiding the search toward the most relevant and useful parts of the state-action space. This approach reduces the size of the explored state action space,

leading to dramatic speedups in planning. Our approach is a formalization of *affordances*, introduced by Gibson (1977) as "what [the environment] offers [an] animal, what [the environment] provides or furnishes, either for good or ill."

We formalize the notion of an affordance as a piece of planning knowledge provided to an agent operating in a Markov Decision Process (MDP). We explain how affordances can be leveraged by a variety of planning algorithms to prune the action set the agent uses dynamically based on the agent's current goal. We call any planning algorithm that uses affordances to prune the action set an *affordance-aware* planning algorithm. 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 3, we demonstrate the effectiveness of affordance-aware subgoal planning on a complicated task in the Minecraft domain - a video is provided of the agent achieving this task [1].**ST: Can we move figure 3 earlier to this reference?** 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).

2 MINECRAFT DOMAIN

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 1.

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

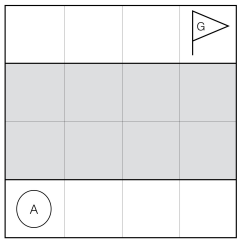


Figure 2: `BRIDGEWORLD`

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 non-noisy sensory data about the Minecraft world.

As a running example, we will consider the problem of an agent attempting to cross a trench in a $4 \times 4 \times 2$ Minecraft world; a schematic appears in Figure 2.**ST: Can we put a minecraft picture of the same scene next to it?** The floor (at $z = 1$)[3] is composed of 8 solid blocks, with horizontal empty trenches at $y = 2$ and $y = 3$. The agent is at the starting location $(1, 1, 2)$ and needs to reach the goal at $(4, 4, 2)$

To solve the problem, the agent must place a block in the trench to form a bridge, then cross the bridge to reach the goal. This task is challenging for planning algorithms to solve because the reachable state space in Minecraft is so large. For example, the number of locations an agent can place and destroy blocks alone can result in a combinatorial explosion of the state space. An affordance-aware planner, however, (when equipped with the proper affordances) will only consider placing or destroying a block when that action moves the agent closer to its goal. Thus, when the agent is in a state in which block placement is not considered useful, the agent will not have access to the block placement action (and will not be able to explore the states that result in applying the block placement action in its current state). As a result, the agent dramatically prunes its effective action space.

3 BACKGROUND

The term "affordance" was introduced by Gibson (1977). At a high-level, an affordance may be thought of as the action-possibilities that an environment presents to an agent. More specifically, Gibson proposed that an affordance be thought of as "what [the environment] offers [an] animal, what [the environment] provides or furnishes, either for good or ill" (Gibson, 1977). He added that an affordance may

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

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

[3]The z-axis is the height of the Minecraft world. Similarly, the x-axis is its width and the y-axis is its length.

not be thought of as a physical property of the environment itself, as an affordance must be defined with respect to the capabilities and features of a specific animal in addition to the environment. There have been many attempts at properly grounding affordances in a variety of academic disciplines (Koppula and Saxena, 2013a; Hartson, 2003; Koppula and Saxena, 2013b; Gorniak and Roy, 2006; Kaschak and Glenberg, 2000). Our aim in this paper is to provide a simple, yet general definition of an affordance in terms of knowledge added to an MDP which enables dramatic speedups in planning times, depending on the agent's goal. Specifically, we define an affordance in terms of the agent's current goal; depending on the needs of the agent, it may only consider certain affordances. For instance, a rock may serve as a paper-weight to an agent seeking to weigh down a stack of papers. If the agent's goal changes and is instead looking for a means of propping a door open, then the agent may instead consider that the rock will serve as an adequate doorstop.

3.1 OO-MDP

We define affordances in terms of propositional functions on states. Our definition builds on the Object-Oriented Markov Decision Process (OO-MDP) (Diuk et al., 2008). 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 $Dom(c.a)$. 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. Formally, if there exists a bijection between two sets of objects that maps each object in one set to an object in the other set with the same object state, then the two sets of objects define the same MDP 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(AGENT)` 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.

4 AFFORDANCES

We define an affordance Δ, as a tuple, $\langle p, g \rangle \longmapsto \alpha$, where:

> α 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 1.

Algorithm 1 pruneActions(*state*, Z, G)
Complexity: $\mathcal{O}(|Z|)$

```
1: A' ← {}
2: for Δ ∈ Z do
3:     if Δ.p(state) and Δ.g = G then
4:         A' ← A' ∪ Δ.α
5:     end if
6: end for
7: return A'
```

Specifically, the algorithm starts by initializing an empty set of actions 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 ($\Delta.\alpha$) are added to the action set A' (line 4). Finally, A' is returned (line 7).

For an example of the action pruning performed, consider the Minecraft world `BRIDGEWORLD` with a task goal of navigating to a location (defined by the *reachGoal* predicate) and a set of three affordances with the same goal description:

$$\Delta_1 = \langle nearTrench, reachGoal \rangle \longmapsto \{place\}$$
$$\Delta_2 = \langle onPlane, reachGoal \rangle \longmapsto \{move\}$$
$$\Delta_3 = \langle nearWall, reachGoal \rangle \longmapsto \{destroy\}$$

The first affordance allows block placement when the agent is adjacent to a trench, since placing blocks in trenches may be necessary if the agent must cross the trench. The second affordance allows movement when the agent is on a flat surface. The third affordance allows block destruction when the agent is at a wall. One important consequence of these affordances is that the agent will not considering placing blocks in locations that cannot enable the agent to reach its goal location; as a result, the large number of irrelevant states that are otherwise reachable by placing different blocks in various locations are removed from the state space explored by the planner.

When a planning algorithm prunes its action set using affordances as described, we call it an *affordance-aware* planner. Pruning the action set affects different planning algorithms in different ways. In particular, we focus on how action pruning benefits *dynamic programming*, *policy rollout*, and *subgoal* planning paradigms in the following sections.

4.1 DYNAMIC PROGRAMMING

In dynamic programming paradigms, the planning algorithm estimates the optimal *value function* for each state. Formally, the optimal value function (V^*) defines the expected discounted return from following the optimal policy in each state:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} \Pr(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma V^*(s') \right]; \tag{1}$$

this equation is known as the Bellman equation (Bellman, 1957). Given the optimal value function, the optimal policy is derived by taking the action that maximizes the values of each state; that is, by taking the action with the highest optimal state-action value:

$$Q^*(s, a) = \sum_{s'} \Pr(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma V^*(s') \right]. \tag{2}$$

Dynamic programming planning algorithms (such as Value Iteration (Bellman, 1957)) estimate the optimal value function by initializing the value of each state arbitrarily and iteratively updating the value of each state by setting its value to the result of the right-hand-side of the Bellman equation using its current estimate of V instead of V^*. Iteratively updating the value function estimate in this way is guaranteed to converge to the optimal value function.

Using a pruned action set in dynamic programming can accelerate its computation in two ways: (1) by reducing the number of actions over which the max operator in the Bellman equation must iterate and (2) by restricting the state space for which the value function is estimated to the states that are reachable with the pruned action set from the initial state. Note that neither of these computational gains come at the cost of solution optimality as long as the pruned action set contains the actions necessary for an optimal policy from the initial state. In the case of the Bellman equation, the max operator makes the value function indifferent to the effects of actions that are not part of the optimal policy; therefore, the action set can be reduced entirely to the actions in the optimal policy without sacrificing optimality. Similarly, since we are only concerned with finding a good policy to dictate behavior from some initial state, the state space for which the value function is computed can be reduced to that which is reachable using only the optimal actions without sacrificing optimality.

4.2 POLICY ROLLOUT

In policy rollout planning paradigms, the agent starts with some initial policy and follows it (or rolls out the policy) from an initial/current state to either some maximum time horizon or until a terminal state is reached. Often, these approaches use samples from the policy rollout to improve estimates of the value function and indirectly improve the rollout policy. Examples of planning algorithms in this paradigm include Monte Carlo methods (Browne et al., 2012; Silver and Veness, 2010) and temporal difference methods (Sutton et al., 1999; Sutton, 1988; Rummery and Niranjan, 1994; Barto et al., 1983; Lagoudakis and Parr, 2003; Peters and Schaal, 2008). By using a pruned action set, the policy space, and resulting state space explored from the searched policies, is reduced, thereby reducing the number of rollouts necessary to find a good policy. Similar to dynamic programming paradigms, as long as the pruned action set contains actions necessary for the optimal policy, solution optimality will not be sacrificed.

In this work, we will explore how real time dynamic programming (RTDP) (Barto et al., 1995) benefits from affordances. RTDP is both a dynamic programming algorithm and a policy rollout algorithm. RTDP starts by initializing the value function optimistically. It then follows a greedy rollout policy with respect to its currently estimated value function. After each action selection in the policy rollout, RTDP updates its estimate of the value function for the last state using the Bellman equation. RTDP is guaranteed to converge to the optimal policy from some initial state and has the advantage that it iteratively refocuses its attention to states that are likely to be on the path of the optimal policy. We chose to use RTDP as a baseline for three reasons: (1) it demonstrates affordance-aware planning for both a dynamic programming paradigm and a policy rollout paradigm; (2) because one of the advantages of RTDP is that it is focused on finding solutions from an initial state rather than the whole state space, which is the type of planning problem our work addresses; and (3) it is easy to check for planning convergence in stochastic domains with RTDP.

In affordance-aware RTDP, the action selection of the rollout policy is restricted to the affordance-pruned action set and the Bellman equation is similarly restricted to operating on the affordance-pruned action set.

4.3 SUBGOAL PLANNING

Subgoal planning is based on the intuition that certain goals in planning domains may only be brought about if certain preconditions are first satisfied. For instance, in the bridge problem, the agent must first place a block in the trench to create a bridge before crossing the trench. Branavan et al. (2012) explore learning subgoals from the Minecraft wiki and applying them in order to plan through a variety of problems in Minecraft.

Formally, in subgoal planning, the agent is given a set of subgoals, where each subgoal is a pair of predicates:

$$SG = \langle x_k, x_l \rangle \tag{3}$$

ST: Isn't subgoal planning a directed graph? Do the $\langle x_k, x_l \rangle$ >'s form a DAG and we search in the dag? In that case I think this defintiion is right but missing some of the intuition of subgoals, and a few setences about the graph nature would help clarify that.

where x_l is the effect of some action sequence performed on a state in which x_k is true. Thus, subgoal planning requires that we perform high-level planning in subgoal space, and low-level planning to get from subgoal to subgoal. The low-level planner may vary, though Metric-FF and A* are popular choices (depending on domain constraints), as is Value Iteration.

In the case of `BRIDGEWORLD`, the agent might consider placing a block somewhere along the trench to be a subgoal. Then, it runs a low-level planner to get from its starting location to the subgoal. Next, it runs the same low-level planner from the first subgoal to the finish.

Using affordances with subgoal planning is particularly appealing because every time the subgoal switches, a different set of affordances become active and restrict the state space to only that which is more immediately relevant. Moreover, since one possible disadvantage of subgoal planning is that much of the same state space may be re-explored every time a the subgoal switches, the affordances help minimize this effect. Because subgoal planning operates on a subgoals defined with predicates, its subgoal definition is entirely compatible with the affordance definition and does not require any additional translation.

5 EXPERIMENTS

We conducted a series of experiments in the Minecraft domain that tested standard planners from each planning paradigm: Value Iteration, RTDP, and Subgoal planning (with RTDP as the low-level planner). These planners were compared with *affordance-aware* versions of each algorithm tasked with the same set of problems. We selected the affordances provided from our background knowledge of the domain. We gave

Table 1: Mutable Results

	VI	A-VI	RTDP	A-RTDP	SG	A-SG
4BRIDGE	71604	100	836	152	1373	141
6BRIDGE	413559	**366**	4561	392	28185	547
8BRIDGE	1439883	904	18833	**788**	15583	1001
DOORB	861084	4368	12207	1945	6368	**1381**
LAVAB	413559	366	4425	993	25792	**597**
TUNNEL	203796	**105**	26624	145	5404	182
BREAD	16406	962	7738	809	7412	**578**

Table 2: Static Results

	VI	A-VI	RTDP	A-RTDP	SG	A-SG
10WORLD	**800**	**800**	1205	985	1263	960
15WORLD	3150	3150	3939	3089	3328	**2331**
20WORLD	7200	7200	10719	8004	8738	**6099**
DOOR	6315	6315	5646	4059	4104	**2886**
JUMP	2940	2940	4313	3548	4262	**2922**
MAZE	4266	4266	5648	2864	**1949**	3073
LAVA	800	800	1328	861	1003	**772**

the agent a single knowledge base of 15 affordances for all of the tasks. Our experiments consisted of a variety of tasks, ranging from basic path planning, to baking bread, 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. 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 five consecutive policy rollouts. In subgoal planning, 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.

The reward function is -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 (0.1) of being applied in the reverse direction.

5.1 RESULTS

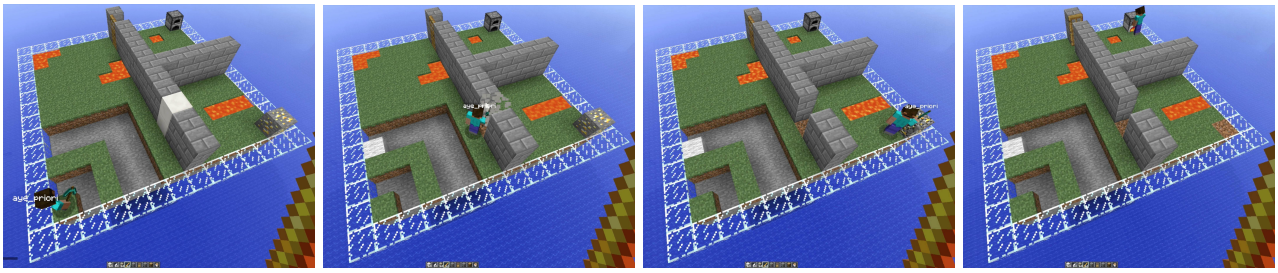
Table 1 shows the results of running the standard planners and their affordance aware counterparts on a set of tasks that require the use of block placement and/or destruction. The affordance aware planners significantly outperformed their unaugmented counterparts in all of these experiments. They proved especially effective when paired with subgoals, as can be seen by the results of the affordance aware subgoal planner, demonstrating that affordances are particularly useful if subgoal knowledge is known. Subgoals combined with affordances allow different types of pruning to occur depending on what the agent is trying to accomplish at each stage. In other words, when the agent is trying to find grain, it will focus on grain finding actions; when it is trying to bake bread, it focuses on bread baking actions. Additionally, affordance aware VI and affordance aware RTDP outperform their unaugmented versions. This result demonstrates that affordances prune away many useless action in these block building, block destruction, and bread baking types of tasks.

One interesting result is that affordance-aware VI sometimes did the best, even better than affordance-aware RTDP. One possible reason for this result is that the affordances sufficiently focused the state space such that there was not much gain in RTDP iteratively refocusing where its updates were applied. Furthermore, since VI would back up values from goal states in the first iteration, it may have allowed the goal information to propagate back to the initial state faster than it would take RTDP to perform enough rollouts to find it.

Table 2 shows the results of running the standard planners and their affordance-aware counterparts on tasks that do *not* require the use of block placement and destruction. In these cases, the affordance awareness did not consistently improve any of the planners from their baseline versions. Affordances are mostly beneficial when they allow an agent to prune actions which could combinatorially alter the state space. More generally, if the action set is relatively small and the actions do not significantly impact the shape of the state space with each application, then affordances will not increase speed of planning. We also collected data on the amount of accumulated reward that resulted from following each algorithm's policy, but found that the difference in accumulated reward across planners was negligible.

One of the most compelling results is the scope of task that affordance-aware planners are capable of solving. With an affordance-aware Subgoal planner (i.e. using an affordance aware RTDP as the low level planner), a Minecraft agent was able to traverse a complicated obstacle course[4], as seen in Figure 3. In this task, the agent had to smelt gold, but in order to do so, it had to jump over a trench, build a bridge over another trench, chop down a wall, mine a block of gold, open a door, avoid lava, and finally smelt the gold ore in a furnace.

[4] A full video of the agent solving this task may be found at: https://vimeo.com/88689171



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

Figure 3: The affordance-aware subgoal planner solving an ore-smelting task with a variety of obstacles. This task was only achievable by an affordance-aware planner.

We have included a video of this task being executed as part of our supplementary materials. Here, the agent was capable of completing several different types of tasks (subgoals) with a single action space and affordance knowledge base. Using affordance-aware subgoal planning, this problem was solved in under five minutes, whereas other non-affordance-aware planning algorithms did not complete even after several hours on the same computer.

6 Related Work

In this section, we discuss the differences between affordance-aware planning and other forms of background knowledge that have been used to accelerate planning. Specifically, we discuss heuristics, temporally extended actions, and related action pruning work. Additionally, we elaborate on other recent attempts of employing Gibson's notion of an affordance to the problems of computer science and robotics.

6.1 HEURISTICS

Heuristics in MDPs are used to convey information about the value of a given state or state-action pair with respect to the task being solved and typically take the form of either *value function initialization*, or *reward shaping*. For planning algorithms that estimate state-value functions, heuristics are often provided by initializing the value function to values that are good approximations of the true value function. For example, initializing the value function to an admissible close approximation of the optimal value function has been shown to be effective for LAO* and RTDP, because it more greatly biases the states explored by the rollout policy to those important to the optimal policy Hansen and Zilberstein (1999). Planning algorithms that estimate Q-values instead of the state value function may similarly initialize the Q-values to an approximation of the optimal Q-values. For instance, PROST Keller and Eyerich (2012) creates a *determinized* version of a stochastic domain (that is, treating each action as if its most likely outcome always occurred), plans a solution in the determinized domain, and then initializes Q-values to the value of each action in the determinized domain.

Reward shaping is an alternative approach to providing heuristics in which 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. Reward shaping differs from value function initialization in that it is not guaranteed to preserve convergence to an optimal policy unless certain properties of the shaped reward are satisfied Ng et al. (1999) that also have the effect of making reward shaping equivalent to value function initialization for a large class of planning/learning algorithms Wiewiora (2003).

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; therefore, different tasks require different heuristics to be provided, whereas affordances are state 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.

6.2 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* (Sutton et al., 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. An option o is defined as follows:

$o = \langle \pi_0, I_0, \beta_0 \rangle$, where:

$$\pi_0 : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$$
$$I_0 : \mathcal{S} \rightarrow \{0, 1\}$$
$$\beta_0 : \mathcal{S} \rightarrow [0, 1]$$

Here, π_0 represents the *option policy*, I_0 represents a precondition, under which the option policy may initiate, and β_0 represent the post condition, which determines which states terminate the execution of the option policy.

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 et al., 2008; Andre and Russell, 2002; Konidaris et al., 2012).

Although temporally extended actions are typically used because they represent action sequences (or sub policies) that are often useful to solving the current task, they can sometimes have the paradoxical effect of increasing the planning time because they increase the number of actions that must be explored. For example, deterministic planning algorithms that successfully make use of macro-actions often avoid the potential increase in planning time by developing algorithms that restrict the set of macro-actions to a small set that is expected to improve planning time for the problem (Botea et al., 2005; Newton et al., 2005) or by limiting the use of macro-actions to certain conditions in the planning algorithms like when the planner reaches heuristic plateaus (areas of the state space in which all successor states have the same heuristic value) Coles and Smith (2007). Similarly, it has been shown that the inclusion of even a small subset of unhelpful options can negatively impact planning/learning time Jong (2008).

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 which may actually be useful to a task. In the future, we plan to more directly explore the benefit from combining these approaches.

6.3 ACTION PRUNING

Work that prunes the action space is the most similar to our affordance-aware planning. 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. Since the actions of the optimal policy of a source task may not include all the actions of the optimal policy in the target task, source task action bias was reduced by randomly perturbing the value function of the source task to produce new synthetic tasks. The action set transferred to the target task was then taken as the union of the actions in the optimal policies for the source task and all the synthetic tasks generated from it.

A critical 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 (or may be inferred).

Rosman and Ramamoorthy Rosman and Ramamoorthy (2012) provide a method for learning action priors over a set of related tasks. Specifically, a Dirichlet distribution over actions was computed by extracting the frequency that each action was optimal in each state for each previously solved task. On a novel task learned with Q-learning, a variant of an ϵ-greedy policy was followed in which the agent selected a random action according to the Dirichlet distribution an ϵ fraction of the time, and the action with the max Q-value the rest of the time. To avoid dependence on a specific state space, the a Dirichlet distribution was created for each observation-action pair (where the observations were task independent) instead of each state-action pair.

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. However, in the future, the action set each affordance defines could be learned using a similar approach.

6.4 AFFORDANCES

Steedman (Steedman, 2002) performed work in formalizing affordances through the Linear Dynamic Event Calculus (LDEC). This work was aimed at linking events and objects in the context of language and prelinguistic cognitive apparatuses, and is partially related to the Frame Problem and symbolic planning. Koppula and Saxena (Koppula and Saxena, 2013a) developed an inference algorithm that enables a robotic agent to anticipate a human partner's actions based on the robots perceived affordances. Additionally, Koppula, and Gupta (Koppula et al., 2013) performed work in predicting object affordances by treating humans as a latent variable in a given scene. Gorniak (Gorniak, 2005) proposed the Affordance-Based-Concept, which targeted using the perception of affordances as a means of determining the set of possible interactions an agent may engage with in a given context. This was then applied toward situated-language understanding and language-generating agents. Lastly, many have used affordances as a paradigm for solving table top grasping problems (ten Pas and Platt, 2013; Sweeney and Grupen, 2007; Detry et al., 2009; Montesano and Lopes, 2009).

7 CONCLUSION

We proposed a novel approach to representing knowledge in terms of *affordances* (Gibson, 1977) that allows an agent to efficiently prune its action space based on domain knowledge. This led to the proposal of affordance-aware planners, which improve on classic planners by providing a significant reduction in the number of state/action pairs the agent needs to evaluate in order to act optimally. We demonstrated the efficacy as well as the portability 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.

In the future, we hope to introduce a more robust inference procedure for pruning actions such that the agent not only prunes away *useless* actions, but also prioritizes between *great* actions, and *mediocre* ones. Additionally, each affordance knowledge base must be designed by hand for planning agents - our immediate next step will be to apply learning techniques to learn affordances directly. Further, we foresee extensions in natural language processing and information extraction, in which affordances may be inferred via text or from dialogue with a human partner. This promises extensions in which a robotic agent receives aid from a human partner through natural language dialogue; the agent may ask for help when it is stuck and receive affordance or subgoal *hints* from a human companion. Lastly, we consider applications to a variety of other planning strategies, including A* for use in a robotic-cooking companion, as well as enhancing POMDPs with affordances, directed at applications to robotic care-giver companions.

References

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.

A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-13(5):834 –846, sept.-oct. 1983.

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

Richard Bellman. Dynamic programming, 1957.

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.

S.R.K. Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning high-level planning from text. In *Proceedings of the Conference of the Association for Computational Linguistics*, ACL '12, 2012.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

Andrew Coles and Amanda Smith. Marvin: A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28:119–156, 2007.

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

R. Detry, M. Popovi, Y. Touati, N. Krger, O. Kroemer, J. Peters, and J. Piater. Learning objectspecific grasp affordance densities. In *In International Conference on Development and Learning*, 2009.

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.

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

Peter Gorniak. *The Affordance-Based Concept*. PhD thesis, Massachusetts Institute of Technology, 9 2005.

Peter Gorniak and Deb Roy. Situated language understanding as filtering perceived affordances. *Cognitive Science*, 31, 2006.

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.

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.

Rex Hartson. Cognitive, physical, sensory, and functional affordances in interaction design. *Behaviour & Information Technology*, 22(5):315–338, 2003.

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.

Michael P Kaschak and Arthur M Glenberg. Constructing meaning: The role of affordances and grammatical constructions in sentence comprehension. *Journal of Memory and Language*, 43:508–529, 2000.

T. Keller and P. Eyerich. Prost: Probabilistic planning based on uct. In *International Conference on Automated Planning and Scheduling*, 2012.

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.

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

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.

Hema S. Koppula and Ashutosh Saxena. Anticipating human activities using object affordances for reactive robotic response. In *Robotics: Science and Systems (RSS)*, 2013a.

Hema S. Koppula and Ashutosh Saxena. Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation. In *International Conference on Machine Learning (ICML)*, 2013b.

Hema S. Koppula, Rudhir Gupta, and Ashutosh Saxena. Learning human activities and object affordances from rgb-d videos. *International Journal of Robotics Research*, 2013.

M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4: 1107–1149, 2003.

Luis Montesano and Manuel Lopes. Learning grasping affordances from local visual descriptors, 2009.

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.

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.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

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.

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.

G.A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report 166, University of Cambridge, Department of Engineering, 1994.

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.

David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *NIPS*, volume 23, pages 2164–2172, 2010.

Mark Steedman. Formalizing affordance. In *In Proceedings of the 24th Annual Meeting of the Cognitive Science Society*, pages 834–839, 2002.

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.

R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

John D. Sweeney and Rod Grupen. A model of shared grasp affordances from demonstration. In *in: Proceedings of the IEEE-RAS International Conference on Humanoids Robots, Humanoids07*, 2007.

Andreas ten Pas and Robert Platt. Localizing grasp affordances in 3-d points clouds using taubin quadric fitting. *CoRR*, abs/1311.3192, 2013.

Eric Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.