Planning with Affordances

Abstract

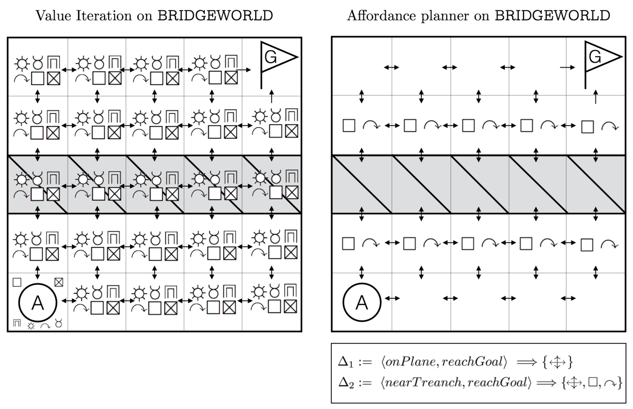
Current methods for exactly solving decision-making under uncertainty require exhaustive enumeration of all possible states and actions, leading to exponential run times, leading to the well-known "curse of dimensionality." Approaches to address this problem by providing the system with formally encoded 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* [2]. 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.

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 [3].

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 [2] 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 human actions [6, 7].

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



Value Iteration on BRIDGEWORLD
Affordance planner on BRIDGEWORLD
G
A
$\Delta_1 := \langle onPlane, reachGoal \rangle \implies \{\cdot\}$
$\Delta_2 := \langle nearTreanch, reachGoal \rangle \implies \{\cdot, \square, \frown\}$

Figure 1: In the above Minecraft planning problem `BRIDGEWORLD`, the agent must place a block in the trench in order to reach the goal (the trench is too wide to jump over). This problem was only solvable by the Affordance planner. (see Fig. 2 for a description of the action-symbols)

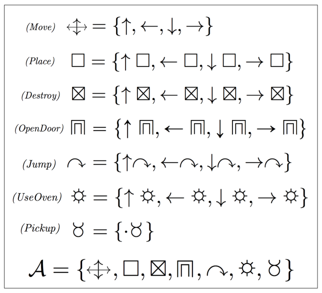


Figure 2: The set of all actions in the Minecraft domain

(MDP) [4]. We demonstrate that, like an option or macro-action, an affordance provides additional information to the agent, enabling more transferable and 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 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 goal. 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) [1] 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 representation of the state space S through the introduction of object classes, each of which has a set of attributes. The state space is represented as a collection of objects, which are instances of the aforementioned classes. Additionally, upon instantiation the attributes of the object's class are given a state (an assignment of values). Finally, the underlying MDP is the union of all the states of its objects [1].

Our motivation for using an OO-MDP instead of an MDP lies in the ability to formulate predicates over classes of objects. As we will see in section 3, this helps us form preconditions and goals that generalize beyond a particular instance of a state space.

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. See Algorithm 3 for the full pseudocode of the algorithm [8].

Algorithm 1 Value-Iteration($\mathcal{A}$, $\mathcal{R}$, $\mathcal{S}$, ϵ, γ)

```
1: while $\delta < \epsilon \frac{(1-\gamma)}{\gamma}$ do
2:     $U \leftarrow U'; \delta \leftarrow 0$
3:     for each state $s \in \mathcal{S}$ do
4:         $U'[s] \leftarrow \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s'} \Pr(s' \mid s, a) U[s']$
5:         if $|U'[s] - U[s]| > \delta$ then
6:             $\delta \leftarrow |U'[s] - U[s]|$
7:         end if
8:     end for
9: end while
10: return U;
```

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.

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 (see Equation 2). Thus, in these scenarios, our Affordance planner has a substantial advantage on classic Value Iteration. These weaknesses are well known [3],

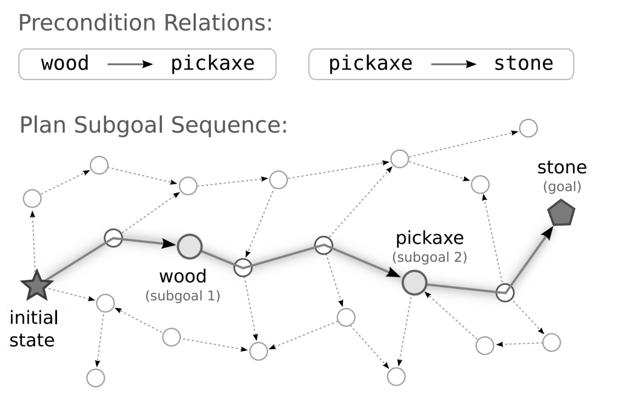


Figure 3: A picture from Branavan et. al's paper [9] displaying the high-level search in subgoal space on the baking bread task

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 [9], they explore learning subgoals from the Minecraft wiki and applying them in order to plan through a variety of problems in Minecraft. For further clarification, see Fig. **??**, directly from Branavan et. al's paper, displaying the high-level search in subgoal space for the baking bread example.

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 2 Plan with Knowledge Base of Subgoals

```
1: subgoalSequence ← BFS(subgoalKB, goal)
2: plan = []
3: curState ← subgoalSequence.pop()
4: for subgoal ∈ subgoalSequence do
5:     plan += ValueIteration(curState, subgoal)
6:     curState ← plan.getLastState()
7: end for
8: 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 $\langle blockInTrench, reachGoal \rangle$ 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

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

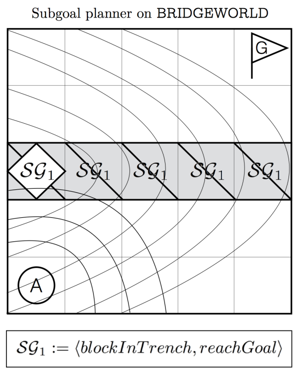

Subgoal planner on BRIDGEWORLD
G
$\mathcal{SG}_1$
$\mathcal{SG}_1$
$\mathcal{SG}_1$
$\mathcal{SG}_1$
$\mathcal{SG}_1$
A
$\mathcal{SG}_1 := \langle blockInTrench, reachGoal \rangle$

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

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:

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

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

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

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.

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 [5] 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.

This also allows the agent to prune way unnecessary actions in $\mathcal{A}$ in each specific planning task, making it possible to solve a engage with a large number of planning scenarios that may call for different actions. Furthermore, since actions may be pruned with respect to a given goal, agents may be endowed with huge action sets that enable them solve a variety of problems across variable state-spaces, yet the branching factor of an affordance agent's exploration will be significantly smaller, since actions that are not relevant to the current goal will be pruned. This makes the affordance formalism extremely robust, as well as transferrable relative to subgoal planning and options.

Algorithm 3 Affordance-Value-Iteration($\mathcal{A}$, $\mathcal{R}$, *initState*, *kb*, *goal*, ϵ, γ,)

```
 1: $\hat{\mathcal{A}} \leftarrow pruneActions(kb, initState, \mathcal{A}, goal)$
 2: $\hat{\mathcal{S}} \leftarrow genStates(kb, \hat{\mathcal{A}})$
 3: while $\delta < \epsilon\frac{(1-\gamma)}{\gamma}$ do
 4:     $U \leftarrow U'; \delta \leftarrow 0$
 5:     for each state $s \in \hat{\mathcal{S}}$ do
 6:         $U'[s] \leftarrow \mathcal{R}(s) + \gamma \max_{a\in\hat{\mathcal{A}}(s)} \sum_{s'} \Pr(s' \mid s, a) U[s']$
 7:         if $|U'[s] - U[s]| > \delta$ then
 8:             $\delta \leftarrow |U'[s] - U[s]|$
 9:         end if
10:     end for
11: end whilereturn U;
```

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.

4 EXPERIMENTS

We conducted a series of experiments in the Minecraft domain that tested each planning system on 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 to demonstrate the scalability of each system. In Table 5, we provide results on testing the planners across each system. For each data point, we ran the planning system 3 times and took the average result (they rarely deviated beyond $\frac{1}{10}$ of a second).

For the scenarios listed in Table 5, we limited the agent's action set across all planners so that it did not have access to block placement (□) and block destruction (⊠). The reason for excluding these two action types up is that, in Minecraft, the agent has an extraordinary ability to modify the state space via placing and destroying blocks. As a result, if an agent begins placing or destroying blocks in cases where it does not need to, the state-action space will explode exponentially and grow far too fast for (almost) any planner to finish in our lifetime.

Consider that the agent is capable of destroying and placing blocks in a 10x10x2 world; there are on the order of:

$$O\left(\sum_{n=1}^{10\cdot 10\cdot 2} \binom{10\cdot 10\cdot 2}{n}\right) \tag{2}$$

states, which is far too large to explore. We will demonstrate that our affordance model *is* capable of handling these types of actions, and can plan using them as in a bridge building scenario or tunnel digging scenario (among others) while the other planners cannot. In fact, with these actions, none of the other planning systems can solve even the most basic path planning (even on just a flat surface with no obstacles). We explore this ability in the results seen in Table 5

As discussed above, one of the major advantages of using Affordances to plan is that they enable an agent to have a massive action set, making Affordances effectively transfer between domains. We conducted an additional set of experiments on the 15x15 world with no obstacles `15WORLD`. For this round of testing, we varied the number of actions available to the agent (starting from $|\mathcal{A}| = 4$ up to $|\mathcal{A}| = 25$) and ran the planner on `15WORLD` with the same goal (to reach the goal in the corner).

An additional advantage of planning with Affordances is that the problems of block-placement and block-destruction illustrated by 2 are overcome. With Affordances, we are able to solve a variety of novel planning problems in the Minecraft domain, such as building a bridge to cross a long trench, or digging a hole through a wall to reach the goal (see Table 5). This is indeed a compelling result, as no other planning system is currently able to avoid falling prey to the state-space explosion mentioned above. Additionally, the malleability of Minecraft that causes this explosion is a reasonable model of the way that an agent in the real world is capable of modifying its surroundings. Thus, we foresee the Affordance planner as being extremely deft at handling real world planning scenarios.

5 RESULTS

As one can see from Table 5, in those cases where $|\mathcal{A}| = 21$ and $|\mathcal{A}| = 25$, the only planning algorithm to actually complete the tasks was the Affordance plan-

	Affordances	Subgoals	VI
10WORLD	**0.6s**	1.8s	1.1s
13WORLD	**2.5s**	10.1s	6.0s
15WORLD	**6.7s**	21.6s	11.8s
17WORLD	**16.6s**	45.4s	28.2s
20WORLD	**57.6s**	144.3s	140.5s
JUMPWORLD	**4.3s**	21.1s	10.1s
BREADWORLD	25.5s	**22.8s**	51.6s
DOORWORLD	**16.3s**	25.0s	25.3s
MAZEWORLD	**17.9s**	114.8s	37.6s
HARDWORLD	**34.5s**	215.9s	149.7s

Table 1: Tests on a variety of tasks without block placement and destruction actions

	Affordances	Subgoals	VI
$\lvert\mathcal{A}\rvert = 4$	?	?	?
$\lvert\mathcal{A}\rvert = 8$	?	?	?
$\lvert\mathcal{A}\rvert = 12$	?	?	?
$\lvert\mathcal{A}\rvert = 16$	?	?	?
$\lvert\mathcal{A}\rvert = 17$	?	?	?
$\lvert\mathcal{A}\rvert = 21$	?	DNF	DNF
$\lvert\mathcal{A}\rvert = 25$	?	DNF	DNF

Table 2: Plan on the simplest possible task (path planning in a flat plane with no obstacles) with incrementally larger action sets.

ner. This is because each of these cases scaled to include block destruction and block placement actions. Thus, any case in which these actions are required to complete the task at hand, only Affordance planning will succeed. This is significant, as Table 5 indicates that the Affordance planner plans more effectively than the other systems, but it can also handle novel problems involving those actions that alter the environment in sever ways. We also include a Bonus round indicating those tasks that only the Affordance planner was able to solve. Finally, since each Affordance is attached to a particular goal, a single knowledge base will scale across state-spaces and task types, causing Affordance planning to be extremely transferrable.

6 CONCLUSION

We proposed a novel approach to representing knowledge in terms of *affordances* [2] 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 providing them directly to the agent. Additionally, we hope to introduce 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] Michael L. Littman Carlos Diuk, Andre Cohen. An object-oriented representation for efcient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, 2008.

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

[3] 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.

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

[5] 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.

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

[7] 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.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education Inc, third edition, 2010.

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