Affordance-Aware Planning

David Abel & Gabriel Barth-Maron, James MacGlashan, Stefanie Tellex
Computer Science Department, 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 reduce the size of the state/action space without sacrificing optimality, 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 experiencex, 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 [4, 14]. Robotic planning tasks 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 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: when a robot is manipulating objects in an environment, an object can be placed anywhere in a large set of locations. Depending on the task, most of these locations are irrelevant; for example, 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. Unfortunately, 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]. 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 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 satisfy a given goal. Affordances 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, 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).

A. OO-MDPs

An Object-Oriented Markov Decision Process (OO-MDP) [8] efficiently represents the state of an MDP, organized around objects and predicates. We use OO-MDP predicates as features for action pruning, allowing for state space independence when predicates generalize across state spaces. 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, $o.state$.

B. Affordances

To perform action pruning, we model the distribution over the optimal OO-MDP action set. We define the optimal action set, $\mathcal{A}^*$, for a given state s and goal G as:

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

Here, $Q^*_G(s,a)$ and $V^*_G(s)$ represent the optimal Q function and value function, respectively.

We induce a distribution over the optimal action set, in a particular state s, under a goal G, and given a knowledge base K.

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

The agent uses this distribution to estimate the optimal action set for each state-goal pair, based on the set of available affordances. To formalize a knowledge base of affordances, we represent K as a set of paired state preconditions and goal types, which induce a distribution over the resulting optimal action set. Formally, K consists of a set of paired preconditions and goal types, $\delta_1 \ldots \delta_{|K|}$, and a parameter vector θ. Each δ_j is a precondition and goal type pair, $\langle p, g \rangle$. The parameter vector, θ_{ij} represents the weight associated with action i, for δ_j. Here, $p \in \mathcal{P}$ is a *predicate* in predicate space, $\mathcal{P}$, and $g \in \mathcal{G}$ is a *goal type* in goal space, $\mathcal{G}$. We rewrite Equation 2 replacing K with its constituents:

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

To represent the state, goal, set of objects $\delta_1 \ldots \delta_{|K|}$ as a set of features, we introduce the function $f : \Delta \times \mathcal{S} \times \mathcal{G} \mapsto \{0, 1\}$. Δ is the space of all δ objects, and $\mathcal{S}$ is the OO-MDP action space. f is computed as follows:

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

Using f, we consolidate the state, goal type, and $\delta_1 \ldots \delta_{|K|}$ into a set of binary goal dependent state features, $\phi_1 \ldots \phi_{|K|}$, where $\phi_j = f(\delta_j, s, G)$:

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

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

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

We abbreviate $a_i \in \mathcal{A}^*$ to a_i. This distribution may be modeled in any number of ways, making this approach quite flexible. We have chosen to model it as a Naive-Bayes, assuming a uniform prior on the optimality of each action, and assuming that the features are conditionally independent of one another. First we factor using Bayes' rule:

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

Next we assume that each feature is independent of the others, given the true class label:

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

We approximate the optimal action set by computing the maximum likelihood estimate of θ from training data and sampling, by counting the number of times an action occurs in an optimal policy when the associated precondition and postcondition are satisfied, described in the following section.

A specific affordance refers to the action distribution associated with a single feature being active, while marginalizing out the rest of the features:

$$= \prod_{i=1}^{|\mathcal{A}|} \frac{\sum_{m \neq j}^{|K|} \Pr(\phi_j, \theta_{ij} \mid a_i, \phi_m) \Pr(\phi_m)}{\Pr(\phi_1, \ldots, \phi_{|K|}, \theta_i)} \tag{9}$$

ST: Can we say something about expert-provided hard affordances here? Basically, that you could specify hard affordances according to equation 9, but in practice, many affordances are relevant... and then the rest of this paragraph As with human agents, many affordances are often relevant in 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. For instance, suppose an affordance-aware Minecraft agent was attempting to cross a trench to reach a goal on the other side. The 'jump' affordance will likely activate, indicating the agent may jump over the trench, or perhaps the 'bridge' affordance will activate if it is a large enough trench, suggesting that the trench may be traversed by constructing a bridge. These affordances are important to consider, but only inform part of the agent's decision making. As with a human agent, the full set of relevant affordances defines the actions the agent ought to consider in any state of the world. **ST: Shouldn't we refer back to the optimal action set?**

C. Learning

To learn affordances, we estimate the parameter vector θ that maximizes the accumulated reward during training, given a set of training worlds, W:

$$\underset{\theta}{\operatorname{argmax}} \sum_{w \in W} \sum_{s \in w} \Pr(\mathcal{A}^* \mid s, w.G, K) \tag{10}$$

We rewrite using Equation 9:

$$\underset{\theta}{\operatorname{argmax}} \propto \sum_{w \in W} \sum_{s \in w} \prod_{i=1}^{|\mathcal{A}|} \prod_{j=1}^{|K|} \Pr(\theta_{ji}, \phi_j \mid a_i) \tag{11}$$

As with a Bernouli naive Bayes, we express $\Pr(\theta_{ji}, \phi_j \mid a_i)$ as follows:

$$\Pr(\theta_{ji}, \phi_j \mid a_i) = \phi_j \cdot \Pr(\theta_{ij} \mid a_i) + (1 - \phi_j) \cdot (1 - \Pr(\theta_{ij} \mid a_i)) \quad (12)$$

We compute $\Pr(\theta_{ij} \mid a_i)$ as the Maximum Likelihood Estimate:

$$\theta_{ji} = \frac{C(\phi_j \mid a_i)}{\sum_{j=1}^{|K|} C(\phi_j \mid a_i)} \quad (13)$$

Where $C(\phi_j \mid a_i)$ denotes the counts of the number of times feature ϕ_j was active in states where a_i was optimal during training. To inform these counts, we randomly generate a set of training worlds, W, that are small enough to synthesize a policy using tabular methods (i.e. thousands of states). We generated W so that each of the possible goal types was assigned to a subset of the generated worlds. **ST: Do we run only on worlds small enough to use a tabular method, or do we let it do bootstrapping?**

D: —-Edited up to here——

III. Experiments

We use Minecraft as our training and evaluation domain. 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 are expressive enough to 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 **??** shows a scene from one of our Minecraft problems. **ST: Add minecraft figure.**

For experiments, we introduce a simplified baseline in which the an expert specified threshold determines the action set computed from Equation **??**. This baseline is listed as *LT* (for learned threshold).

A. Minecraft Tests

We conducted a series of experiments in the Minecraft domain that compared the performance of several planner without affordances to their affordance-aware counterparts. We created a set of expert affordances from our background knowledge of the domain, which are listed in Figure **??**. Additionally, we ran our full learning process and learned affordances for each task. We compared Real Time Dynamic Programming (RTDP) with its expert-affordance-aware and learned-affordance-aware counterparts. **D: Add note about comparison to VI like Stefanie mentioned**

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 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 10 simple state spaces of each map type (50 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.**D: We'll need to update all this once we actually count state space sizes**

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. RTDP was 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.

B. Learning Rate

Additionally, we conducted experiments in which we varied the number of states visited at training time. As in Table **??**, 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**

C. Temporally Extended Actions

Furthermore, 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 macro actions provided were hand coded by domain experts.

D: Do we need a note about how many actions/macroactions/options given?

D. Robotic Task

Finally, we deployed an affordance-aware planner onto Baxter for use in an assistive cooking task. **D: ToDo**

IV. Results

A. Minecraft: Expert vs Learned vs None

D: Should we switch tables to bar charts so we can show error bars?

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

Planner	CPU	Reward	Bellman
RTDP	-	-	-
ERTDP	-	-	-
LRTDP	-	-	-
LTRTDP	-	-	-

TABLE I

D: THOUGHTS ON THIS STYLE TABLE INSTEAD? THEN WE WOULD JUST REPORT AVERAGES? OR A BAR CHART? ONLY DOWNSIDE IS THAT WE DON'T SPECIFY THE STATE SPACE SIZE

B. Minecraft: Learning rate

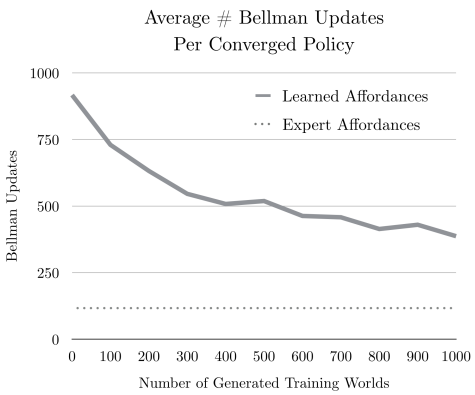

Fig. 1. Place holder for learning rate results

C. Temporally Extended Actions

D: These results are no longer accurate (from the previous iteration of math/implementation

Augmentation	CPU	Reward	Bellman
RTDP	4.86s	-4.66	4792.0
w/ MA	21.73s	-4.66	5255.0
w/ Options	6.43s	-4.66	3890.33
w/ Options + MA	26.10s	-4.66	5310.0
w/ Affordances	3.48s	-4.66	1907.0
w/ Affordances+MA	7.54s	-4.66	2088.0
w/ Affordances+Options	**3.32s**	-4.66	**1829.66**
w/ Affordances+MA+Options	-8.45s	-4.66	2053.66

TABLE II
AFFORDANCES VS. TEMPORALLY EXTENDED ACTIONS

D. Baxter

Fig. 2. Placeholder for baxter results/image

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**.

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**

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: needs to be shorter**

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.

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 actions 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 planners 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.

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 the relevance of the action pruning. Additionally, we hope to decrease the amount of knowledge given to the planner by implementing lowering the expert seed requirements for learning affordances. One plan is to only provide a base of primitive predicates, and to

implement Incremental Feature Dependency Discovery [9], allowing 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.