Planning with Affordances

Abstract

Current methods for exactly solving decision-making under uncertainty require exhaustive enumeration of all possible states and actions, leading to exponential run times, leading to the well-known "curse of dimensionality." Approaches to address this problem by providing the system with formally encoded knowledge such as options or macro-actions still fail to prevent the system from considering many actions which seem obviously irrelevant for a human partner. To address this issue, we introduce a novel approach to representing knowledge about how to plan in terms of *affordances* [1]. Our affordance formalism and associated planning framework allows an agent to efficiently prune its action space based on domain knowledge. 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 on several planning and building tasks, showing a significant increase in speed and reduction in state-space exploration compared to subgoal planning, options, and macro-actions.

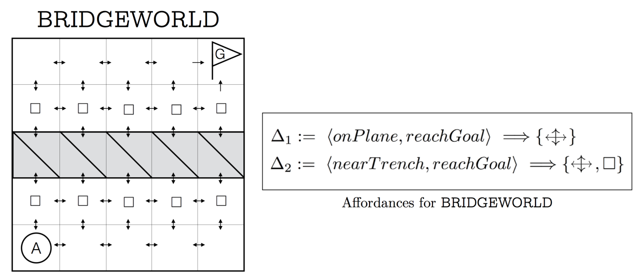


Figure 1: In the above Minecraft planning problem `BRIDGEWORLD`, the agent must place a block in the trench in order to reach the goal. This problem was only solvable by the Affordance planner.

1 INTRODUCTION

As robots move out of the lab and into the real world, planning algorithms need to be able to scale to domains of increased noise, size, and complexity. A classic formalization of this issue is the sequential decision making problem, where increases in problem size and complexity directly correspond to an explosion in the state-action space. Current approaches to solving sequential decision making problems cannot tackle these problems as the state-action space becomes large [**?**].

There is a strong need for a generalizable form of knowledge that, when coupled with a planner, is capable of solving problems in these massive domains. Humans provide an excellent existence proof for such planning, as we are capable of searching over an immense number of possible actions when presented with a goal. One approach to explaining how humans solve this planning problem is by focusing on problem-specific aspects of the environment which focus the search toward the most relevant and useful parts of the state-action space. Formally, Gibson [1] proposed to define this intuition as an *affordance*, "what [the environment] offers [an] animal, what [the environment] provides or furnishes, either for good or ill." Additionally, roboticists have recently become interested in leveraging affordances for perception and prediction of humans [3, 4].

In this paper we will formalize the notion of an affordance as a piece of planning knowledge provided to an agent operating in a Markov Decision Process (MDP) [2]. We demonstrate that, like an option or macro-action, an affordance provides additional information to the agent, enabling more efficient planning. However, unlike previous approaches, an affordance enables more significant speedups by reducing the size and branching-factor of the search space, enabling an

$$\begin{aligned}
\updownarrow\!\!\!\!\leftrightarrow &= \{\uparrow, \leftarrow, \downarrow, \rightarrow\} \\
\square &= \{\uparrow\square, \leftarrow\square, \downarrow\square, \rightarrow\square\} \\
\boxtimes &= \{\uparrow\boxtimes, \leftarrow\boxtimes, \downarrow\boxtimes, \rightarrow\boxtimes\} \\
\\
\mathcal{A} &= \{\updownarrow\!\!\!\!\leftrightarrow, \square, \boxtimes\}
\end{aligned}$$

Figure 2: The agent's set of actions in the Minecraft domain ($\square$ is the set of placement actions, $\boxtimes$ is the set of destruction actions)

agent to focus its search on the most relevant part of the problem at hand. This approach means that a *single* set of affordances provides general domain knowledge, becoming relevant just when the agent reasons that it needs to pursue a particular subgoal. Furthermore, Affordances are not specific to a particular state-space nor problem-type, and thus, provide the agent with transferrable knowledge that is effective in a wide variety of domains and problems, unlike other approaches.

2 BACKGROUND

2.1 OO-MDP

The Object Oriented Markov Decision Process (OO-MDP) [**?**] is an extension of the classic Markov Decision Process (MDP), a fundamental building block of Reinforcement Learning (RL). RL is an algorithmic approach to sequential decision making problems, or more simply: planning.

A finite 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 $s' \in \mathcal{S}$, and $\mathcal{R}(s, a)$ denotes the reward at s when action a is applied, and γ is a discount factor. The OO-MDP changes the underlying representation of the state space S so that worlds consist of instances of objects that are grouped into classes, where each object obtains of a set of attributes. Thus, a state in the OO-MDP is just the union of the attribute values of all objects in the world. As with a classical finite MDP, planning with an OO-MDP involves running Value Iteration to determine a policy. Recall that value iteration involves propagating reward throughout the state space until propagation has converged, at which point, a policy may be deduced. Reward propagation occurs as in a Bellman update:

$$U_{i+1}(s) \leftarrow \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s'} \Pr(s' \mid s, a) U_i(s') \quad (1)$$

Where $U_i(s)$ is the *utility* of state s at iteration i, representing the expected reward of being in that state.

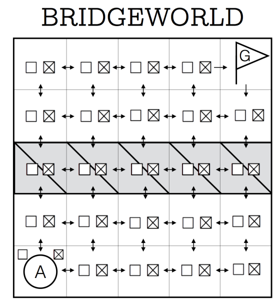


Figure 3: Reachability while running VI

In the Russell and Norvig AI text, they provide

In practice, basic Value Iteration scales very poorly, either as the state space grows, or the action set grows. This is because the state-action space, depending on the domain, grows exponentially w.r.t the number of ways that the agent can change the environment. This is ameliorated slightly by introducing the OO-MDP, but it still fails in just about all of the planning scenarios we introduce here. On `BRIDGEWORLD`, the planner was incapable of enumerating the state space prior to filling up a 2Gb Java heapspace, presented with a single block and minimal action set (it could only place blocks in front of it, and could not destroy blocks) and thus, failed outright.

The reason for this failure is that in classical Value Iteration, the agent tries to explore all states that result from applying every action in every state - this is downright silly, as in `BRIDGEWORLD`, the agent will inevitably end up in the corner of the room placing and destroying blocks. This is an especially bad tactic in the Minecraft domain, as block placement results in a combinatoric explosion of the state space. Thus, in these scenarios, our Affordance planner has a substantial advantage on classic Value Iteration. These weaknesses are well known [**?**], and have resulted in attempts to make planning more practical in domains of a larger scale.

2.2 SUBGOALS

Subgoal planning leverages the intuition that certain goals in planning domains may only be brought about if certain preconditions are first satisfied. For instance, in the Minecraft domain, one must be in possession of grain in order to bake bread. In Branavan et. al [**?**], they 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 set of subgoals, where each subgoal is a pair of predicates:

$$SG = \langle x_k, x_l \rangle$$

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.

Algorithm 1 Plan with Knowledge Base of Subgoals

subgoalSequence ← BFS(subgoalKB, goal)
plan = []
curState ← subgoalSequence.pop()
for subgoal ∈ subgoalSequence **do**
  plan ← ValueIteration(curState, subgoal)
  curState ← plan.getLastState()
end for
return plan;

In the case of `BRIDGEWORLD`, we might consider placing a block somewhere along the trench to be a subgoal. Then, we run Value Iteration to get from the agent's starting location to the point at which we've placed a block in the trench, stop, and run Value Iteration again from the first subgoal to the finish. There are a few problems with this approach, however.

Problem 1: Loss of generality One important thing to note about subgoals that *are* general enough to enhance an agent's planning abilities in a wide variety of state spaces is that they propose *necessary* claims about the domain that the agent occupies. If the subgoals are *contingent* (i.e. true in some state spaces of the domain but not in others), then they can be shown to completely lose their scalability. For instance, consider the task in `BRIDGEWORLD`, in which the agent must place a block in the trench that separates the agent from the goal. The subgoal $< blockInTrench, reachGoal >$ might be a perfectly useful subgoal in `BRIDGEWORLD`, but an adversary could easily come up with thousands of worlds in which such a subgoal would completely derail the agent's planner. Thus, many subgoals do not scale beyond a particular instance of a state space. In order for subgoals to be useful, they must be necessary claims about the domain, otherwise, one can always come up with a counter world (by definition of necessary).

Problem 2: Granular Planning The second problem is that those that subgoals that do scale across state spaces are often not useful. For instance, the vast majority of tasks in Minecraft are not so easily broken into useful, necessary subgoals. Movement, for instance, is particularly difficult. As stated before, scalable subgoals must be necessary preconditions for a particular goal, and such preconditions are often difficult to come up with in a way that actually makes planning easier. One idea would be to create the movement subgoal that the agent is one away from the goal - this is a poor choice however, since this subgoal is hardly useful unless we repeat it (i.e. the agent is one away from the next subgoal, and so on). The result is an extremely granular and low level planning system that is no better than standard Value Iteration. If necessary preconditions existed for many goal types, then subgoal planning would be a great approach. Unfortunately, coming up with such subgoals is not an easy task, and often the best we can do is to plan at such a low level that we lose any benefit of planning over subgoals to begin with

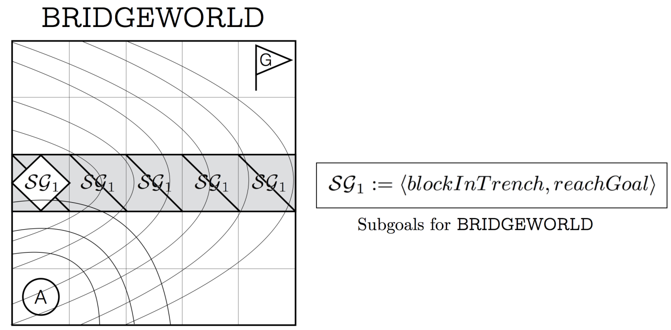


Figure 4: The agent re-explores a large portion of the state space once it finds $\mathcal{SG}_1$. Also note that this subgoal highlights **Problem 1**, in that it would be useless in many other Minecraft state spaces

Problem 3: Researching the Space The last problem with subgoal planning is that the use of subgoals actually requires that we research a huge portion of the state space. Consider the `BRIDGEWORLD` example in which the subgoal is to place a block along the trench somewhere - once we plan from the state in which a block has been placed at the trench, we research the entire first side of the trench. This problem only magnifies as you add more subgoals.

A final, but less significant problem, is that Subgoal planning still requires the use of Value Iteration, which does not scale well - if there is ever a case in which planning between two subgoals is at all complex, then Subgoal planning is out of luck.

2.3 OPTIONS

The options framework proposes incorporating high-level policies to 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:

$$\pi_0 : (s, a) \to [0, 1]$$

$$I_0 : s \to \{0, 1\}$$

$$\beta_0 : s \to [0, 1]$$

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.

As Konidaris and Barto point out, the classic options framework is not generalizable, as it does not enable an agent to transfer knowledge from one state space to another. Recently, Konidaris and Barto's [**?**] expand on the classic options framework and allow for a more portable implementation of options. Still, though, planning with options requires either that we plan in a mixed space of actions *and* options (which blows up the size of the search space), or requires that we plan entirely in the space of options. Additionally, providing an agent with an option policy is a difficult task for a human designer (especially if we want an optimal policy, which we do).

2.4 MACROACTIONS

Running Example

3 AFFORDANCES

Formally, an Affordance is defined as:

$\Delta = \langle p, g \rangle \longrightarrow \alpha$, where:

$$\alpha \subseteq \mathcal{A}$$

$$p : s \longrightarrow \{0, 1\}$$

$$g : s \longrightarrow \{0, 1\}$$

Where α is a subset of the agent's given set of actions $\mathcal{A}$, p is a *precondition* that is a predicate over states, and g is a *goal* or *subgoal* that is also a predicate over states.

The intuition is that in a huge number of planning scenarios, given a goal, the agent should be able to focus on only a subset of its available actions. The result is that the state-action space that the agent explores is astronomically smaller than in standard Value Iteration (especially in domains where the agent can change the environment to the degree of Minecraft). This parallels the intuition of Gibson's concept of an affordance, in which a human is capable of trimming down his or her considered action space by a huge amount when directed toward a particular goal. For instance, consider an agent with the standard Minecraft action set seen in Figure 2 - if the agent need only walk across a flat surface to reach the goal, it should not even bother trying to place blocks or destroy blocks. If it needs to dig a ten block hole, then the agent should not consider movement or placement.

The reason that each goal encodes information about the goal relevant to those actions is that it, given perfect subgoal knowledge for a particular planning task, the affordance formalism will find an optimal policy *extremely* quickly. We imagine extensions in which an agent gets stuck and must ask a human partner for help using natural language, and the resulting dialogue could endow the agent with subgoal knowledge.

Most importantly, within the Affordance framework, agent's may have *huge* action sets, since they will effectively prune their action set as needed. As a result, Affordance-equipped agents are able to solve a variety of problems across variable state-spaces with a single set of knowledge. This makes affordances extremely robust, as well as transferrable relative to subgoal planning and options.

Algorithm 2 Plan with Knowledge Base of Affordances

```
                                        ▷ Reachability Analysis
currAs ← affordKB.pruneAs(initS, d.goal, actions)
allStateActions ← ⟨State, ⟨Action⟩⟩
for s in allStateActions do
    nextAs ← affordKB.pruneAs(s, d.goal, allStateActions ⟨s⟩)
    allStateActions.add(s.do(nextAs))
end for
                                        ▷ ValueIteration
policy ← ValueIteration(d, allStateActions)
      return policy;
```

The Affordance formalism introduced above and expanded on in this paper resolves the weaknesses of these other frameworks by limiting the complexity of the seed knowledge required of the designer, while still providing enough knowledge to limit the search space but also maintain scalability.

We should be able to prove that given a "good" set of subgoals the agent will be able to reach one after the other with high probability. Therefore, in the case that the agent cannot reach a subgoal there is likely a better one. The agent should then prompt for a more specific subgoal that will better allow it to reach the next one.

4 EXPERIMENTS

We tested each planning algorithm (vanilla Value Iteration, Subgoal planning, Options, Affordance planning) on 5 distinct tasks, each with its own state space. For each task, the agent was given the same set of actions seen in Figure 2 and one block to place.

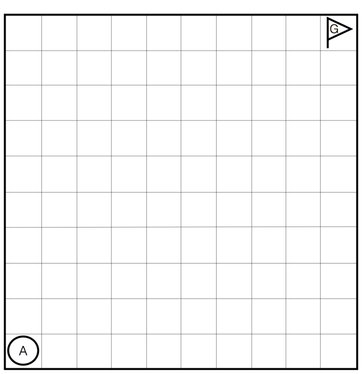
In the Knowledge Round, Options, Subgoals, Macro-actions, and Affordances were all given a hand crafted set of knowledge for each individual task. This round is intended to demonstrate the efficacy of each planning system, given perfect knowledge.

In the Transfer Round, we required that each system use one set of knowledge to solve all five planning scenarios. We ran the algorithms on the same set of tasks. This was intended to measure the transferability of each knowledge representation.

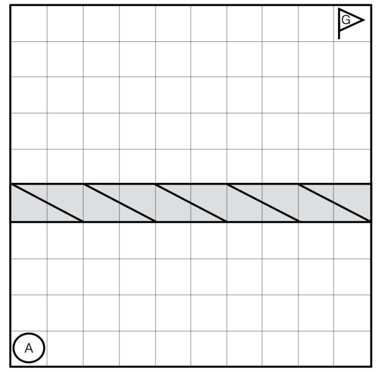
For each iteration, we measured the wall-clock time to find the goal, the percentage of the state-space explored, and the number of states that were re-explored (in the case of subgoals).

The set of tasks were inspired by those planning scenarios that the creators of each other planning system proposed.

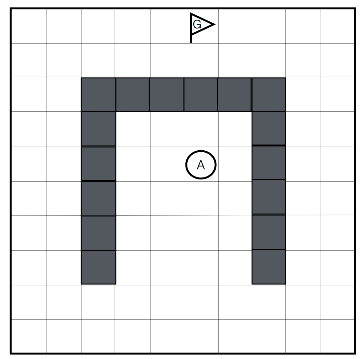
T1: `FLATWORLD` A 10x10 world with no obstacles.



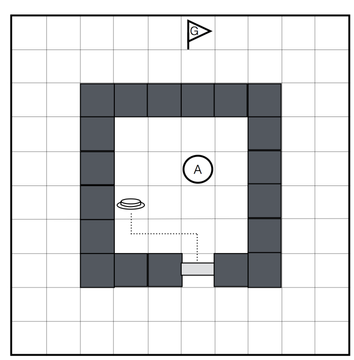
T2: `BRIDGEWORLD-10x10` A 10x10 world with a trench cutting across the center, separating the agent from the goal.



T3: `WALLWORLD` A 10x10 world in which the agent is nearly surrounded by an indestructible wall (i.e. it cannot destroy the grey blocks) and must first move farther away from the goal in order to reach the goal.



T4: `LIGHTWORLD` This is an adaptation of one of Konidaris' examples demonstrating the efficacy of portable options. A 10x10 world where the agent is surrounded on all sides by indestructible walls (i.e. the agent cannot destroy the grey blocks). The agent must press the button in order to open the door and proceed to the goal.



T5: `BIGWORLD` Identical to **T1:** `FLATWORLD`, but the dimensions of the world are 100x100.

5 RESULTS

5.1 ROUND A:

	Affordances	Options	Subgoals	VI
T1	1	1	0	0
T2	1	1	0	1
T3	1	1	1	0
T4	1	1	0	0
T5	1	1	0	0

5.2 ROUND B:

	Affordances	Options	Subgoals	VI
T1	1	1	0	0
T2	1	1	0	1
T3	1	1	1	0
T4	1	1	0	0
T5	1	1	0	0

6 CONCLUSION

We proposed a novel approach to representing knowledge in terms of *affordances* [1] that allows an agent to efficiently prune its action space based on domain knowledge. This pruning was shown to significantly reduce the number of state/action pairs the agent needs to evaluate in order to act optimally, and resulted in faster planning than subgoal planning, options, and vanilla value iteration. We demonstrated the efficacy as well as the transferability of the affordance model in a series of planning tasks in the Minecraft domain. In the future, we hope to learn affordances from experience as opposed to assuming that they are given by a human partner. Additionally, we hope to introduce some uncertainty into the action set that is pruned, in order to improve the effectiveness of the pruning. Lastly, we hope to incorporate aid from a human partner through natural language dialogue, in which the agent may ask for help when it is stuck and receive subgoal *hints* from a human companion.

References

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

[2] Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.

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

[4] 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)*, 2013.