Affordances-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 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 reduce the size of the state-action space we formalize the notion of *affordances* as goal-oriented knowledge added to an Object Oriented Markov Decision Process (OO-MDP). Affordances prune actions based on the current state and the robot's goal, reducing the number of state-action pairs the planner must evaluate in order to synthesize a near optimal policy. We show that an agent can learn affordances through 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, with minimal loss in quality of the synthesized policy. Additionally, we employ affordance-aware planning on a Baxter robot, demonstrating it is able to assist a person performing a collaborative cooking task.**

I. Introduction

Robots operating in unstructured, stochastic environments such as a factory floor or a kitchen face a difficult planning problem due to the large state space and inherent uncertainty due to unreliable perception and actuation [4, 14]. Robotic planning tasks are often 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 tasks are so complex that modeling them as an MDP results in a massive state-action space, which in turn restricts the types of robotics problems that are computationally tractable. 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. Moreover, the task is made further difficult by the fact that, most of these locations are irrelevant: when making brownies, the oven and flour are important, while the soy sauce and sauté pan are not. For a different task, such as stir-frying broccoli, a different set of objects and actions are relevant.

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]. Deterministic forward-search algorithms like hierarchical task networks (HTNs) [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 by these methods is quite extensive, reducing the agent's autonomy.

Instead, we propose augmenting an MDP with a formalization of *affordances*. **D: Insert aft line here**. An affordance [10] focuses an agent's attention on aspects of the environment that are most relevant to solving its current goal and avoid exploration of irrelevant parts of the world. 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, we show that an agent can learn affordances through 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, which has a very large state-action space, and on a real-world robotic cooking assistant.

II. Technical Approach

We formalize affordances as knowledge added to a Markov Decision Process (MDP). An 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 future rewards (the agent prefers to maximize immediate rewards as γ decreases).

Our representation of affordances builds on an Object-Oriented MDPs (OO-MDP) [8]. An OO-MDP efficiently represents the state of an MDP through the use of objects and predicates. 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. OO-MDPs enable planners to use 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. We use OO-MDP predicates as features for action pruning, allowing for state space independence.

A. Modeling $\mathcal{A}^*$

Our goal is to formalize affordances in a way that enables a planning algorithm to prune away suboptimal actions in each state. We define the optimal action set, $\mathcal{A}^*$, for a given state s and goal G as:

$$\mathcal{A}^* = \{a \mid Q^*_G(s,a) = V^*_G(s)\}, \tag{1}$$

where, $Q^*_G(s,a)$ and $V^*_G(s)$ represent the optimal Q function and value function, respectively. **D: This makes a pretty big jump from introducing an MDP. Do we need to introduce the bellman equation/value function above, too?E: I honestly don't think we need to mention the Q fn or V fn since most people have an intuitive sense of "optimal" action and the functions don't have any immediate bearing on anything specific to our research**

We aim to learn a probability distribution over the optimal action set for a given state (s), goal (G), and knowledge base (K) from which action pruning may be informed (see Section II-B):

$$\Pr(\mathcal{A}^* \mid s, G, K) \tag{2}$$

We formalize our knowledge base, K, as a set of paired preconditions and goal types, $\{(p_1, g_1) \dots (p_{|K|}, g_{|K|})\}$,**E: maybe overly picky but since our KB includes theta isn't it incorrect to say we have —K— deltas** and a parameter vector θ. We abbreviate each pair (p_j, g_j) to δ_j for simplicity. Each precondition $p \in \mathcal{P}$ is a *predicate* in predicate space, $\mathcal{P}$, defined by the OO-MDP, and $g \in \mathcal{G}$ is a *goal type* in goal space. We assume that the goal space consists of logical expressions of state predicates. A goal type specifies the sort of problem the agent is trying to achieve. In the context of Minecraft, a goal type might refer to the agent retrieving an object of a certain type from the environment, reaching a particular location, or crafting a structure. The parameter vector, θ represents model parameters for the probability distribution **E: I think it's worthwhile expanding this explanation – it's not obvious what theta is at a first pass to me**.

We rewrite Equation 2 replacing K with its constituents:

$$\Pr(\mathcal{A}^* \mid s, G, K) = \Pr(\mathcal{A}^* \mid s, G, \delta_1 \dots \delta_{|K|}, \theta) \tag{3}$$

When a precondition and goal type ($\langle p, g \rangle = \delta$) is paired with a state and goal to be solved in a planning problem, we can evaluate whether δ is relevant by checking if the predicate of δ is true in the current state and if the current goal is equivalent to the goal type of δ. This relevance is defined with the indicator function f:

$$f(\delta, s, G) = \begin{cases} 1 & \delta.p(s) \wedge G == \delta.g \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

Evaluating f for each δ_j given the current state and goal gives rise to a set of binary features, $\phi_j = f(\delta_j, s, G)$, which we use to reformulate our probability distribution:

$$\Pr(\mathcal{A}^* \mid s, G, \delta_1 \dots \delta_{|K|}, \theta) = \Pr(\mathcal{A}^* \mid \phi_1, \dots, \phi_{|K|}, \theta) \tag{5}$$

We assume each action's optimality is independent from all other actions':

$$= \prod_{i=1}^{|\mathcal{A}|} \Pr(a_i \in \mathcal{A}^* \mid \phi_1, \dots, \phi_{|K|}, \theta_i) \tag{6}$$

where θ_i represents the set of parameters relevant to modeling the probability distribution of action $a_i \in \mathcal{A}^*$. Henceforth, we abbreviate $a_i \in \mathcal{A}^*$ to a_i.

B. Learning $\mathcal{A}^*$

Though this distribution may be modeled in a number of ways making this approach quite flexible, we have chosen to model it as a Naive-Bayes. First we factor using Bayes' rule:

$$= \prod_{i=1}^{|\mathcal{A}|} \frac{\Pr(\phi_1, \dots, \phi_{|K|}, \mid a_i, \theta) \Pr(a_i \mid \theta)}{\Pr(\phi_1, \dots, \phi_{|K|} \mid \theta_i)} \tag{7}$$

Next we assume that each feature is conditionally independent of the others, given whether the action is optimal:

$$= \prod_{i=1}^{|\mathcal{A}|} \frac{\Pr(a_i \mid \theta) \prod_{j=1}^{|K|} \Pr(\phi_j \mid a_i, \theta)}{\Pr(\phi_1, \dots, \phi_{|K|} \mid \theta)} \tag{8}$$

E: It's confusing to me that order of terms in the numerator was swapped. Is there a reason for this?

Finally, we assume a uniform prior **E: why are we using a uniform prior instead of just frequency of action optimality in the training set? Maybe this is explained later... In any case I don't think we need to specify what our prior is – at least not at his point in the paper** on the optimality of each action.

This distribution fully describes the model used by an affordance-aware planner.

C. Affordance-Aware Planning

Given a probability distribution over the optimal action set, any typical planner can be made to be an affordance-aware planner by pruning its action set in each state based on the probability distribution. In this work we propose several approaches to perform this pruning.

The first method of pruning is to impose an expert defined threshold on the posterior. In this way, the affordances prune away any actions whose probability of being optimal is below the provided threshold for each state. For experiments, this method is denoted as `LARTDP`, and was set to $\frac{0.2}{|\mathcal{A}|}$, where $|\mathcal{A}|$ is the size of the full action space of the OO-MDP.

Affordances may also be specified by a domain expert using a discriminative OR model in place of the Naive Bayes. In this method, the expert specifies a set of actions associated with a precondition-goal type pair. For instance, if an agent is standing above a block of buried gold and is trying to smelt a block of gold, then an expert may indicate that the agent should consider the actions of looking down and digging. Each active affordances contributes some set of actions to consider. During experiments, this method is denoted as `EARTDP`.

Additionally, actions may be pruned on a state by state basis by sampling actions from the probability distribution

over actions as specified by Equation 8. We treat each action's probability mass as a Bernouli trial and sample across the entire action set. In preliminary results, this method did not perform as well as baseline RTDP - likely because the weights associated with each action were too small. In future work, we are interested in investigating more sophisticated approaches to sampling as a means of action pruning **E: I'm a bit uncomfortable with us jumping the gun and talking about results and then future work here – I think this paragraph is good but we should shift the content towards the end to the correct sections.**.

In a recent review on the theory of affordances, **?**] suggests that an affordance is a relation between the features of an environment and an agent's abilities. We have chosen to ground this interpretation, where the features of the environment correspond to the goal-dependent state features, ϕ, and the agent's abilities correspond to the OO-MDP action set. In our model, there is an affordance for each δ_j, with preconditions $\delta_j.p$, goal type $\delta_j.g$ and action distribution $\Pr(\mathcal{A}^*|\phi_j, \theta)$, which is computed in our Naive Bayes model by marginalizing over all the features not associated with δ_j.

As with human agents, multiple affordances often inform decision making at a given time. Thus, affordance-aware planning agents operating within an OO-MDP will rarely make specific reference to particular affordances, but will instead reason about the world using the relevant action possibilities identified by the distribution in Equation 8.

D. Learning

E: I feel as though learning should go before action pruning since it specifies the action prior that is then used for pruning as detailed in Action Pruning.

To learn affordances, we input a set of training worlds (W) for which the optimal policy may be tractably computed. Then, we determine the parameter vector θ that maximizes the probability of the true computed optimal action set in each state of the world:

$$\underset{\theta}{\operatorname{argmax}} \sum_{w \in W} \sum_{s \in w} \Pr(\mathcal{A}^* \mid s, w.G, K) \tag{9}$$

where $w.G$ is the goal that was solved in world w.

Under our Bernouli Naive Bayes model, we need to estimate the parameters $\theta_i = \Pr(a_i)$ and $\theta_{i,j} = \Pr(\phi_j|a_i)$, for which the maximum likelihood estimates are:

$$\theta_i = \frac{C(a_i)}{C(a_i) + C(\bar{a}_i)} \tag{10}$$

$$\theta_{ji} = \frac{C(\phi_j, a_i)}{C(a_i)} \tag{11}$$

where $C(a_i)$ is the number of observed occurrences where a_i was optimal, $C(\bar{a}_i)$ is the number of observed occurrences where a_i was not optimal, and $C(\phi_j, a_i)$ is the number of occurrences where $\phi = 1$ and a_i was optimal.

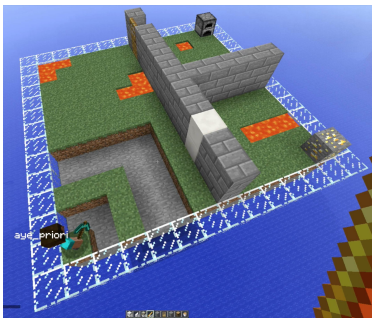

Fig. 1. A gold smelting task in the Minecraft domain

III. Results

We evaluate our approach using the game Minecraft, as well as on a collaborative cooking robot. Minecraft is a 3-D blocks game in which the user can place, craft, and destroy blocks of different types. Minecraft's physics and action space allow users to create complex systems, including logic gates and functional scientific graphing calculators[1]. Minecraft serves as a model for robotic tasks such as cooking assistance, assembling items in a factory, object retrieval, and complex terrain traversal. As in these tasks, the agent operates in a very large state-action space in an uncertain environment. Figure 1 shows a scene from one of our Minecraft problems **E: This doesn't seem fair to say since it this is much more difficult than our actual tests.**.

A. ARTDP in Minecraft

We conducted a series of experiments in the Minecraft domain that compared the performance of Real Time Dynamic Programming (RTDP) [**?**] without affordances to its affordance-aware counterpart (ARTDP). We created a set of expert affordances from our background knowledge of the domain, which are listed in Figure **??** **D: Expert ignored for now**. Additionally, we ran our full learning process and learned a single knowledge base of affordances for use in all tests. We compared RTDP with its expert-affordance-aware and learned-affordance-aware counterparts.

D: We still need to stick in a note about the VI results, and the fact that affordances can be plugged into other MDP/OO-MDP solvers

Our experiments consisted of 5 common tasks in Minecraft, including constructing bridges over trenches, smelting gold, tunneling through walls, basic path planning, and digging to find an object. We tested on randomized worlds of varying size and difficulty. The generated test worlds varied in size from tens of thousands of states to hundreds of thousands of states.

The training data consisted of 20 simple state spaces of each map type (100 total), each map approximately a 1,000-10,000 state world with randomized features that mirrored the agent's actual state space. We conducted all tests with a single knowledge base.

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

Our (A)RTDP and domain parameters are set as follows. We terminated each planner when the maximum change in the value function was less than 0.01 for 100 consecutive policy rollouts, or the planner failed to converge after 1500 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.

TABLE I
RTDP VS. LEARNED AFFORDANCE-AWARE RTDP

State Space	Planner	Bellman	Reward	Cpu
Mine	RTDP	17142.1 (±3843)	**-6.5** (±1)	**17.6s** (±4)
	LHRTDP	**12664.0** (±9340)	-12.7 (±5)	33.1s (±23)
	ERTDP	14357.4 (±3275)	**-6.5** (±1)	31.9s (±8)
Smelt	RTDP	30995.0 (±6730)	**-8.6** (±1)	45.1s (±14)
	LHRTDP	**2821.9** (±662)	-9.8 (±2)	**7.5s** (±2)
	ERTDP	28544.0 (±5909)	**-8.6** (±1)	72.6s (±19)
Wall	RTDP	45041.7 (±11816)	-56.0 (±51)	**68.7s** (±22)
	LHRTDP	**24020.8** (±9239)	**-15.8** (±5)	80.5s (±34)
	ERTDP	32552.0 (±10794)	-34.5 (±25)	96.5s (±39)
Trench	RTDP	16183.5 (±4509)	-8.1 (±2)	53.1s (±22)
	LHRTDP	11758.4 (±2815)	**-8.7** (±1)	57.9s (±20)
	ERTDP	**8674.8** (±2700)	-8.2 (±2)	**35.9s** (±15)
Plane	RTDP	52407.8 (±18432)	-82.6 (±42)	877.0s (±381)
	LHRTDP	**19089.9** (±9158)	**-7.5** (±1)	**246.4s** (±159)
	ERTDP	32928.0 (±14997)	-44.9 (±34)	505.3s (±304)

The evaluation metric for each trial was the number of Bellman updates that were executed by each planning algorithm, the accumulated reward **E: of the average plan right?**, and the CPU time taken to find a plan. Table I shows the average CPU time, accumulated reward, and Bellman updates for RTDP and affordance-aware RTDP (ARTDP) after planning in 20 different maps of each goal type (100 total). Because the planners were forced to terminate after only 1000 rollouts, they did not always converge to the optimal policy. In every task, affordance-aware-RTDP outperformed or performed comparably to RTDP. As the results suggest, ARTDP found a better plan (-11.9 reward to -231.4 reward) in significantly less time than RTDP in the gold smelting task (6 seconds to 543 seconds), as well as the plane traversal task (17.5 seconds to 168.4 seconds).

D: Put in a note about the threshold issues, maybe include the results below to reinforce threshold business

B. ARTDP and Temporally Extended Actions in Minecraft

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 affordances, macroactions, and options. We conducted these experiments with the same configurations as our Minecraft experiments. The option policies and macroactions provided were hand coded by domain experts.

TABLE II
AFFORDANCES VS. TEMPORALLY EXTENDED ACTIONS

Planner	Augmentation	Bellman	Reward	CPU
RTDP	Vanilla	27439(±2348)	-22.6(±9)	107(±33)
	w/ Opt	26663(±2298)	-17.4(±4.0)	129(±35)
	w/ MA	31083(±2468)	-21.7 (±5.0)	336(±28)
	w/ MA+Opt	27143(±2380)	-16.9(±3.0)	323(±38)
Affordance-Aware	Vanilla	9935(±1031)	-12.4(±1.0)	53(±5.0)
	w/ Opt	9675(±953)	-11.5(±1.0)	93(±10)
	w/ MA	9854(±1034)	-11.7(±1.0)	162(±17)
	w/ MA+Opt	10622(±1046)	-13.4(±1.0)	237(±29)

Table II indicates the results of comparing RTDP equipped with macro actions, options, and affordances across 100 different executions in the same randomly generated Minecraft worlds. The results above are averaged across tasks of each type presented in Table I. As the results suggest, both macroactions and options add a significant amount of time to planning. This is due to the fact that it is computationally expensive to predict the expected reward associated with applying an option or a macro-actions. Furthermore, the branching factor of the state-action space significantly increases when augmented with additional actions, causing the planner to run for longer and perform more Bellman updates. With affordances, the planner found a better plan in less CPU time, and with fewer Bellman updates. This supports the claim that affordances can handle the augmented action space provided by temporally extended actions by pruning away unnecessary actions.

C. Learning Rate in Minecraft

Additionally, we conducted experiments in which we varied the number of states visited at training time. As in Table I, we randomly generated simple state spaces containing several thousand states containing features that mirrored the agent's state space at test time. We then solved the OO-MDP with knowledge bases learned from 10 to 10000 states.**D: update when training complete**

D: Experiments have not yet finished for learning rate (underway)

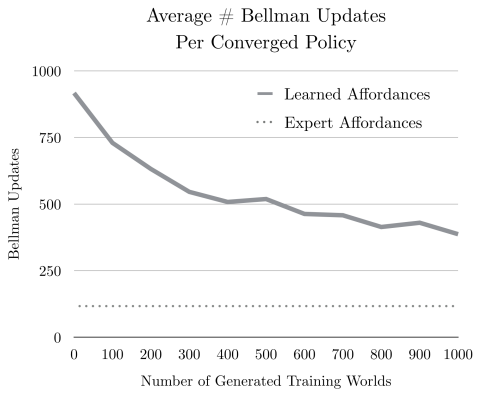

Fig. 2. PLACE HOLDER for learning rate results

D. ARTDP and Baxter

D: Insert description of baxter stuff here

D: Experiments not yet finished

Fig. 3. Placeholder for baxter results/image

IV. 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 designed for use with deterministic domains.

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].

Since temporally extended actions may negatively impact planning time [13] by adding to the number of actions the agent can choose from in a given state, combining affordances with temporally extended actions allow for even further speedups in planning, as demonstrated in II. This suggests that affordances are complementary to macro actions and options.

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 goal types**E: This is the first time we are now talking about LGD?**, 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 can handle a wide variety of tasks in a single knowledge base, as demonstrated by Table I

D: Policy priors addition?

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 may be learned easily for 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 since they deal with non-stochastic problems, but there are a number of salient parallels and contrasts to be drawn nonetheless.

1) 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**

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.

However there are two essential distinctions between affordances and traditional HTNs. (1) HTNs do not incorporate

reward into their planning. Consequently, they lack mathematical guarantees of optimal planning. (2) 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. **E: Need citations for HTNs**

2) 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, a 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.

E: Maybe add how LTL is similar to what we are doing?

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

V. 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 actions based on learned 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 a RTDP to its affordance-aware equivalent 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.

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 robot in a collaborative cooking 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 affordances to plug into high-level subgoal planning, which will reduce the size of the explored state-action space and improve the relevance of the action pruning.

Additionally, we hope to explore additional methods that capitalize on the distribution over optimal actions, such as incorporating affordances with a forward search sparse sampling algorithm [**?**], or replacing the Naive Bayes model with a more sophisticated model, such as Logistic Regression or a Noisy-OR. We are also investigating methods of learning the thresholded value in a more principled way - one such approach is to initialize the planner with a strict threshold, and slowly relax the threshold until a near optimal policy is found. We are also interested in updating model parameters on-line by using planning data to update the distribution over optimal actions.

Lastly, we hope to decrease the amount of knowledge given to the planner by learning predicates, or relations between predicates (i.e. Incremental Feature Dependency Discovery [9]) to further enhance autonomy.

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.