Affordance-Aware Planning

David Abel & Gabriel Barth-Maron, James MacGlashan, Stefanie Tellex
Department of Computer Science, Brown University
{dabel,gabrielbm,jmacglashan,stefie10}@cs.brown.edu

***Abstract*—Planning algorithms for non-deterministic domains are often intractable in large state spaces due to the well-known curse of dimensionality. Existing approaches to planning in large stochastic state spaces fail to prevent autonomous agents from considering many actions that are obviously irrelevant to a human solving the same task. To solve this problem, we formalize the notion of *affordances* as knowledge added to an Object Oriented Markov Decision Process (OO-MDP) D: Should we specify the fact that we are adding affordances to an OO-MDP in particular?. Affordances prune actions on a state-by-state basis in a way that is not specific to a particular reward function or state-space. This action pruning reduces the number of state-action pairs the agent must evaluate in order to behave nearly optimally. Furthermore, we show that an agent can learn affordances through unsupervised experience, and that learned affordances can equal or surpass the performance of those provided by experts. We demonstrate our approach in the state-rich Minecraft domain, showing significant increases in speed and reductions in state-space exploration during planning without loss in quality of the synthesized policy. Additionally, we employ affordance-aware planning on a robot in a cooking assistant task.**

I. Introduction

Robots operating in unstructured, stochastic environments face a highly difficult planning problem [4, 14]. Robotics planning problems are classically formalized as a stochastic sequential decision making problem, modeled as a Markov Decision Process (MDP). In these problems, the agent must find a mapping from states to actions for some subset of the state space that enables the agent to achieve a goal while minimizing costs along the way. However, many robotics problems are of such exceeding complexity that modeling them as an MDP leads to in an immense state-action space. This large state-action space, in turn, restricts the classes of robotics problems that are computationally tractable. **D: We say "robotics problems" way too often here**

For example, when a robot is manipulating objects in an environment an object can be placed anywhere in a large set of locations. The size of the state space increases exponentially with the number of objects, which bounds the placement problems that the robot is able to expediently solve.

To address this state-action space explosion, prior work has explored adding knowledge to the planner, such as options [25] and macroactions [5, 20]. However, while these methods allow the agent to search more deeply in the state space, they add high-level actions to the planner which *increase* the size of the state-action space. The resulting augmented space is even larger, which can have the paradoxical effect of increasing the search time for a good policy [13]. **D: Should we mention heuristic planners or action pruning work here too?**

Deterministic forward-search algorithms like hierarchical task networks (HTNs), such as SHOP [19], and temporal logical planning (TLPLAN) [2, 3], add knowledge to the planner that greatly increases planning speed, but do not generalize to stochastic domains. Additionally, the knowledge provided to the planner is must be given by a domain expert, reducing the agent's autonomy.

To address these issues, we propose augmenting an MDP with a formalization of *affordances*. An affordance [10] specifies which actions an agent should consider in different states of the world in order to achieve its goal**D: I added "the world" here to acknowledge the existence of affordances outside our context, but now "its" is ambiguous. Thoughts?**. By applying affordances to planning, we prune the agent's action set to focus on aspects of the environment that are most relevant to solving its current goal and avoid exploration of irrelevant parts of the state-action space. Affordances are not specific to a particular reward function or state space, and provide the agent with transferable knowledge that is effective in a wide variety of problems. Moreover, our methods enable a single agent to autonomously learn affordances through unsupervised experience, making affordances a concise, transferable, and learnable means of representing useful planning knowledge.

Our experiments demonstrate that affordances provide dramatic speedups for a variety of planning tasks compared to baselines and apply across different state-spaces. We conduct experiments in the game Minecraft, and on a robotic cooking assistant.

II. Affordances

D: We shouldn't immediately start with MDPs. We need to have some text here before transitioning to this subsection.

A. MDPs

A classic MDP is a five-tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a state-space; $\mathcal{A}$ is the agent's set of actions; $\mathcal{T}$ denotes $\mathcal{T}(s' \mid s, a)$, the transition probability of an agent applying action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ and arriving in $s' \in \mathcal{S}$; $\mathcal{R}(s, a, s')$ denotes the reward received by the agent for applying action a in state s and and transitioning to state s'; and $\gamma \in [0, 1)$ is a discount factor that defines how much the agent prefers immediate rewards over distant rewards (the agent prefers to maximize immediate rewards as γ decreases). A classic way to provide a factored representation of an MDP state is to represent each MDP state as a single feature vector.

B. OO-MDPs

An Object-Oriented Markov Decision Process (OO-MDP) [8] can be used to represent an MDP. Unlike the vectorized state of an MDP, an OO-MDP state is a collection of objects, $O = \{o_1, \ldots, o_o\}$. Each object o_i belongs to a class, $c_j \in \{c_1, \ldots, c_c\}$. Every class has a set of attributes, $Att(c) = \{c.a_1, \ldots, c.a_a\}$, each of which has a domain, $Dom(c.a)$, of possible values. The collection of attribute values of a given object is termed that object's state, $o.state$. A vectorized MDP state rcan be equivalently understood as the set of all the object states, $s \in \mathcal{S} = \cup_{i=1}^{o}\{o_i.state\}$, in an OO-MDP.

E: we need to fix this formalism as per what we talked about Dave – i.e. it's an object or a mapping... D: Ellis how about this? It's a little weird for the mapping to utilize elements of the tuple, but I think this paints the right picture. I like the tuple without L much more just in terms of simplicity, but I think we need it (or n, and integer specifying action set size). Alternatively we could change M to be a Dirichlet-multinomial?

C. Affordance Formalism

We define an affordance, Δ, as the tuple $\langle p, g, M, L, \delta \rangle$, where:

- p is a predicate on states, $s \longrightarrow \{0, 1\}$ representing the *precondition* for the affordance.
- g is a *goal type*, representing the type of problem the agent is solving.
- M is a multinomial over $\mathcal{A}$, the OO-MDP action space.
- L is a multinomial over the integers from 1 to $|\mathcal{A}|$, where $|\mathcal{A}|$ is the size of the OO-MDP action set.
- δ is a mapping from an OO-MDP state s and a goal type G to a set of actions $\mathcal{A} \subset A$.

The precondition and goal type refer to predicates that are defined in the OO-MDP definition. M represents how relevant the affordance finds each action. L represents the number of relevant actions for the affordance. The mapping δ represents a function that determines the relevant action possibilities for the affordance in each state.

$$\delta(s, G) = \begin{cases} A, & \text{if } \Delta.p(s) \wedge G \models \Delta.g \\ \varnothing, & \text{otherwise} \end{cases} \tag{1}$$

We compute A, the set of actions suggested by the affordance, by taking a sample n from the multinomial over action set sizes L. Then, we define A to be the result of taking n samples from the multinomial over actions, M:

$$n \leftarrow L \tag{2}$$

$$A \leftarrow_n M \tag{3}$$

Our definition of affordances builds on OO-MDPs. Using OO-MDP predicates for affordance preconditions and goal types allows for state space independence We achieve this independence as predicates generalize across specific state-spaces. **E: I think a sentence explaining/emphasizing this would be good since it's not entirely obvious at a first pass why OO-MDPs give you state-space independence. D: I added a note here but it sounds superfluous to me. I think we should get rid of it. Maybe we should just put a small note in the same sentence about predicates?** Thus, a planner equipped with affordances can be used in any number of different environments. For instance, the affordances defined for navigation problems can be used in any task regardless of the spatial size of the world, number of objects in the world, and specific goal the agent is trying to satisfy.

D. Affordance-Aware Planning

Affordances are used to restrict the action set of any planner on a state by state basis to a subset of the action set. Namely, for each state, the union of all action sets provided by all affordances, $\mathcal{A}_\Delta$ is a subset of the full action set $\mathcal{A}$.

$$\mathcal{A}_\Delta = \left(\bigcup_{\Delta} \Delta.\delta(s, G) \right) \subseteq \mathcal{A} \tag{4}$$

A domain expert may specify the multinomials M and L directly, allowing $\mathcal{A}$ to be determined probabilistically or deterministic if the expert desires. If an expert is not involved, the agent will learn L and M during training time for each affordance.

Our goal for each state is that the set of affordance actions, $\mathcal{A}_\Delta$, is equal to the set of optimal actions, $\mathcal{A}^o$:

$$\Pr(\mathcal{A}_\Delta = \mathcal{A}^o \mid s, \Delta_1 \ldots \Delta_K) = 1 \tag{5}$$

This probability can be equivalently understood as the probability that each optimal action is in $\mathcal{A}_\Delta$ and each non-optimal action is not in $\mathcal{A}_\Delta$:

$$= \prod_{i}^{|\mathcal{A}^o|} \sum_{j}^{|\Delta|} \Pr(a_i \in \Delta_j \mid s, \Delta_j)$$

$$\times \left[1 - \prod_{i}^{|\overline{\mathcal{A}^o}|} \sum_{j}^{|\Delta|} \Pr(b_i \in \Delta_j \mid s, \Delta_j) \right] \tag{6}$$

Where a_i is an optimal action for the given state, $a_i \in \mathcal{A}^o$, and b_i is a non-optimal action for the given state, $b_i \in \overline{\mathcal{A}^o}$. Equation 6 represents the probability that each optimal action is in affordance action set, and that each non-optimal action is not in the affordance action set.

If M and L are supplied by a domain expert, then $\Pr(a_i \in \Delta_j(s) \mid s, \Delta_j)$ is simply defined by the multinomials $\Delta_j.M$ and $\Delta_j.L$. **D: Write this better in math? Should sync up with the learning equation below.. (that uses the dirmult)** Even with expert domain knowledge it is often unclear how to set up an optimal knowledge base, and it can be arduous to specify a large number of affordance. We developed a learning process that requires minimal expert intervention, detailed in the following section.

E. Learning Affordances

To learn an affordance knowledge base, we require that a domain expert supply a set of relevant domain specific predicates, $\mathcal{P}$ and possible goals the agent will have to solve, $\mathcal{G} \subset \mathcal{P}$. Additionally, a domain expert must provide a means of generating candidate state spaces in which each goal $g \in \mathcal{G}$ may be satisfied (i.e. the function $createTestWorld(g)$ at line 5 in Algorithm 1).

We compute each learned affordance's contributed action set using a Dirichlet-multinomial distribution:

$$\Pr(a_i \in \Delta_j(s) \mid s, \Delta_j) = DirMult(\Delta_j.\alpha, \Delta_j.n) \quad (7)$$

D: I could use a second pair of eyes to compare this to the bit above about how we define this probability for experts. Where $\Delta_j.\alpha$ denotes the hyper parameter vector for the Dirichlet-multinomial, and $\Delta_j.n$ indicates the number of samples to draw. We define $\Delta_j.n$ to be a sample from a Dirichlet distribution over the affordances hyper parameter vector β:

$$\Delta_j.n \sim Dir(\Delta_j.\beta) \quad (8)$$

Provided that the Dirichlet-multinomial and Dirichlet associated with each affordance are properly specified, the probability of retrieving the optimal action set across all affordances, as seen in Equation 5, approaches 1 as the counts of α and β increase. Properly setting these counts is a difficult problem, however. The learning problem is then reduced to specifying reasonable (p, g) pairs, and solving for α and β values for each affordance.

Algorithm 1 $learn(\mathcal{P}, \mathcal{G})$

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

The full learning algorithm may be seen in Algorithm 1. **D: Is this too late in the section already?**.

To learn p and g for each affordance, the agent forms a set of candidate affordances Δ with every combination of $\langle p, g \rangle$, for $p \in \mathcal{P}$ and $g \in \mathcal{G}$, as seen in line 1-3 of Algorithm 1. Affordances whose predicate-goal pairs don't provide useful information are thrown out by the $removeLowInfoAffordances(knowledgeBase)$ function call.

To learn the α and β for each affordance, we synthesize a policy of m goal-annotated OO-MDPs that have small state spaces, but share similar characteristics to the state spaces the agent might expect to see in more complex environments. For example, the agent learns to build towers of blocks in small state spaces that can be solved exactly (i.e. a state space of several thousand states), but generalizes its knowledge to worlds that are too large to solve with exact algorithms (state spaces of hundreds of thousands of states). With this policy, we know the optimal action in each state and can generalize this optimality to larger state spaces.

Algorithm 2 $updateParameters(knowledgeBase, \pi)$

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

We update α and β according to Algorithm **??**. For each optimal policy, for each affordance, α is set to the number of states in which an action was optimal when its affordance's predicate was true and goal was entailed by the present goal. Additionally, we define β as a vector representing counts of integers 1 to $|\mathcal{A}|$. Then, for each optimal policy, we count the number of unique actions that were optimal for each activated affordance Δ_i, and increment that value for $\Delta_i.\beta$. This captures how large or small optimal action sets are expected to be for each affordance. **D: Need to add beta counts to algorithm 2. (a bit tricky to do concisely so I'm taking a bit of time on it.**

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

D: I replaced most of the pseudocode with math instead. I think introducing the mapping δ for each affordance makes everything much clearer

E: isn't this just hard affordances? shouldn't we mention them by name – also why are we talking about our experiments before our experiments section. I'm a bit confused by the purpose of the above sentenceD: We got rid of the notion of hard affordances from the paper for simplicity. For experiments, we include results from them. I'm taking this sentence out.

E: This is as far as I got on my first pass (I added the above paragraph)

III. Experiments

We use Minecraft as our planning and evaluation domain. Minecraft is a 3-D blocks world game in which the user can place and destroy blocks of different types. It serves as a model for a variety of complex planning tasks involving assembly, crafting, and construction. Minecraft's physics and action space are expressive enough to allow very complex systems to be created by users, including logic gates and

LIST OF EXPERT AFFS

functional scientific graphing calculators[1]. Minecraft serves as a model for robotic tasks such as cooking assistance, assembling items in a factory, and object retrieval. As in these tasks, the agent operates in a very large state-action space in an uncertain environment.

A. Minecraft Tests

We conducted a series of experiments in the Minecraft domain that compared the performance of several OO-MDP solvers without affordances to their affordance-aware counterparts. We created a set of expert affordances from our background knowledge of the domain. Additionally, we ran our full learning process and learned affordances for each task. We compared standard paradigm planners (Real Time Dynamic Programming and Value Iteration) with their expert-affordance-aware counterparts and with their learned-affordance-aware counterparts.

For the expert affordances, we provided the agent with a knowledge base of 17 affordances, which are listed in Figure III-A. Our experiments consisted of 5 common tasks in Minecraft, including constructing bridges over trenches, smelting gold, tunneling through walls, and digging to find an object. We tested on worlds of varying size and difficulty to demonstrate the scalability and flexibility of the affordance formalism.

For the learning process, the training data consisted of 20 simple state spaces of each map type (100 total), each approximately a 1,000-10,000 state world with randomized features that mirrored the agent's actual state space. The same training data was used for each test state space.

The evaluation metric for each trial was the number of Bellman updates that were executed by each planning algorithm, as well as the CPU time taken to find a plan. Value Iteration was terminated when the maximum change in the value function was less than 0.01. RTDP terminated when the maximum change in the value function was less than 0.01 for fifty consecutive policy rollouts, or the planner failed to converge after 2500 rollouts. We set the reward function to -1 for all transitions, except transitions to states in which the agent was on lava, which returned -10. The goal was set to be terminal. The discount factor was set to $\lambda = 0.99$. For all experiments, movement actions (move, rotate, jump) had a small probability (0.05) of incorrectly applying a different movement action.

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

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

2) Temporally Extended Actions: Additionally, we compared our approach to Temporally Extended Actions: Macroactions and Options. We compared RTDP with expert affordances, expert Macroactions, and expert Options, as well as the combination of affordance, macro actions, and options. We conducted these experiments with the same configurations as our regular Minecraft experiments. The option policies and macro actions provided were hand coded by domain experts.

B. Robotic Task

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

IV. Results

A. Baxter

Fig. 1. Placeholder for baxter results/image

B. Minecraft: Expert vs Learned vs None

D: These are preliminary results and will not be included in the final. I will run experiments on more and larger worlds (currently showing average after planning in 5 worlds per task type - worlds were 2x3x4, I'll run on 8x8x8).

State Space	RTDP	Learned	Learned Threshold	Expert
Trench	-	-	-	-
Mining	-	-	-	-
Smelting	-	-	-	-
Wall	-	-	-	-
Tower	-	-	-	-

TABLE I
LEARNED AFFORDANCE RESULTS: AVG. NUMBER OF BELLMAN UPDATES PER CONVERGED POLICY (AVERAGE OVER 5 WORLDS PER GOAL TYPE)

State Space	RTDP	Learned Soft	Learned Hard	Expert
Trench	0.96s	1.17s	0.77s	**0.47s**
Mining	0.34s	0.54s	**0.21s**	0.26s
Smelting	0.91s	1.25s	**0.70s**	0.81s
Wall	1.12s	1.49s	**0.78s**	0.85s
Tower	0.95s	1.04s	**0.78s**	0.88s

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

C. Minecraft: Learning rate

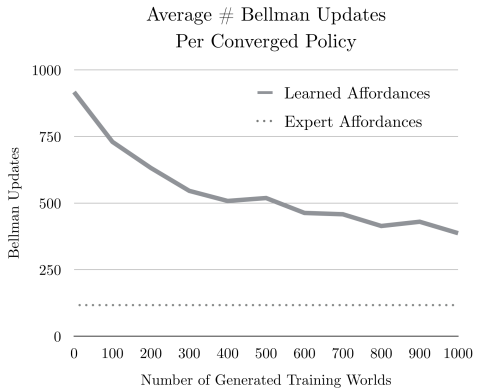

Fig. 2. Placeholder - will recollect this data given recent updates

D. Options

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

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

V. RELATED WORK

In this section, we discuss the differences between affordance-aware planning and other forms of knowledge engineering that have been used to accelerate planning. We divide these approaches into those that are built to plan in stochastic domains, and those that are deterministic planners.

A. Stochastic Approaches

1) 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* [12] and *options* [25]. Macro-actions are actions that always execute the same sequence of primitive actions. Options are defined with high-level policies that accomplish specific sub tasks. For instance, when an agent is near a door, the agent can engage the 'door-opening-option-policy', which switches from the standard high-level planner to running a policy that is hand crafted to open doors.

Although the classic options framework is not generalizable to different state spaces, creating *portable* options is a topic of active research [17, 15, 22, 6, 1, 16].

Given the potential for unhelpful temporally extended actions to negatively impact planning time [13], we believe combing affordances with temporally extended actions may be especially valuable because it will restrict the set of temporally extended actions to those useful for a task. We conducted a set of experiments to investigate this intuition.

2) Action Pruning: Sherstov and Stone [24] considered MDPs with a very large action set and for which the action set of the optimal policy of a source task could be transferred to a new, but similar, target task to reduce the learning time required to find the optimal policy in the target task. The main difference between our affordance-based action set pruning and this action transfer work is that affordances prune away actions on a state by state basis, where as the learned action pruning is on per task level. Further, with lifted goal descriptions, affordances may be attached to subgoal planning for a significant benefit in planning tasks where complete subgoal knowledge is known.

Rosman and Ramamoorthy [23] provide a method for learning action priors over a set of related tasks. Specifically, they compute a Dirichlet distribution over actions by extracting the frequency that each action was optimal in each state for each previously solved task.

There are a few limitations of the actions priors work that affordance-aware planning does not possess: (1) the action priors can only be used with planning/learning algorithms that work well with an ϵ-greedy rollout policy; (2) the priors are only utilized for fraction ϵ of the time steps, which is typically quite small; and (3) as variance in tasks explored increases, the priors will become more uniform. In contrast, affordance-aware planning can be used in a wide range of planning algorithms, benefits from the pruned action set in every time step, and the affordance defined lifted goal-description enables higher-level reasoning such as subgoal planning.

3) Heuristics: Heuristics in MDPs are used to convey information about the value of a given state-action pair with respect to the task being solved and typically take the form of either *value function initialization*, or *reward shaping*. Initializing the value function to an admissible close approximation of the optimal value function has been shown to be effective for LAO* and RTDP [11].

Reward shaping is an alternative approach to providing heuristics. The planning algorithm uses a modified version of the reward function that returns larger rewards for state-action pairs that are expected to be useful, but does not guarantee convergence to an optimal policy unless certain properties of the shaped reward are satisfied [21].

A critical difference between heuristics and affordances is that heuristics are highly dependent on the reward function and state space of the task being solved, whereas affordances are state space independent and transferable between different reward functions. However, if a heuristic can be provided, the combination of heuristics and affordances may even more greatly accelerate planning algorithms than either approach alone.

B. Deterministic Approaches

There have been several successful attempts at engineering knowledge to decrease planning time for deterministic planners. These are fundamentally solving a different problem from what we are interested in, but there approaches are

interesting to consider. **D: Need to rephrase this with justification for dealing with deterministic planners**.

C. Hierarchical Task Networks

D: I think we should have a shoutout to Branavan's Learning High Level Plans from Text paper in this section (and include subgoal planning as part of this section

E: I've been writing traditional as I expect we'll discover some HTNs that grapple with the issues stated below – which we should probably cite Traditional Hierarchical Task Networks (HTNs) employ *task decompositions* to aid in planning. The goal at hand is decomposed into smaller tasks which are in turn decomposed into smaller tasks. This decomposition continues until primitive tasks that are immediately achievable are derived. The current state of the task decomposition, in turn, informs constraints which reduce the space over which the planner searches.

At a high level both HTNs and affordances fulfill the same role: both achieve action pruning by exploiting some form of supplied knowledge. HTNs do so with the use of information regarding both the task decomposition of the goal at hand and the sorts constraints that said decomposition imposes upon the planner. Similarly, affordances require knowledge as to how to extract values for propositional functions of interest by querying the state.

However there are three of essential distinctions between affordances and traditional HTNs. (1) HTNs deal exclusively with deterministic domains as opposed to the stochastic spaces with which affordances grapple. As a result they produce plans and not policies. (2) Moreover, HTNs do not incorporate reward into their planning. Consequently, they lack mathematical guarantees of optimal planning. **E: I think.. We should double check this.** (3) On a qualitative level, the degree of supplied knowledge in HTNs surpasses that of affordances: whereas affordances simply require relevant propositional functions, HTNs require not only constraints for sub-tasks but a hierarchical framework of arbitrary complexity. Thus, despite a superficial similarity between affordances and HTNs wherein both employ supplied knowledge, the two deal with disparate forms of planning problems; HTN's planning problem is deterministic, reward-agnostic and necessitates a plethora of knowledge while affordances solve a planning problem that is stochastic, reward-aware and requires only relatively basic knowledge about the domain. **E: Need citations for HTNs**

D. Temporal Logic

Bacchus and Kabanza [2, 3] provided planners with domain dependent knowledge in the form of a first-order version of linear temporal logic (LTL), which they used for control of a forward-chaining planner. With this methodology, STRIPS style planner may be guided through the search space by checking whether candidate plans do not falsify a given knowledge base of LTL formulas, often achieving polynomial time planning in exponential space.

The primary difference between this body of work and affordance-aware planning is that affordances may be learned (increasing autonomy of the agent), while LTL formulas are far too complicated to learn effectively, placing dependence on an expert.

VI. CONCLUSION

D: Conclusion could use some work/rewriting We proposed a novel approach to representing transferable knowledge in terms of *affordances* [10] that allows an agent to efficiently prune its action space based on domain knowledge, providing a significant reduction in the number of state-action pairs the agent needs to evaluate in order to act near optimally. We demonstrated the effectiveness of the affordance model by comparing standard MDP solvers to their affordance-aware equivalents in a series of challenging planning tasks in the Minecraft. domain. Further, we designed a full learning process that allows an agent to autonomously learn useful affordances that may be used across a variety of task types, reward functions, and state-spaces, allowing for convenient extensions to robotic applications. The results support the effectiveness of the learned affordances, suggesting that the agent may be able to discover novel affordance types and learn to tackle new types of problems on its own.

We compared the effectiveness of augmenting planners with affordances compared to temporally extended actions. The results suggest that affordances, when combined with temporally extended actions, provide substantial reduction in the portion of the state-action space that needs to be explored.

Lastly, we deployed an affordance-aware planner on a robotic task with a massive state space. **D: Need to flesh out once we have more detail**.

In the future, we hope to automatically discover useful state-space-specific-subgoals online - a topic of some active research [18, 7]. This will allow for affordances to plug into high-level subgoal planning, which will reduce the size of the explored state-action space and improve transferability across task types. Additionally, we hope to decrease the amount of knowledge given to the planner by implementing Incremental Feature Dependency Discovery [9], which will allow our affordance learning algorithm to discover novel preconditions that will further enhance action pruning. **D: Maybe put in a note about the forward search sparse sampling algorithm? Or perhaps the Bayesian planner?**

REFERENCES

[1] 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.

[2] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to control search in a forward chaining planner. In *In Proceedings of the 3rd European Workshop on Planning*, pages 141–153. Press, 1995.

[3] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:2000, 1999.

[4] Mario Bollini, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. Interpreting and executing recipes with a cooking robot. In *Proceedings of International Symposium on Experimental Robotics (ISER)*, 2012.

[5] 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.

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

[7] Özgür Şimşek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 816–823, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102454. URL http://doi.acm.org/10.1145/1102351.1102454.

[8] C. Diuk, A. Cohen, and M.L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, 2008.

[9] Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 881–888, New York, NY, USA, 2011. ACM. URL http://www.icml-2011.org/papers/473_icmlpaper.pdf.

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

[11] 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.

[12] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229. Morgan Kaufmann Publishers Inc., 1998.

[13] 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.

[14] Ross A. Knepper, Stefanie Tellex, Adrian Li, Nicholas Roy, and Daniela Rus. Single assembly robot in search of human partner: Versatile grounded language generation. In *Proceedings of the HRI 2013 Workshop on Collaborative Manipulation*, 2013.

[15] 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.

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

[17] 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.

[18] Amy Mcgovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *In Proceedings of the eighteenth international conference on machine learning*, pages 361–368. Morgan Kaufmann, 2001.

[19] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 968–973, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1624312.1624357.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.