Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

Goal-Based Action Priors

David Abel, D. Ellis Hershkowitz, Gabriel Barth-Maron,
Stephen Brawner, Kevin O'Farrell, James MacGlashan, Stefanie Tellex
Brown University, Computer Science Department
115 Waterman Street, 4th floor Providence, RI 02912

Abstract

Robots that interact with people must flexibly respond to requests by planning in stochastic state spaces that are often too large to solve for optimal behavior. In this work, we develop a framework for goal and state dependent action priors that can be used to prune away irrelevant actions based on the robot's current goal, thereby greatly accelerating planning in a variety of complex stochastic environments. Our framework allows these goal-based action priors to be specified by an expert or to be learned from prior experience in related problems. We evaluate our approach in the video game Minecraft, whose complexity makes it an effective robot simulator. We also evaluate our approach in a robot cooking domain that is executed on a two-handed manipulator robot. In both cases, goal-based action priors enhance baseline planners by dramatically reducing the time taken to find a near-optimal plan.

1 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 the very large set of possible tasks (Bollini et al. 2012; Knepper et al. 2013). A powerful and flexible robot such as a mobile manipulator in the home has a very large set of possible actions, any of which may be relevant depending on the current goal (for example, robots assembling furniture (Knepper et al. 2013) or baking cookies (Bollini et al. 2012).) 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 explodes exponentially with the number of objects, which bounds the placement problems that the robot is able to expediently solve. Depending on the goal, any of these states and actions may be relevant to the solution, but for any specific goal most of them are irrelevant. For instance, when making brownies, actions related to the oven and flour are important, while those involving the soy sauce and sauté pan are not. For a different task, such as stir-frying broccoli, the robot must use a different set of objects and actions.

Robotic planning tasks are often formalized as a stochastic sequential decision making problem, modeled as a Markov Decision Process (MDP) (Thrun, Burgard, and Fox 2008). In these problems, the agent must find a mapping from states to actions for some subset of the state space that enables the agent to achieve a goal while minimizing costs along the way. However, many robotics problems correspond to a family of related MDPs; following STRIPs terminology, these problems come from the same *domain*, but each problem may have a different reward function or goal. For example, Figure 1 shows an example of two problems with the same goal (smelting gold) and domain (the game Minecraft (Mojang 2014)).

Figure 1: Two problems from the same domain, where the agent's goal is to smelt the gold in the furnace while avoiding the lava. Our agent is unable to solve the problem on the right before learning because the state/action space is too large (since it can place the gold block anywhere). After learning on simple problems like the one on the left, it can quickly solve the larger problem.

To confront the state-action space explosion that accompanies complex domains, prior work has explored adding knowledge to the planner, such as options (Sutton, Precup, and Singh 1999) and macro-actions (Botea et al. 2005; Newton, Levine, and Fox 2005). However, while these methods allow the agent to search more deeply in the state space, they add non-primitive actions to the planner which *increase* the branching factor 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 (Jong 2008). Deterministic forward-search algorithms like hierarchical task networks (HTNs) (Nau et al. 1999), and temporal logical planning (TLPlan) (Bacchus and Kabanza 1995; 1999), 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.

To address state-action space explosions in robotic planning tasks, we augment an Object Oriented Markov Decision Process (OO-MDP) with a specific type of action prior conditioned on the current state and an abstract goal description. This *goal-based action prior* enables the robot to prune irrelevant actions on a state-by-state basis according to the robot's current goal, focusing the robot on the most promising parts of the state space.

Goal-based action priors can be specified by hand or learned through experience in related problems, making them a concise, transferable, and learnable means of representing useful planning knowledge. Our results demonstrate that these priors provide dramatic improvements for a variety of planning tasks compared to baselines in simulation, and are applicable across different state spaces. Moreover, while manually provided priors outperform baselines on difficult problems, our approach is able to learn goal-based action priors from experience on simple, tractable, training problems that yield even greater performance on the difficult problems than manually provided priors.

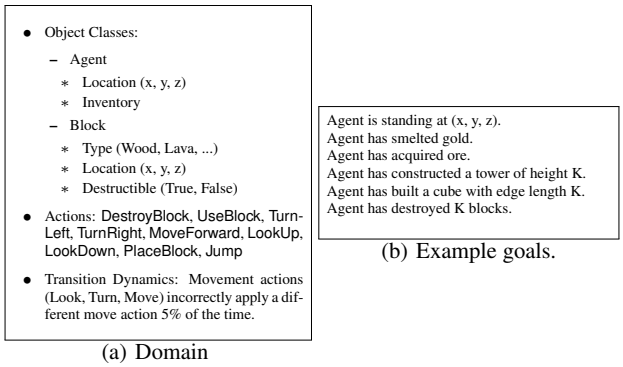
We conduct experiments in the game Minecraft, which has a very large state-action space, and on a real-world robotic cooking assistant. Figure 1 shows an example of two problems from the same domain in the game Minecraft; the agent learns on simple randomly generated problems (like the problem in the left image) and tests on new harder problems from the same domain that it has never previously encountered (like the problem in the right image). All associated code with this paper may be found at http://h2r.cs.brown.edu/affordances.

Because we condition on both the state and goal description, there is a strong connection between our priors and the notion of an *affordance*. Affordances were originally proposed by (Gibson 1977) as action possibilities prescribed by an agent's capabilities in an environment, and have recently received a lot of attention in robotics research (Koppula and Saxena 2013; Koppula, Gupta, and Saxena 2013).

In a recent review on the theory of affordances, (Chemero 2003) suggests that an affordance is a relation between the features of an environment and an agent's abilities. Our goal-based action priors are analogously interpreted as a grounding of Chemero's interpretation of an affordance, where the features of the environment correspond to the goal-dependent state features, and the agent's abilities correspond to the OO-MDP action set. It is worth noting that the formalism proposed by Chemero differs from the interpretation of affordance that is common within the robotics community.

2 Technical Approach

We define a *goal-based action prior* as knowledge added to a family of related Markov Decision Processes (MDPs) from the same domain. 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 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). MDPs may also include terminal states that cause all action to cease once reached (such as in goal-directed tasks).



(a) Domain

(b) Example goals.

Figure 2: Part of the OO-MDP Domain for Minecraft.

Goal-based action priors build on Object-Oriented MDPs (OO-MDPs) (Diuk, Cohen, and Littman 2008). 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 value 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. Since predicates operate on collections of objects, they generalize beyond specific states within the domain. For instance, in Minecraft, a predicate checking the contents of the agent's inventory generalizes to many states across many Minecraft tasks. We capitalize on this generalization by using OO-MDP predicates as features for action pruning.

Following STRIPs-like terminology, we define a *domain* as an OO-MDP for which the reward function and terminal states are unspecified. Furthermore, a *problem* of the domain is a completion of the domain's underspecified OO-MDP, wherein a reward function, terminal states, and an initial state are provided. We are concerned with OO-MDPs where the reward function is goal-directed; the agent receives a negative reward at each time step that motivates it to reach the goal state, which is terminal for all problems. A goal, g, is a predicate operating on states that specifies the terminal states of a problem. Figure 2 shows part of the OO-MDP domain definition for Minecraft that we used and the types of goals used in different Minecraft problems that we tested. Figure 3 shows three example initial states and goals for simple Minecraft problems.

(a) Mine the gold and smelt it in the furnace

(b) Dig down to the gold and mine it, avoiding lava.

(c) Navigate to the goal location, avoiding lava.

Figure 3: Three different problems from the Minecraft domain.

Modeling the Optimal Actions

Our goal is to formalize planning knowledge that allows an agent to avoid searching suboptimal actions in each state based on the agent's current goal. We define the optimal action set, $\mathcal{A}^*$, for a given state s and goal G as:

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

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

We aim to learn a probability distribution over the optimality of each action for a given state (s) and goal (G). Thus, we want to infer a Bernoulli distribution for each action's optimality:

$$\Pr(a_i \in \mathcal{A}^* \mid s, G) \quad (2)$$

for $i \in \{1, \ldots, |\mathcal{A}|\}$, where $\mathcal{A}$ is the OO-MDP action space for the domain.

To generalize across specific low-level states, we abstract the state and goal into a set of n paired preconditions and goals, $\{(p_1, g_1) \ldots (p_n, g_n)\}$. We abbreviate each pair (p_j, g_j) to δ_j for simplicity. Each precondition $p \in \mathcal{P}$ is a *predicate* in the set of predicates $\mathcal{P}$ defined by the OO-MDP domain, and G is a *goal* which is a predicate on states that is true if and only if a state is terminal. For example, a predicate might be $nearTrench(agent)$ which is true when the agent is standing near a trench. In general a precondition is an arbitrary logical expression of the state; in our experiments we used unary predicates defined in the OO-MDP domain. A goal specifies the sort of problem the agent is trying to solve, such as the agent retrieving an object of a certain type from the environment, reaching a particular location, or creating a new structure. Depending on the agent's current goal, the relevance of each action changes dramatically. We rewrite Equation 2:

$$\Pr(a_i \in \mathcal{A}^* \mid s, G) = \Pr(a_i \in \mathcal{A}^* \mid s, G, \delta_1 \ldots \delta_n). \quad (3)$$

We introduce the indicator function f, which returns 1 if and only if the given δ's predicate is true in the provided state s, and δ's goal is entailed by the agent's current goal, G:

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

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:

$$\Pr(a_i \in \mathcal{A}^* \mid s, G, \delta_1 \ldots \delta_n) = \Pr(a_i \in \mathcal{A}^* \mid \phi_1, \ldots, \phi_n) \quad (5)$$

This distribution may be modeled in a number of ways, making this approach quite flexible. One model that can be easily specified by an expert is an OR model. In the OR model some subset of the features ($\phi^i \subset \phi$) are assumed to cause action a_i to be optimal; as long as one of the features is on, the probability that a_i is optimal is one. If none of the features are on, then the probability that a_i is optimal is zero. More formally,

$$\Pr(a_i \in \mathcal{A}^* \mid \phi_1, \ldots, \phi_n) = \phi^i_1 \vee \ldots \vee \phi^i_m, \quad (6)$$

where m is the number of features that can cause a_i to be optimal ($m = |\phi^i|$).

In practice, we do not expect such a distribution to be reflective of reality; if it were, then no planning would be needed because a full policy would have been specified. However, it does provide a convenient way for a designer to provide conservative background knowledge. Specifically, a designer can consider each precondition-goal pair and specify the actions that could be optimal in that context, ruling out actions that would be known to be irrelevant or dependent on other state features being true. For example, Table 1 shows example expert-provided conditions that we used in our Minecraft experiments.

Because the OR model is not expected to be reflective of reality and because of other limitations (such as not allowing support for an action to be provided when a feature is off), the model is not practical for learning. Learned priors have the potential to outperform hand-coded priors by more flexibly adapting to the features that predict optimal actions over a large training set. An alternative more expressive model that does lend itself to learning is Naive Bayes. We first factor using Bayes' rule, introducing a parameter vector θ_i of feature weights:

$$= \frac{\Pr(\phi_1, \ldots, \phi_n, \mid a_i \in \mathcal{A}^*, \theta_i) \Pr(a_i \in \mathcal{A}^* \mid \theta_i)}{\Pr(\phi_1, \ldots, \phi_n | \theta_i)} \quad (7)$$

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

$$= \frac{\prod_{j=1}^{n} \Pr(\phi_j \mid a_i \in \mathcal{A}^*, \theta_i) \Pr(a_i \in \mathcal{A}^* \mid \theta_i)}{\Pr(\phi_1, \ldots, \phi_n | \theta_i)} \quad (8)$$

Finally, we define the prior on the optimality of each action to be the fraction of the time each action was optimal during training. Although we only explore the OR and Naive Bayes models in this work, other models, like logistic regression and Noisy-OR, could also be used.

Learning the Optimal Actions

Using the above model allows us to learn goal-based action priors through experience. We provide a set of training worlds from the domain (W), for which the optimal policy, π, may be tractably computed using existing planning methods. We compute model parameters using the small training worlds, and then evaluate performance on a different set of much harder problems at test time. To compute model parameters using Naive Bayes, we compute the maximum likelihood estimate of the parameter vector θ_i for each action using the policy.

Under our Bernouli Naive Bayes model, we estimate the parameters $\theta_{i,0} = \Pr(a_i)$ and $\theta_{i,j} = \Pr(\phi_j | a_i)$, for $j \in \{1, \ldots, n\}$, where the maximum likelihood estimates are:

$$\theta_{i,0} = \frac{C(a_i)}{C(a_i) + C(\bar{a}_i)} \tag{9}$$

$$\theta_{i,j} = \frac{C(\phi_j, a_i)}{C(a_i)} \tag{10}$$

Here, $C(a_i)$ is the number of observed occurrences where a_i was optimal across all worlds W, $C(\bar{a}_i)$ is the number of observed occurrences where a_i was not optimal, and $C(\phi_j, a_i)$ is the number of occurrences where $\phi_j = 1$ and a_i was optimal. We determined optimality using the synthesized policy for each training world, π_w. More formally:

$$C(a_i) = \sum_{w \in W} \sum_{s \in w} (a_i \in \pi_w(s)) \tag{11}$$

$$C(\bar{a}_i) = \sum_{w \in W} \sum_{s \in w} (a_i \notin \pi_w(s)) \tag{12}$$

$$C(\phi_j, a_i) = \sum_{w \in W} \sum_{s \in w} (a_i \in \pi_w(s) \wedge \phi_j == 1) \tag{13}$$

During the learning phase, the agent learns which actions are useful under different conditions. For example, consider the three different problems shown in Figure 3. During training, we observe that the `destroy` action is often optimal when the agent is looking at a block of gold ore and the agent is trying to smelt gold ingots. Likewise, when the agent is not looking at a block of gold ore in the smelting task we observe that the `destroy` action is generally not optimal (i.e. destroying grass blocks is typically irrelevant to smelting). This information informs the distribution over the optimality of the `destroy` action, which is used at test time to encourage the agent to destroy blocks when trying to smelt gold and looking at gold ore, but not in other situations (unless the prior suggests using `destroy`). Example learned priors are shown in 2.

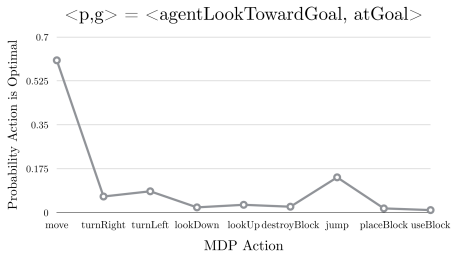
At test time, the agent will see different, randomly generated worlds from the same domain, and use the learned priors to increase its speed at inferring a plan. For simplicity, our learning process uses a strict separation between training and test; after learning is complete our model parameters remain fixed.

Action Pruning with Goal-Based Action Priors

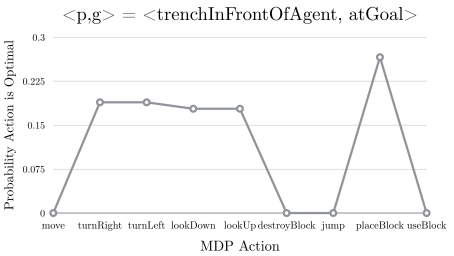
A planner using a goal-based action prior will prune actions on a state-by-state basis.

Under the expert specified OR model, when $\Pr(a_i \in \mathcal{A}^* \mid \phi_1, \ldots, \phi_n) = 0$ action a_i is pruned from the planner's consideration. When $\Pr(a_i \in \mathcal{A}^* \mid \phi_1, \ldots, \phi_n) = 1$, action a_i remains in the action set to be searched by the planner.

When a model like Naive Bayes is used, a less restrictive approach must be taken. In this work, we prune actions whose probability is below some threshold and keep the rest. Empirically, we found the heuristic of setting the threshold to $\frac{0.2}{|\mathcal{A}|}$ to be effective (where $|\mathcal{A}|$ is the size of the full action space of the OO-MDP). This threshold is quite conservative and means that only actions that are extremely unlikely to be optimal are pruned. In the future, we plan on exploring stochastic action pruning methods with anytime planning algorithms, but an advantage to this threshold pruning approach is that it can be used in a large range of different planning algorithms including A* (for deterministic domains), Value Iteration, and Real-time Dynamic Programming (RTDP) (Barto, Bradtke, and Singh 1995). In this work, we present results using RTDP.



(a) agentLookTowardGoal



(b) trenchInFrontOfAgent

Figure 4: When the agent is looking toward its goal location, it is generally better to move forward than do anything else. Alternatively, when the agent is faced with a trench, the agent should walk along the trench to look for gaps, or build a bridge across by looking down and placing blocks.

Precondition	Goal	Actions
`lookingTowardGoal`	`atLocation`	{`move`}
`lavaInFront`	`atLocation`	{`rotate`}
`lookingAtGold`	`hasGoldOre`	{`destroy`}

Table 1: Example of an expert-provided goal-based action prior.

3 Results

We evaluate our approach using the game Minecraft and a collaborative robotic cooking task. 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. 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 un-

Planner	Bellman	Reward	CPU
Mining Task			
RTDP	17142.1 (±3843)	**-6.5** (±1)	**17.6s** (±4)
EP-RTDP	14357.4 (±3275)	**-6.5** (±1)	31.9s (±8)
LP-RTDP	**12664.0** (±9340)	-12.7 (±5)	33.1s (±23)
Smelting Task			
RTDP	30995.0 (±6730)	**-8.6** (±1)	45.1s (±14)
EP-RTDP	28544.0 (±5909)	**-8.6** (±1)	72.6s (±19)
LP-RTDP	**2821.9** (±662)	-9.8 (±2)	**7.5s** (±2)
Wall Traversal Task			
RTDP	45041.7 (±11816)	-56.0 (±51)	**68.7s** (±22)
EP-RTDP	32552.0 (±10794)	-34.5 (±25)	96.5s (±39)
LP-RTDP	**24020.8** (±9239)	**-15.8** (±5)	80.5s (±34)
Trench Traversal Task			
RTDP	16183.5 (±4509)	**-8.1** (±2)	53.1s (±22)
EP-RTDP	**8674.8** (±2700)	-8.2 (±2)	**35.9s** (±15)
LP-RTDP	11758.4 (±2815)	-8.7 (±1)	57.9s (±20)
Plane Traversal Task			
RTDP	52407 (±18432)	-82.6 (±42)	877.0s (±381)
EP-RTDP	32928 (±14997)	-44.9 (±34)	505.3s (±304)
LP-RTDP	**19090** (±9158)	**-7.8** (±1)	**246s** (±159)

Table 2: RTDP vs. EP-RTDP vs. LP-RTDP

certain environment. Figure 3 shows three example scenes from Minecraft problems that we explore. Additionally, we used expert-provided priors to enable a manipulator robot to infer helpful actions in response to a person working on a kitchen task, shown in Figure 6.

Minecraft

Our experiments consist of five common goals in Minecraft: bridge construction, gold smelting, tunneling through walls, digging to find an object, and path planning.

The training set consists of 20 randomly generated instances of each goal, for a total of 100 instances. Each instance is extremely simple: 1,000-10,000 states (small enough to solve with tabular approaches). The output of our training process is the model parameter θ, which informs our goal-based action prior. The full training process takes approximately one hour run in parallel on a computing grid, with the majority of time devoted to computing the optimal value function for each training instance.

The test set consists of 20 randomly generated instances of the same goal, for a total of 100 instances. Each instance is extremely complex: 50,000-1,000,000 states (which is far too large to solve with tabular approaches).

We fix the number of features at the start of training based on the number predicates defined by the OO-MDP, $|\mathcal{P}|$, and the number of goals, $|G|$. We provide our system with a set of 51 features that are likely to aid in predicting the correct action across instances.

We use Real-Time Dynamic Programming (RTDP) (Barto, Bradtke, and Singh 1995) as our baseline planner, a sampling-based algorithm that does not require the planner to exhaustively explore states. We

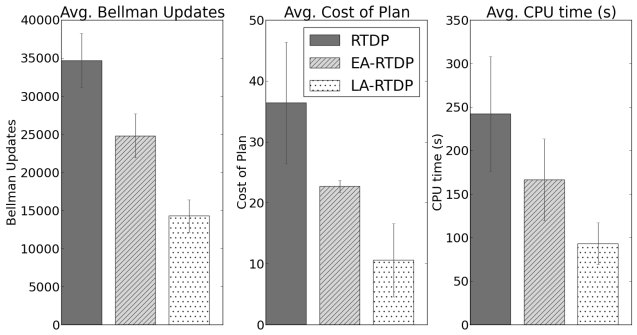


Figure 5: Average results from all maps.

compare RTDP with learned priors RTDP (LP-RTDP), and expert priors RTDP (EP-RTDP). We terminate each planner when the maximum change in the value function is less than 0.01 for 100 consecutive policy rollouts, or the planner fails to converge after 1000 rollouts. The reward function is -1 for all transitions, except transitions to states in which the agent is in lava, where we set the reward to -10. The goal specifies terminal states, and the discount factor is $\gamma = 0.99$. To introduce non-determinism into our problem, movement actions (move, rotate, jump) in all experiments have a small probability (0.05) of incorrectly applying a different movement action. This noise factor approximates noise faced by a physical robot that attempts to execute actions in a real-world domain and can affect the optimal policy due to the existence of lava pits that the agent can fall into.

We report the number of Bellman updates executed by each planning algorithm, the accumulated reward of the average plan, and the CPU time taken to find a plan. Table 2 shows the average Bellman updates, accumulated reward, and CPU time for RTDP, LP-RTDP and EP-RTDP after planning in 20 different maps of each goal (100 total). Figure 5 shows the results averaged across all maps. We report CPU time for completeness, but our results were run on a networked cluster where each node had differing computer and memory resources. As a result, the CPU results have some variance not consistent with the number of Bellman updates in Table 2. Despite this noise, overall the average CPU time shows statistically significant improvement overall with our priors, as shown in Figure 5. Furthermore, we reevaluate each predicate every time the agent visits a state, which could be optimized by caching predicate evaluations, further reducing the CPU time taken for EP-RTDP and LP-RTDP.

Because the planners terminate after a maximum of 1000 rollouts, they do not always converge to the optimal policy. LP-RTDP on average finds a comparably better plan (10.6 cost) than EP-RTDP (22.7 cost) and RTDP (36.4 cost), in significantly fewer Bellman updates (14287.5 to EP-RTDP's 24804.1 and RTDP's 34694.3), and in less CPU time (93.1s to EP-RTDP's 166.4s and RTDP's 242.0s). These results indicate that while learned priors provide the largest improvements, expert-provided priors can also significantly enhance

performance. Depending on the domain, expert-provided priors can add significant value in making large state spaces tractable without the overhead of supplying training worlds.

For some task types, LP-RTDP finds a slightly worse plan on average than RTDP (*e.g.* the mining task). This worse convergence is due to the fact that LP-RTDP occasionally prunes actions that are in fact optimal (such as pruning the `destroy` action in certain states of the mining task). Additionally, RTDP occasionally achieved a faster clock time because EP-RTDP and LP-RTDP also evaluate several OO-MDP predicates in every state, adding a small amount of time to planning.

Temporally Extended Actions and Goal-Based Action Priors

We compare our approach to temporally extended actions: macro-actions and options. We conduct these experiments with the same configurations as our Minecraft experiments. Domain experts provide the option policies and macro-actions.

Table 3 indicates the results of comparing RTDP equipped with macro-actions, options, and goal-based priors across 100 different executions in the same randomly generated Minecraft worlds. The results are averaged across goals of each type presented in Table 2. Both macro-actions and options add a significant amount of time to planning due to the fact that the options and macro-actions are being reused in multiple OO-MDPs that each require recomputing the resulting transition dynamics and expected cumulative reward when applying each option/macro-action (a cost that is typically amortized in classic options work where the same OO-MDP state space and transition dynamics are used). This computational cost might be reduced when using a Monte Carlo planning algorithm that does not need the full transition dynamics and expected cumulative reward. Furthermore, the branching factor of the state-action space significantly increases with additional actions, causing the planner to run for longer and perform more Bellman updates. Despite these extra costs in planning time, earned reward with options was higher than without, demonstrating that our expert-provided options add value to the system.

With goal-based action priors, the planner finds a better plan in less CPU time, and with fewer Bellman updates. These results support the claim that priors can handle the augmented action space provided by temporally extended actions by pruning away unnecessary actions, and that options and goal-based action priors provide complementary information.

Planner	Bellman	Reward	CPU
`RTDP`	27439 (±2348)	-22.6 (±9)	107 (±33)
`LP-RTDP`	**9935** (±1031)	**-12.4** (±1)	**53** (±5)
`RTDP+Opt`	26663 (±2298)	-17.4 (±4)	129(±35)
`LP-RTDP+Opt`	**9675** (±953)	**-11.5** (±1)	**93** (±10)
`RTDP+MA`	31083 (±2468)	-21.7 (±5)	336 (±28)
`LP-RTDP+MA`	**9854** (±1034)	**-11.7** (±1)	**162** (±17)

Table 3: Priors with Temporally Extended Actions

Planner	Bellman	Reward	CPU
Dry Ingredients			
`RTDP`	20000 (±0)	-123.1 (±0)	56.0s (±2.9)
`EP-RTDP`	**2457.2** (±53.2)	**-6.5** (±0)	**10.1s** (±0.3)
Wet Ingredients			
`RTDP`	19964 (±14.1)	-123.0 (±0)	66.6s (±9.9)
`EP-RTDP`	**5873.5** (±53.7)	**-6.5** (±0)	**15.6s** (±1.2)
Brownie Batter			
`RTDP`	20000 (±0)	-123.4 (±0.7)	53.3s (±2.4)
`EP-RTDP`	**6642.4** (±36.4)	**-7.0** (±0)	**31.9s** (±0.4)

Table 4: RTDP vs. EP-RTDP for robotic kitchen tasks

Cooking Robot

To assess goal-based action priors applied to a real-world robotic task, we created a cooking domain that requires the robot to choose helpful actions for a person following a recipe. The human participant and the robotic companion are each modeled as separate OO-MDPs. From the robot's perspective, the human is just a stochastic element of its OO-MDP's transition dynamics.

The robot's OO-MDP contained three spaces: human counter, robot counter, sink; four ingredient bowls, two mixing bowls, and two tools that could be in any of the three spaces, in any configuration. Additionally, the robot's OO-MDP contains the following ingredients: cocoa powder, sugar, eggs, and flour. Each container/tool may occupy one of three spaces, and each ingredient in one of the containers is either mixed or unmixed.

Although this fine-grained state space is much larger than needed for any one recipe, it enables support for a variety of different recipes, ranging from brownies to mashed potatoes. Because of its fine-grained nature, our cooking state space has 4.73×10^7 states when configured with the ingredients and tools necessary to make brownies.

We divide a brownie recipe into three subgoals: combining and mixing the dry ingredients, combining and mixing the wet ingredients, and combining these two mixtures into a

Figure 6: Goal-based action priors enable a robot to efficiently infer helpful actions in very large state spaces, such as a kitchen.

batter. For each subgoal, we provide action priors specific to the objects used in that subgoal; for example, a whisk should only be used to mix wet ingredients. We use EP-RTDP to search for the least-cost plan to complete the recipe. The robot infers actions such as handing off the whisk to the person to mix the wet ingredients.

In Table 4 we compare between standard RTDP and EP-RTDP planning for each of the three subgoals. Because the state-action space is reduced significantly, EP-RTDP can plan successfully in a short amount of time. Standard RTDP always encountered the maximum number of rollouts specified at the maximum depth each time, even with a relatively small number of objects. Unlike our previous CPU time results, these experiments were conducted on the same multicore computer.

EP-RTDP running on a robot can help a person cook by dynamically replanning through constant observations. After observing the placement of a cocoa container in the robot's workspace, the robot fetches a wooden spoon to allow the person to mix. After observing an egg container, the robot fetches a whisk to help beat the eggs. The robot dynamically resolves failures and accounts for unpredictable user actions; in the video, the robot fails to grasp the wooden spoon on the first attempt and must retry the grasp after it observed no state change.

4 Related Work

This paper builds on previous work published at two workshops (Barth-Maron et al. 2014; Abel et al. 2014). In this section, we discuss the differences between goal-based action priors and other forms of knowledge engineering that have been used to accelerate planning.

Stochastic Approaches

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* (Hauskrecht et al. 1998) and *options* (Sutton, Precup, and Singh 1999). 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 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 (Konidaris and Barto 2007; 2009; Ravindran and Barto 2003; Andre and Russell 2002; Konidaris, Scheidwasser, and Barto 2012).

Since temporally extended actions may negatively impact planning time (Jong 2008) by adding to the number of actions the agent can choose from in a given state, combining our priors with temporally extended actions allows for even further speedups in planning, as demonstrated in Table 3. In other words, goal-based action priors are complementary knowledge to options and macro-actions.

Sherstov and Stone (Sherstov and Stone 2005) considered MDPs 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. Goal-based action priors prune away actions on a state-by-state basis, enabling more aggressive pruning whereas the learned action pruning is on per-task level.

Rosman and Ramamoorthy (Rosman and Ramamoorthy 2012) 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. These action priors can only be used with planning/learning algorithms that work well with an ϵ-greedy rollout policy, while our goal-based action priors can be applied to almost any MDP solver. Their action priors are only active for a fraction ϵ of the time, which is quite small, limiting the improvement they can make to the planning speed. Finally, as variance in tasks explored increases, the priors will become more uniform. In contrast, goal-based action priors can handle a wide variety of tasks in a single prior, as demonstrated by Table 2.

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 (Hansen and Zilberstein 1999), or reward shaping (Ng, Harada, and Russell 1999). However, heuristics are highly dependent on the reward function and state space of the task being solved, whereas goal-based action priors are state space independent and may be learned easily for different reward functions. If a heuristic can be provided, the combination of heuristics and our priors may even more greatly accelerate planning algorithms than either approach alone.

Previous approaches, such as KnowRob (Tenorth and Beetz 2012; 2009) have developed complex reasoning systems designed to integrate existing forms of knowledge from a variety of sources for use in robotics applications. These systems emphasize knowledge representation and processing rather than planning. KnowRob, in particular, works "by exploiting existing sources of knowledge as much as possible," as opposed to learning through simulation.

Deterministic Approaches

There have been several 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 interesting parallels nonetheless.

Hierarchical Task Networks (HTNs) employ *task decompositions* to aid in planning (Erol, Hendler, and Nau 1994). The agent decomposes the goal into smaller tasks which are in turn decomposed into smaller tasks. This decomposition continues until immediately achievable primitive tasks 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 HTNs and goal-based action priors both achieve action pruning by exploiting some

form of supplied knowledge. We speculate that the additional action pruning provided by our approach is complementary to the pruning offered by HTNs.

One significant difference between HTNs and our planning system is that HTNs do not incorporate reward into their planning. Additionally, the degree of supplied knowledge in HTNs far exceeds that of our priors: HTNs require not only constraints for sub-tasks but a hierarchical framework of arbitrary complexity. Goal-based action priors require either simple symbolic knowledge, as illustrated in Table 1, or a set of predicates for use as features and a means of generating training instances.

An extension to the HTN is the probabilistic Hierarchical Task Network (pHTN) (Li et al. 2010). In pHTNs, the underlying physics of the primitive actions are deterministic. The goal of pHTN planning is to find a sequence of deterministic primitive actions that satisfy the task, with the addition of matching user preferences for plans, which are expressed as probabilities for using different HTN methods. As a consequence, the probabilities in pHTNs are with regard to probabilistic search rather than planning in stochastic domains, as we do.

Bacchus and Kabanza (Bacchus and Kabanza 1995; 1999) 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 pruning candidate plans that falsify the given knowledge base of LTL formulas, often achieving polynomial time planning in exponential space. LTL formulas are difficult to learn, placing dependence on an expert, while we demonstrate that our priors can be learned from experience.

Models

Our planning approach relies critically on the the ability of the OO-MDP to express properties of objects in a state, which is shared by other models such as First-Order MDPs (FOMDPs) (Boutilier, Reiter, and Price 2001). As a consequence, a domain that can be well expressed by a FOMDP may also benefit from our planning approach. However FOMDPs are purely symbolic, while OO-MDPs can represent states with objects defined by numeric, relational, categorical, and string attributes. Moreover, OO-MDPs enable predicates to be defined that are evaluative of the state rather than attributes that define the state, which makes it easy to add high-level information without adding complexity to the state definition and transition dynamics to account for them. These abilities make OO-MDPs better-suited for the kind of robotic tasks in which we are interested, since it is common to have different objects with spatial properties that are best expressed with numeric values.

5 Conclusion

We propose a novel approach to representing transferable planning knowledge in terms of *goal-based action priors*. These priors allow an agent to efficiently prune actions based on learned or expert provided knowledge, significantly reducing the number of state-action pairs the agent needs to evaluate in order to act near optimally. We demonstrate the effectiveness of these priors by comparing RTDP with and without goal-based action priors in a series of challenging planning tasks in the Minecraft domain. Further, we designed a learning process that allows an agent to autonomously learn useful priors that may be used across a variety of task types, reward functions, and state spaces, allowing for convenient extensions to robotic applications. Additionally, we compared the effectiveness of augmenting planners with priors, temporally extended actions, and the combination of the two. The results suggest that our priors may be combined with temporally extended actions to provide improvements in planning. Lastly, we deploy EP-RTDP on a robot in a collaborative cooking task, showing significant improvements over RTDP.

In the future, we hope to automatically discover useful state space specific subgoals online—a topic of some active research (Mcgovern and Barto 2001; Şimşek, Wolfe, and Barto 2005). Automatic discovery of subgoals would allow goal-based action priors to take advantage of the task-oriented nature of our priors, and would further reduce the size of the explored state-action space by improving the effectiveness of action pruning. Another promising direction to explore is an on-line approach to learning as opposed to the batch style presented here. In an online learning paradigm, the agent would modify its prior over action optimality after each action execution as opposed to separating training and test instances. We are also investigating methods to stochastically prune actions rather than requiring a hard threshold parameter.

References

Abel, D.; Barth-Maron, G.; MacGlashan, J.; and Tellex, S. 2014. Toward affordance-aware planning. In *First Workshop on Affordances: Affordances in Vision for Cognitive Robotics*.

Andre, D., and Russell, S. 2002. State abstraction for programmable reinforcement learning agents. In *Eighteenth national conference on Artificial intelligence*, 119–125. American Association for Artificial Intelligence.

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

Bacchus, F., and Kabanza, F. 1999. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116:2000.

Barth-Maron, G.; Abel, D.; MacGlashan, J.; and Tellex, S. 2014. Affordances as transferable knowledge for planning agents. In *2014 AAAI Fall Symposium Series*.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1):81–138.

Bollini, M.; Tellex, S.; Thompson, T.; Roy, N.; and Rus, D. 2012. Interpreting and executing recipes with a cooking robot. In *Proceedings of International Symposium on Experimental Robotics (ISER)*.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24:581–621.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order mdps. In *IJCAI*, volume 1, 690–697.

Chemero, A. 2003. An outline of a theory of affordances. *Ecological psychology* 15(2):181–195.

Şimşek, O.; Wolfe, A. P.; and Barto, A. G. 2005. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22Nd International Conference on Machine Learning*, 816–823.

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

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Htn planning: Complexity and expressivity. In *AAAI*, volume 94, 1123–1128.

Gibson, J. 1977. The concept of affordances. *Perceiving, acting, and knowing* 67–82.

Hansen, E. A., and Zilberstein, S. 1999. Solving markov decision problems using heuristic search. In *Proceedings of AAAI Spring Symposium on Search Techniques from Problem Solving under Uncertainty and Incomplete Information*.

Hauskrecht, M.; Meuleau, N.; Kaelbling, L. P.; Dean, T.; and Boutilier, C. 1998. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, 220–229. Morgan Kaufmann Publishers Inc.

Jong, N. K. 2008. The utility of temporal abstraction in reinforcement learning. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*.

Knepper, R. A.; Tellex, S.; Li, A.; Roy, N.; and Rus, D. 2013. Single assembly robot in search of human partner: Versatile grounded language generation. In *Proceedings of the HRI 2013 Workshop on Collaborative Manipulation*.

Konidaris, G., and Barto, A. 2007. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI '07, 895–900.

Konidaris, G., and Barto, A. 2009. Efficient skill learning using abstraction selection. In *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*, 1107–1112.

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

Koppula, H. S., and Saxena, A. 2013. Anticipating human activities using object affordances for reactive robotic response. In *Robotics: Science and Systems (RSS)*.

Koppula, H. S.; Gupta, R.; and Saxena, A. 2013. Learning human activities and object affordances from rgb-d videos. *International Journal of Robotics Research*.

Li, N.; Cushing, W.; Kambhampati, S.; and Yoon, S. 2010. Learning probabilistic hierarchical task networks to capture user preferences. *arXiv preprint arXiv:1006.0274*.

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

Mojang. 2014. Minecraft. http://minecraft.net.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, 968–973.

Newton, M.; Levine, J.; and Fox, M. 2005. Genetically evolved macro-actions in ai planning problems. *Proceedings of the 24th UK Planning and Scheduling SIG* 163–172.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, 278–287.

Ravindran, B., and Barto, A. 2003. An algebraic approach to abstraction in reinforcement learning. In *Twelfth Yale Workshop on Adaptive and Learning Systems*, 109–144.

Rosman, B., and Ramamoorthy, S. 2012. What good are actions? accelerating learning using learned action priors. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, 1–6. IEEE.

Sherstov, A., and Stone, P. 2005. Improving action selection in mdp's via knowledge transfer. In *Proceedings of the 20th national conference on Artificial Intelligence*, 1024–1029. AAAI Press.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1):181–211.

Tenorth, M., and Beetz, M. 2009. Knowrobknowledge processing for autonomous personal robots. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 4261–4266. IEEE.

Tenorth, M., and Beetz, M. 2012. Knowledge processing for autonomous robot control. In *AAAI Spring Symposium: Designing Intelligent Robots*.

Thrun, S.; Burgard, W.; and Fox, D. 2008. *Probabilistic robotics*. MIT Press.