Affordance-Aware Planning

Paper-ID [add your ID here]

***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 address this problem fail to prevent the planner from considering many actions which would be obviously irrelevant to a human solving the same problem. We formalize the notion of affordances [7] as knowledge added to an MDP that prunes actions in a state- and reward- general way. This pruning significantly reduces the number of state-action pairs the agent needs to evaluate in order to act optimally. We demonstrate our approach in the Minecraft domain, showing significant increase in speed and reduction in state-space exploration compared to the standard versions of these algorithms. Further, we provide a learning framework that enables an agent to learn affordances through experience, removing the agent's dependence on the expert. We provide preliminary results indicating that the learning process effectively produces affordances that help solve an MDP faster.**

I. Introduction

As robots move out of the lab and into the real world, planning algorithms need to scale to domains of increased noise, size, and complexity. A classic formalization of this problem is a stochastic sequential decision making problem in which the agent must find a policy (a mapping from states to actions) for some subset of the state space that enables the agent to achieve a goal from some initial state, while minimizing any costs along the way. Increases in planning problem size and complexity directly correspond to an explosion in the state-action space. Current approaches to solving sequential decision making problems in the face of uncertainty cannot tackle these problems as the state-action space becomes too large [8].

To address this state-space explosion, prior work has explored adding knowledge to the planner to solve problems in these massive domains, such as options [20] and macroactions [4, 15]. However, these approaches add knowledge in the form of additional high-level actions to the agent, which *increases* the size of the state-action space (while also allowing the agent to search more deeply within the space). The resulting augmented space is even larger, which can have the paradoxical effect of increasing the search time for a good policy. Further, other approaches fall short of learning useful, transferable knowledge, either due to complexity or lack of generalizability.

Instead, we propose a formalization of *affordances* [7] that enables an agent to focus on aspects of the environment that are most relevant toward solving its current goal. Our approach avoids exploration of irrelevant parts of the state-action space, which leads to dramatic speedups in planning.

We formalize the notion of an affordance as a piece of planning knowledge provided to an agent operating in a Markov Decision Process (MDP). Further, we propose a learning process that enables agents to autonomously learn affordances through experience, lessening the agent's dependence on expert knowledge. Affordances are not specific to a particular reward function or state space, and thus, provide the agent with transferable knowledge that is effective in a wide variety of problems.

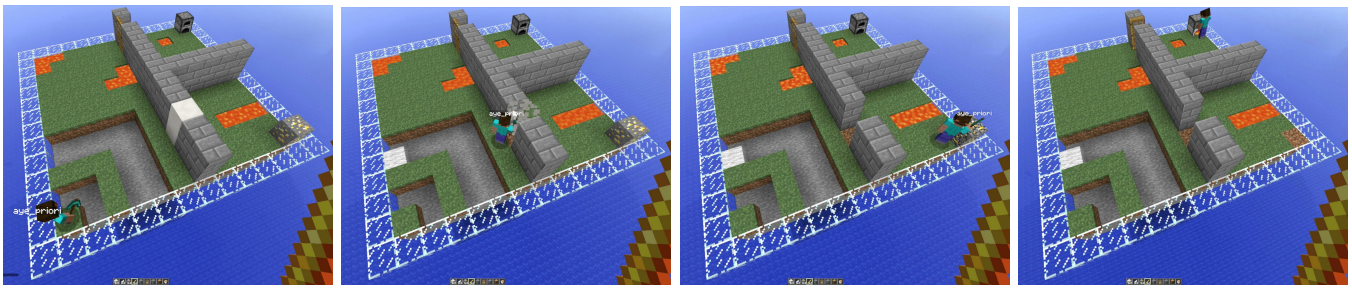
We capture the intuition that affordances are an inherent property of an environment containing a goal-oriented agent by focusing the agent on specific actions when trying to satisfy a particular goal. The affordances that are on hand to the agent will be determined by what the agent is trying to accomplish at any given time, thus determining which actions are most pertinent. Because affordances define the *kind* of goals for which actions are useful, affordances also enable high-level reasoning that can be combined with approaches like high-level subgoal planning for even greater performance gains. For now, we have chosen to ignore the direct perception of affordances in the environment, and have instead opted to emphasize an affordances ability to direct a planning agent through large stochastic state spaces. We foresee future methods that incorporate vision and perception solutions with affordance-aware planners to create robust, near-autonomous robotic agents.

II. Background

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. Minecraft's physics and action space is expressive enough to allow very complex worlds to be created by users, such as a functional scientific graphing calculator[1]; simple scenes from a Minecraft world appear in Figure 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.

Minecraft serves as an effective parallel for the actual world, both in terms of approximating the complexity and scope of planning problems, as well as modeling the uncertainty and noise presented to a real world agent. For instance, robotic agents are prone to uncertainty all throughout their system, including noise in their sensors (cameras, LIDAR, microphones, etc.), odometry, control, and actuation. In order to accurately capture some of the inherent difficulties of planning under uncertainty, the Minecraft agent's actions were modified to have stochastic outcomes. These stochastic outcomes may require important changes in the optimal policy in contrast to deterministic actions, such as keeping the agent's distance from a pit of lava. We chose to give the Minecraft agent perfect

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



(a) Start (b) Destroy Wall (c) Collect Ore (d) Smelt Ore

Fig. 1. Affordance-aware RTDP tasked with a gold-smelting task with a variety of obstacles. This planning task was only solved by an affordance-aware planner.

sensor data about the Minecraft world, as that is presently beyond the focus of this work.

A. OO-MDPs

We define affordances in terms of propositional functions on states. Our definition builds on the Object-Oriented Markov Decision Process (OO-MDP) [6]. OO-MDPs are an extension of the classic Markov Decision Process (MDP). 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 more greatly prefers to maximize more 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. By contrast, an OO-MDP represents the state space as 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. Upon instantiation of an object class, its attributes are given a state $o.state$ (an assignment of values to its attributes). The underlying MDP state is the set of all the object states: $s \in \mathcal{S} = \cup_{i=1}^{o} \{o_i.state\}$.

There are two advantages to using an object-oriented factored state representation instead of a single feature vector. First, different states in the same state space may contain different numbers of objects of varying classes, which is useful in domains like Minecraft in which the agent can dynamically add and remove blocks to the world. Second, MDP states can be defined invariantly to the specific object references. For instance, consider a Minecraft world with two block objects, b_1 and b_2. If the agent picked up and swapped the position of b_1 and b_2, the MDP state before the swap and after the swap would be the same, because the MDP state definition is invariant to which object holds which object state.

While the OO-MDP state definition is a good fit for the Minecraft domain, our motivation for using an OO-MDP lies in the ability to formulate 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. In the original OO-MDP work, these predicates were used to model and learn an MDP's transition dynamics.

While an OO-MDP reduces the size of the Minecraft state space by a significant factor, the resulting state space is still far too large to solve with any existing (OO)-MDP solver. This is the primary motivator for incorporating affordances as a means of reducing the amount of the state space that an OO-MDP agent will have to explore.

III. Affordances

We define an affordance Δ as the mapping $\langle p, g \rangle \longmapsto \mathcal{A}'$, where:

> $\mathcal{A}' \subseteq \mathcal{A}$, a subset of the action space, representing the relevant *action-possibilities* of the environment.
> p is a predicate on states, $s \longrightarrow \{0, 1\}$ representing the *precondition* for the affordance.
> g is an ungrounded predicate on states, g, representing a *lifted goal description.*

The precondition and goal description predicates refer to predicates that are defined in the OO-MDP definition. Using OO-MDP predicates for affordance preconditions and goal descriptions allows for state space independence. Thus, a planner equipped with affordances can be used in any number of different tasks.

A. Affordance Aware Planning

We call any planner that uses affordances an *affordance-aware* planner. For a given state, our goal is to solve for the probability of getting a particular action set $\mathcal{A}^*$, and approximate sampling from this distribution. This ensures that in the limit, it is possible to apply each action in each state.

$$\Pr(\mathcal{A}^* \mid s, \Delta_0 \ldots \Delta_N) \tag{1}$$

We let each affordance contribute a set $\mathcal{A}'$ in each state:

$$\Pr(\mathcal{A}'_0 \cup \mathcal{A}'_N \mid s, \Delta_0 \ldots \Delta_N) \tag{2}$$

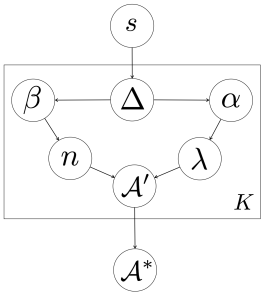


Fig. 2. The full graphical model approximating a distribution over $\mathcal{A}^*$, the pruned action set for a given state s

We approximate this term assuming the sets $\mathcal{A}'_i$ are disjoint:

$$\sum_{i}^{K} \Pr(\mathcal{A}'_i \mid s, \Delta_i) \tag{3}$$

Given a set of K domain affordances $Z = \{\Delta_1, ..., \Delta_K\}$ and a current agent goal condition defined with an OO-MDP predicate G, the action set that a planning algorithm considers is pruned on a state by state basis as shown in Algorithm 1. For instance, the affordances defined for Minecraft navigation problems can be used in any task regardless of the spatial size of the world, number of blocks in the world, and specific goal location that needs to be reached. Each activated affordance contributes a suggested action set, determined by Algorithm 2. We call an affordance *activated* when its predicate is true and the lifted goal description g matches the agent's current goal.

Algorithm 1 getActionsForState($state$, Z, G)

1: $\mathcal{A}^* \leftarrow \{\}$
2: **for** $\Delta \in Z$ **do**
3: **if** $\Delta.p(state)$ and $\Delta.g = G$ **then**
4: $\mathcal{A}^* \leftarrow \mathcal{A}^* \cup \Delta.getActions(s)$
5: **end if**
6: **end for**
7: **return** $\mathcal{A}^*$

Specifically, we prune actions on a state by state basis by initializing an empty set of actions $\mathcal{A}^*$ (line 1). The algorithm then iterates through each of the domain affordances (lines 2-6). If the affordance precondition ($\Delta.p$) is satisfied by some set of objects in the current state and the affordance goal condition ($\Delta.g$) is defined with the same predicate as the current goal (line 3), then the actions associated with the affordance ($\Delta.\mathcal{A}' = \Delta.getActions(s)$) are added to the action set $\mathcal{A}^*$ (line 4). Finally, $\mathcal{A}^*$ is returned (line 7).

Thus, for each affordance, we get an action set $\mathcal{A}'$. This process is outlined by Algorithm 2. We form a Dirichlet Multinomial over actions (λ), and a Dirichlet over the size (N) of each action set:

$$\Pr(\mathcal{A}_i \mid s, \Delta_i) = \Pr(\mathcal{A}'_i \mid N, \lambda) = \Pr(\lambda \mid \alpha) \cdot \Pr(N \mid \beta) \tag{4}$$

For a given affordance Δ_i, we sample from our distribution over action set size to get a candidate action set size (lines 1-2). We then take that many samples from our distribution over actions to get a candidate action set $\mathcal{A}'$ (lines 3-5).

$$\Pr(\lambda \mid \alpha) = DirMult(\alpha) \tag{5}$$

$$\Pr(N \mid \beta) = Dir(\beta) \tag{6}$$

Algorithm 2 $\Delta_i.getActions(s)$

1: $\lambda \leftarrow DirMult(\Delta_i.\alpha)$
2: $N \leftarrow Dir(\Delta_i.\beta)$
3: **for** 1 to N **do**
4: $\Delta_i.\mathcal{A}' \leftarrow \lambda$
5: **end for**
6: **return** $\Delta_i.\mathcal{A}'$

Through the use of Algorithms 1 & 2, any OO-MDP solver made be made *affordance aware*. For a planner to be made affordance-aware, we require that an expert provide a set $\mathcal{P}$ of predicates for the domain of relevance (i.e. Minecraft). Additionally, the expert must specify a set $\mathcal{G} \subset \mathcal{P}$, that indicates which predicates may serve as goal conditions. If the expert wishes to provide the affordances directly, they must specify the Dirichlet parameters α and β. Note that in the limit, the expert may fix α and β in a way that forces a given affordance to always suggest a specific set of actions.

The real strength of our affordance formalism is that it is simple enough to learn useful affordances directly.

B. Learning Affordances

To learn affordances, we propose a scaffolded learning process that computes α and β by solving OO-MDPs in small state-spaces that present similar sorts of features to the state spaces the agent might expect to see.

Given the set of predicates $\mathcal{P}$ and possible goals $\mathcal{G} \subset \mathcal{P}$, we form a set of candidate affordances Δ with every combination of $\langle p, g \rangle$, for $p \in \mathcal{P}$ and $g \in \mathcal{G}$.

Then, we randomly generate a large number of state spaces which are small enough to be solved using tabular methods (typically on the order of several hundred to a few thousand), annotated with their lifted goal description $g \in \mathcal{G}$. We solve the OO-MDP in each state space to find an optimal policy π_j.

For each optimal policy, we count the number of policies that used each action when each affordance was activated.[2] These counts represent α. Then, we count the number of unique actions used by each policy, representing β.

IV. Experiments

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 selected the expert affordances from our background knowledge of the domain. Additionally, we tested our learning

[2]Recall we call an affordance *activated* when its predicate is true and the lifted goal description g matches the agent's current goal.

procedure and compared the performance of RTDP solving the OO-MDP with (1) No affordances, (2) Learned affordances, and (3) Expert provided affordances. We generated between 10-5000 simple state spaces to learn on, each a $3 \times 3 \times 3$ world with randomized features based on the features of the agent's actual state space.

We gave the agent a single knowledge base of 5 types of affordances, which are listed in Figure 3. Our experiments consisted of a variety of common tasks in Minecraft, ranging from basic path planning, to smelting gold, to opening doors and tunneling through walls. We also tested each planner on worlds of varying size and difficulty to demonstrate the scalability and flexibility of the affordance formalism. The evaluation metric for each trial was the number of state backups that were executed by each planning algorithm. 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 five consecutive policy rollouts. In subgoal planning, the high-level subgoal plan was solved using breadth-first search; which only took a small fraction of the time compared to the total low-level planning and therefore is not reported. We used RTDP as the low-level planner for subgoal planning.

We set the reward function to -1 for all transitions, except transitions to states in which the agent was on lava, which returned -200. The goal was set to be terminal. The discount factor was set to $\lambda = 0.99$. For all experiments, the agent was given stochastic actions. Specifically, actions associated with a direction (e.g. movement, block placement, jumping, etc.), had a small probability (0.3) of moving in another random direction.

V. Results

Table I shows the number of Bellman updates required when solving the OO-MDP with conventional methods (left column) compared to solving the OO-MDP with an Affordance-Aware method (right column). The affordance aware methods significantly outperformed their unaugmented counterparts in all of these experiments. These results, while unsurprising, concretely demonstrate that a small set of affordances prune away many useless actions across many different types of Minecraft tasks.

Table II indicates the average number of Bellman updates required by RTDP to solve the OO-MDP in each of the four candidate worlds. The learned affordances clearly improved on standard RTDP by a significant margin, though there is still a substantial gap between the learned affordance performance and that of the expert affordances. This indicates that there is still room for improvement in the learning process.

$$\Delta_1 = \langle nearTrench, reachGoal \rangle \longmapsto \{place, jump\}$$
$$\Delta_2 = \langle onPlane, reachGoal \rangle \longmapsto \{move\}$$
$$\Delta_3 = \langle nearWall, reachGoal \rangle \longmapsto \{destroy\}$$
$$\Delta_4 = \langle nearFurnace, makeGold \rangle \longmapsto \{place\}$$
$$\Delta_5 = \langle nearOre, makeGold \rangle \longmapsto \{destroy\}$$

Fig. 3. The five affordance types used in expert experiments.

TABLE I
Expert Affordance Results: Avg. Number of Bellman Updates per Converged Policy

	VI	A-VI	RTDP	A-RTDP	SG	A-SG
4TRENCH	71604	**100**	836	**152**	1373	**141**
6TRENCH	413559	**366**	4561	**392**	28185	**547**
8TRENCH	1439883	**904**	18833	**788**	15583	**1001**
DOOR	861084	**4368**	12207	**1945**	6368	**1381**
LAVA	413559	**366**	4425	**993**	25792	**597**
TUNNEL	203796	**105**	26624	**145**	5404	**182**
GOLD	16406	**962**	7738	**809**	7412	**578**

TABLE II
Learned Affordance Results: Avg. Number of Bellman Updates per Converged Policy

	No Affordances	Learned	Expert
Tiny World	879	414	94
Small World	1460	802	321
Medium World	3993	2412	693
Large World	8344	5100	1458

Figure 4 demonstrates the added effect of more training data. The chart indicates that the more time invested into learning affordances for a knowledge base, the faster an OO-MDP may be solved. In these experiments, we tested on map types that mimicked the features of the worlds generated during training, but this process could be extended to allow for scaffolded learning and more complicated maps (such as the gold smelting task in Figure 1). The averages reported are from solving the OO-MDP 20 times for each world, with each knowledge base. There was a negligible difference in the quality of the policies generated.

VI. Related Work

In this section, we discuss the differences between affordance-aware planning and other forms of knowledge that have been used to accelerate planning.

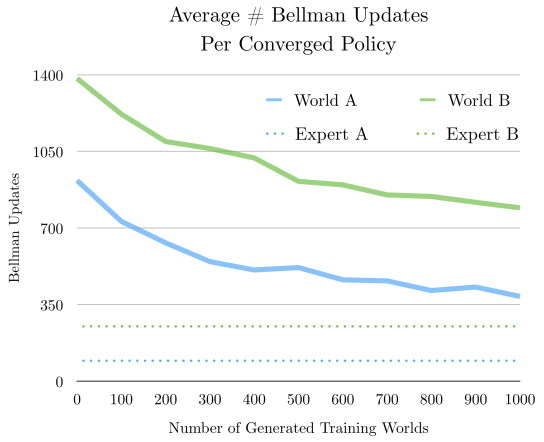


Fig. 4. The effectiveness of added training data on solving an OO-MDP.

A. Temporarily 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* [10] and *options* [20]. 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 [14, 12, 17, 5, 1, 13].

Given the potential for unhelpful temporally extended actions to negatively impact planning time [11], 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. In the future, we plan to explore the benefit from combining these approaches.

B. Action Pruning

Sherstov and Stone [19] 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 (or may be inferred).

Rosman and Ramamoorthy [18] 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.

C. Temporal Logic

Bacchus and Ady [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 - often achieving polynomial time planning in exponential space. 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.

While this approach is theoretically sound and has achieved empirical success, its main drawback is the complexity of its knowledge. The primary difference between this body of work and our own is that the affordance formalism may be learned, while LTL formulas are far too complicated to learn effectively.

D. Heuristics

Heuristics in MDPs are used to convey information about the value of a given state or 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 [9].

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 [16].

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 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.

VII. CONCLUSION

We proposed a novel approach to representing knowledge in terms of *affordances* [7] that allows an agent to efficiently prune its action space based on domain knowledge. This led to the proposal of affordance-aware planners, which improve on classic planners by providing a significant reduction in the number of state-action pairs the agent needs to evaluate in order to act optimally. We demonstrated the efficacy as well as the portability of the affordance model by comparing standard paradigm 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. We provided preliminary results indicating the effectiveness of the learned affordances, suggesting that the agent may be able to learn to tackle new types of problems on its own.

In the future, we hope to increase our coverage of Minecraft to solve even more difficult planning problems, as well as extending this approach beyond the Minecraft domain and onto actual robots, and other games (i.e. Atari). We also hope to learn subgoals from text for use in forming a high level plan in conjunction with our affordance formalism.

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] 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.

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

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

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

[8] Matthew Grounds and Daniel Kudenko. Combining reinforcement learning with symbolic planning. In *Proceedings of the 5th, 6th and 7th European conference on Adaptive and learning agents and multi-agent systems: adaptation and multi-agent learning*, ALAS '05, 2005.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.