In *The Twelfth International Conference on Autonomous Agents and Multiagent Systems* (AAMAS 2013), St. Paul, MN, May 2013.

Learning Exploration Strategies in Model-Based Reinforcement Learning

Todd Hester
Dept of Computer Science
University of Texas at Austin
todd@cs.utexas.edu

Manuel Lopes
INRIA Bordeaux Sud-Ouest
Bordeaux, France
manuel.lopes@inria.fr

Peter Stone
Dept of Computer Science
University of Texas at Austin
pstone@cs.utexas.edu

ABSTRACT

Reinforcement learning (RL) is a paradigm for learning sequential decision making tasks. However, typically the user must hand-tune exploration parameters for each different domain and/or algorithm that they are using. In this work, we present an algorithm called LEO for learning these exploration strategies on-line. This algorithm makes use of bandit-type algorithms to adaptively select exploration strategies based on the rewards received when following them. We show empirically that this method performs well across a set of five domains. In contrast, for a given algorithm, no set of parameters is best across all domains. Our results demonstrate that the LEO algorithm successfully learns the best exploration strategies on-line, increasing the received reward over static parameterizations of exploration and reducing the need for hand-tuning exploration parameters.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

Reinforcement Learning, Exploration

1. INTRODUCTION

Reinforcement learning (RL) is a paradigm for learning sequential decision making tasks, where an agent seeks to maximize long-term rewards through experience in its environment. During the learning process the agent has to decide whether to look for new information (explore) or to use its current model to maximize reward (exploit). In addition, when choosing to explore, the agent must decide *what* and/or *where* to explore. Many exploration strategies have been proposed which aim to ensure that: 1) the agent will have enough information to find the optimal policy; and 2) the agent will find the optimal policy in as few actions as possible. Some of these exploration methods provide theoretical bounds on how many steps the agent will take to learn the optimal policy.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.

Nevertheless, for any particular problem the theoretical bounds do not indicate which exploration strategy will perform the best [5, 13]. Typically, RL algorithms have a number of exploration parameters to determine how the agent explores and how it solves the exploration-exploitation trade-off. Using such methods requires the user to hand-tune these exploration parameters.

In this paper, we present a novel algorithm called LEO (Learning Exploration On-line) that is given a set of exploration strategies and learns which strategy is best while interacting with the environment. This new algorithm leverages all previous results on exploration by enabling the agent to choose which one works best for the task at hand. We treat each strategy as one arm of a multi-armed bandit and utilize some insights from bandit algorithms [2] to learn which strategy is providing the best results. We present results showing that LEO performs better than alternative parameterizations of exploration within the TEXPLORE-VANIR RL algorithm [10, 11].

2. BACKGROUND

We adopt the standard Markov Decision Process (MDP) formalism for this work [21]. An MDP is defined by a set of states S, a set of actions A, a reward function $R(s,a)$, and a transition function $T(s,a,s') = P(s'|s,a)$. In each state $s \in S$, the agent takes an action $a \in A$. Upon taking this action, the agent receives a reward $R(s,a)$ and reaches a new state s', determined from the probability distribution $P(s'|s,a)$. Many domains utilize a factored state representation, where the state s is represented by a vector of n state variables: $s = \langle s_1, s_2, ..., s_n \rangle$. A policy $\pi = P(a|s)$ specifies for each state a distribution over actions which the agent will take.

The goal of the agent is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agent's lifetime:

$$J = \sum_t \gamma^t r_t, \tag{1}$$

where $0 < \gamma < 1$ is the discount factor and r_t is the reward obtained at time step t. The value $Q^\pi(s,a)$ of a given state-action pair is an estimate of the expected future reward that can be obtained from (s,a) when following policy π. The optimal value function $Q^*(s,a)$ provides maximal values in all states and is determined by solving the Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a'). \tag{2}$$

The optimal policy π is then as follows:

$$\pi(s) = \mathrm{argmax}_a Q^*(s, a). \tag{3}$$

2.1 On-line Selection of Experts

LEO makes use of bandit algorithms to choose which exploration strategy to follow in a domain. In a bandit problem [1, 2], a gambler must select which of k slot machine arms to pull. Each arm provides some payoff, and the gambler's goal is to pull the correct arms in order to maximize the total payoff over a number of trials. These methods see each exploration strategy as a different arm whose value must be estimated. Algorithms for bandit problems have been used to select different active learning strategies online [3, 12]. Many of these ideas have been integrated in a unified perspective under the name of the *strategic student problem* [16] that considers choices in an abstract way that can include active learning methods, exploration methods or even multi-task problems.

2.2 Model-Based Exploration Strategies

RL methods fall into two general classes: model-based and model-free methods. Model-based RL methods learn a model of the domain by approximating $R(s, a)$ and $P(s'|s, a)$ for each state and action. The agent can then calculate a policy (i.e. plan) using this model through a method such as value iteration, effectively updating the Bellman equations for each state using its model. Model-free methods work without a model, typically updating the values of actions only when taking them in the real task. Generally model-based methods are more sample efficient than model-free methods. Model-free methods must visit each state many times for the value function to converge, while the sample efficiency of model-based methods is only constrained by how many samples it takes to learn a good model.

An advantage of model-based methods is their ability to plan multi-step exploration trajectories. One of the best known examples of model-based reinforcement learning is R-MAX, which is guaranteed to learn a near-optimal policy within polynomial time. For finite MDPs, this algorithm learns a maximum-likelihood tabular model of the environment. The algorithm classifies each state-action as known or unknown according to the number of times it was visited. When planning on the model, known state-actions are modeled with the learned reward, while unknown state-actions are given the maximum one-step reward in the domain, r_{max}. At each step, the agent acts greedily with respect to this modified MDP. Assuming that unknown states have a reward of r_{max} drives the agent to explore them. This intrinsic reward is one example of an *exploration strategy.*

There are many other methods which utilize different exploration strategies. For example, there are extensions of R-MAX to factored domains where similar rewards are provided for the agent to explore unvisited state factors [8, 9]. Other methods do not explicitly consider known versus unknown states and instead add exploration bonuses for states with higher potential of learning [4], or with higher uncertainty [13, 15] [14,16]. This type of exploration bonus, or intrinsic reward, was also used in the TEXPLORE-VANIR algorithm [11], which we use as the starting point for the experiments in this paper.

TEXPLORE-VANIR learns random forest models of the domain and then provides two different intrinsic rewards to drive exploration. First, it rewards state-actions for which the different trees within its random forest model differ in their predictions. Second, it rewards the state-actions that are most different from what its model has been trained on.

Each of these algorithms drives exploration in a different way. In fact, the best exploration strategy to use is highly dependent on the model learning approach taken by the algorithm and the domain the agent is acting in. In this work, our goal is to learn the best exploration strategies for a given algorithm and domain on-line.

3. ALGORITHM

In this section, we present our algorithm, Learning Exploration On-line (LEO), for learning the best exploration strategies on-line. It is a general approach that works with any model-based RL method. LEO is given a set of different *exploration strategies* and its goal is to choose the best exploration strategy for each task while interacting with the environment on-line. Since we are concerned with on-line performance of the algorithm, LEO evaluates the performance of each exploration strategy based on the rewards received by the agent while following that strategy. Thus, LEO chooses the exploration strategies that find the rewards and goals the fastest, limiting the costs of exploration by exploring efficiently.

LEO treats each of these exploration strategies like one of the arms in a multi-armed bandit problem. Pseudo-code for our approach is shown in Algorithm 1. Briefly, the agent follows these steps: 1) it selects one of the strategies based on the past payouts received from following it; 2) it follows the selected strategy while tracking the similarity of the other strategies to the one it is following; and 3) at the end of the episode, it updates the expected payouts for each strategy (even the ones not followed). Each step of this process is explained in detail below.

The algorithm is given a set of strategies, E. Each strategy has a weight, w_e, which is an estimate of the expected normalized return for an episode when following that strategy. At the start of each episode, LEO uses these weights to compute a soft-max distribution over the set of strategies, similar to the EXP4 bandit algorithm [2]:

$$P(e) \leftarrow \frac{e^{\beta(w_e - min(w))}}{\sum_j e^{\beta(w_j - min(w))}}. \tag{4}$$

After calculating this distribution, RUN-EPISODE is called on line 5. RUN-EPISODE runs the agent through one episode, sampling strategies from this distribution every 10 steps. A large range of values between the extremes of 1 and an entire episode work well, and 10 was chosen through informal experiments. It is important for LEO to follow a given exploration strategy for multiple steps, but following a bad strategy for an entire episode could greatly impact the agent's performance. At the end, it returns the normalized discounted reward received on the episode and the similarity of each strategy to the followed strategy. This similarity is calculated using importance sampling [17, 21] and is the likelihood of the followed trajectory under this strategy's policy.

After an episode is completed, the estimate of the expected normalized discounted return for each strategy is updated with the following equation on line 6:

$$w_e \leftarrow w_e + \eta \cdot \frac{sim_e}{\sum_f sim_f}(\hat{J} - w_e). \tag{5}$$

Algorithm 1 Learning Exploration On-Line (LEO)

1: **Input:** E ▷ Set of strategies E
2: $w_e \leftarrow 1.0, \forall e \in E$ ▷ Initialize strategy weights
3: **loop** ▷ Loop over episodes
4: $\quad P(e) \leftarrow \frac{e^{\beta(w_e - min(w))}}{\sum_j e^{\beta(w_j - min(w))}}$ ▷ Dist. over strategies
5: $\quad sim, \hat{J} \leftarrow$RUN-EPISODE$(P(e))$
6: $\quad w_e \leftarrow w_e + \eta \cdot \frac{sim_e}{\sum_f sim_f}(\hat{J} - w_e), \forall e \in E$
7: **end loop**

The weight changes are divided between the strategies based on each strategy's proportion of the total similarity, $\frac{sim_e}{\sum_f sim_f}$, so that the sum of the weight changes for all strategies is η, the learning rate. Thus, strategies that were more similar to the followed policy in an episode are moved closer to the return from that episode than strategies that were not as similar. These updated weights then affect the new distribution over strategies calculated before the next episode.

Algorithm 2 shows what LEO does during an episode. Every 10 steps, the algorithm selects a new strategy from the distribution over strategies (line 7). Typically, one of these strategies is to act greedily with respect to the learned model of external reward in the task, and the other strategies' policies maximize other intrinsic rewards for exploration. Through informal testing, we found that strictly following any one of these exploration strategies can lead to poor performance in the task, as they are followed even if they contradict knowledge of the external rewards in the task. Thus, the algorithm plans a separate execution policy, π_x, on line 10. This execution policy combines exploration and exploitation by maximizing both the intrinsic rewards of the selected strategy e as well as the model of task rewards in the domain. The task reward is added in at this phase rather than into each exploration strategy itself so that each exploration strategy remains independent for similarity calculations.

While a particular strategy is being followed, the algorithm tracks the similarity of *all* the strategies, so that their weights can be updated even if they were not selected. Updating values of policies that are not being followed is called *off-policy* learning, and LEO uses a version of importance sampling to address this problem [17, 21]. To track the similarity of the other strategies, at every step, a separate softmax policy is planned for each exploration strategy with the call to PLAN-POLICY on line 9. When an action is taken in the domain, each strategy's similarity is updated by the probability that it would have taken the selected action, $\pi(s, a)$:

$$sim_e \leftarrow sim_e * \pi_e(s, a). \tag{6}$$

Thus, at the end of the episode, the algorithm has a similarity of each strategy's policy to the policy that was actually followed by the agent.

Throughout the episode, LEO tracks the discounted reward, J, that the agent has received. At the end of the episode, it calculates a normalized return $\hat{J}$, where the minimum possible discounted return in the domain is 0 and the maximum possible discounted return in the domain is 1:

$$\hat{J} \leftarrow \frac{J - \frac{r_{min}}{1-\gamma}}{\frac{r_{max}}{1-\gamma} - \frac{r_{min}}{1-\gamma}}. \tag{7}$$

This normalized return is calculated so that the return has some meaning for how well the agent performed across tasks.

Algorithm 2 RUN-EPISODE$(P(e))$

1: **Input:** S, A, E ▷ S: state space, A: action space, E: set of strategies
2: $i \leftarrow 0.0$
3: $sim_e \leftarrow 1.0, \forall e \in E$ ▷ Reset strategy weights
4: $J \leftarrow 0.0$ ▷ Discounted return
5: **while** Episode Not Over **do**
6: $\quad$ **if** i MOD $10 = 0$ **then**
7: $\quad\quad$ Sample strategy b from $P(e \in E)$
8: $\quad$ **end if**
9: $\quad \pi_e \leftarrow$ PLAN-POLICY$(e), \forall e \in E$
10: $\quad \pi_x \leftarrow$ PLAN-POLICY$(b + task)$ ▷ Plan exec. pol.
11: $\quad$ Sample action a from $\pi_x(s, a \in A)$
12: $\quad$ Take action a, observe r, s'
13: $\quad M \leftarrow$ UPDATE-MODEL$(M \langle s, a, s', r \rangle)$
14: $\quad sim_e \leftarrow sim_e * \pi_e(s, a), \forall e \in E$ ▷ Update sim.
15: $\quad s \leftarrow s'$
16: $\quad J \leftarrow J + \gamma^i * r$
17: $\quad i \leftarrow i + 1$
18: **end while**
19: $\hat{J} \leftarrow \frac{J - \frac{r_{min}}{1-\gamma}}{\frac{r_{max}}{1-\gamma} - \frac{r_{min}}{1-\gamma}}$ ▷ Calculate normalized return
20: **return** $sim, \hat{J}$

It is then returned to Algorithm 1 and used to update the weights of the strategies.

4. EMPIRICAL EVALUATION

In this section, we evaluate LEO in comparison with predefined exploration strategies across a set of domains. While a hand-picked exploration strategy can perform well on one domain, the domains were selected so that it would be difficult to find one exploration strategy that was the best across all domains. In addition, finding the best strategy even for a single domain can require a lot of hand-tuning, whereas LEO self-tunes on-line automatically.

4.1 Exploration Strategies

We analyze our approach within the framework of the TEXPLORE-VANIR algorithm [11]. TEXPLORE-VANIR works in factored domains, learning a model of the domain by learning a separate prediction for each of the n state features. In order to learn the model quickly, it incorporates generalization into the model learning, using random forests [7] of decision trees to predict the next state and reward given the current state and action. We chose to examine LEO on this approach because it is not clear in this case what the best exploration strategy is.

TEXPLORE-VANIR tracks multiple hypotheses of the true dynamics of the domain in the form of a random forest. A random forest is a collection of m decision trees, each of which differs because it is trained on a random subset of experiences and has some randomness when choosing splits at the decision nodes [7]. The agent then plans over the average of the predictions made by each tree in the forest. TEXPLORE-VANIR (or TEXPLORE with variance and novelty intrinsic rewards) incorporates two intrinsic rewards for exploration. The first is the VARIANCE reward, which provides rewards for where there is variance between the predictions of each tree in the random forest. The variance in the predictions for a given state-action is multiplied by a coefficient

v and used as intrinsic reward. This reward drives the agent to state-actions where its models have not converged and it may need more experiences. The second intrinsic reward is the NOVELTY reward, which rewards the agent for visiting state-actions which are most different in feature space from the ones that it has visited previously. The L_1 distance in feature space from a state-action to the nearest previously visited state-action is calculated and multiplied by a coefficient n and used as intrinsic reward. This reward drives the agent to explore the state-actions that are most different from what it has been trained on, which are where its trees are most likely to have generalized incorrectly. When running TEXPLORE-VANIR, the user must provide the v and n coefficients to define how much each of these two exploration strategies is weighted relative to each other and to exploiting the model of external task rewards.

For our experiments, LEO is given the following strategies:

1. Maximize model of task reward
2. Use VARIANCE intrinsic reward
3. Use NOVELTY intrinsic reward
4. Reward exploring UNVISITED state-actions
5. Reward maximizing/minimizing individual state features

These strategies were chosen as representative strategies that might be relevant in the evaluation domains.

The first strategy is to maximize the model of the task reward, which is a purely exploitative policy. This enables the agent to learn the exploration-exploitation trade-off on-line, as it can choose to take an exploitative strategy. The second and third strategies are the two from the TEXPLORE-VANIR algorithm. The fourth strategy is similar to the exploration performed by R-MAX [5]. This strategy provides intrinsic rewards for any state-actions that the agent has not visited yet. A parameter, u, defines how much reward is given to unvisited state-actions. Finally, we give the agent strategies that reward or punish particular state features. For example, the agent's reward may be the value of the first state feature, encouraging the agent to maximize this feature, or it could be the negative value of the first feature, encouraging the agent to minimize this feature. For the number of state features in the domain, n, there will be $2n + 4$ strategies: n strategies that maximize the value of each feature, n strategies that minimize the value of each feature, and the first 4 strategies presented above.

We compared against TEXPLORE-VANIR using 6 static parameterizations of the VARIANCE, NOVELTY, and UNVISITED exploration strategies:

1. Greedy ($v = 0, n = 0, u = 0$)
2. VARIANCE only ($v = 5, n = 0, u = 0$)
3. NOVELTY only ($v = 0, n = 5, u = 0$)
4. UNVISITED only ($v = 0, n = 0, u = 5$)
5. LOW V-N ($v = 5, n = 5, u = 0$)
6. HIGH V-N ($v = 80, n = 80, u = 0$)

These 6 options give us a variety of exploration strategies which have been shown to work well in previous research [10, 11]. There are three versions which are only using a single exploration strategy (Num. 2-4), one using no exploration (Num 1), and two which combine the VARIANCE and NOVELTY strategies with different weights compared to the task reward (Num. 5 and 6).

4.2 Domains

We evaluated our algorithm over a set of five domains. We chose a set of domains where no single exploration strategy should perform well across all domains. Rather than hand-tuning the best exploration strategy for each domain, our algorithm can learn the best strategy in each domain on-line without any parameter tuning. We expect that while using the best strategy for one domain will perform better than LEO on that domain, none of the individual strategies will perform well across all five domains.

The first task we tested the algorithms on is called **Fuel World** [10], shown in Figure 1. In it, the agent starts in the middle left of the domain and is trying to reach a terminal state in the middle right of the domain which has a reward of 0. The agent has a fuel level that ranges from 0 to 60. The agent's state vector, s, is made up of three features: its ROW, COL, and FUEL. Each step the agent takes reduces its fuel level by 1. If the fuel level reaches 0, the episode terminates with reward -400. There are fuel stations along the top and bottom row of the domain which increase the agent's fuel level by 20. The agent can move in eight directions: NORTH, EAST, SOUTH, WEST, NORTHEAST, SOUTHEAST, SOUTHWEST, and NORTHWEST. The first four actions each move the agent one cell in that direction and have a reward of -1. The last four actions move the agent to the cell in that diagonal direction and have reward -1.4. The agent starts with a random amount of fuel between 14 and 18, which is not enough to reach the goal, and must learn to go to one of the fuel stations on the top or bottom row before heading towards the goal state.

Actions from a fuel station have an additional cost, which is defined by:

$$R(x) = base - (x \bmod 5)a, \tag{8}$$

where $R(x)$ is the reward of a fuel station in column x, base is a baseline reward for that row, and a controls how much the costs vary across columns. The parameters are $a = 5.0$ and $base = -10.0$ for the bottom row and -13.0 for the top row, resulting in rewards from -10.0 to -33.0. As the agent explores some of the fuel stations, each of its trees may make different hypotheses about this cost function. Therefore, we hypothesize that the VARIANCE exploration will be the best, although its unclear how this should be weighted relative to exploiting the task reward.

The second domain is a version of the **Light World** domain [14], shown in Figure 2. In this domain, the agent is trying to leave a room. The room has a door, a lock, and a key. The agent must go pick up the key and then go press the lock in order to unlock the door. At this point, it can leave the room, terminating the episode. Open doors, locks, and keys each emit a different color light that the agent can see. The agent has sensors that detect each color light in each cardinal direction. The sensors have a maximal value of 1 when the agent is at the light, and their values decrease linearly to 0 when the light is 20 steps away. The agent's state is made up of 17 different features: its X and Y location in the room, whether it has the KEY, whether the door is LOCKED, and the values of the 12 light sensors, which detect each of the three lights in the four cardinal directions. The agent can take six possible actions: it can move in each of the four cardinal directions, PRESS the lock, or PICKUP the key. The PRESS and PICKUP actions are only effective when the agent is on top of the lock and the key. The agent

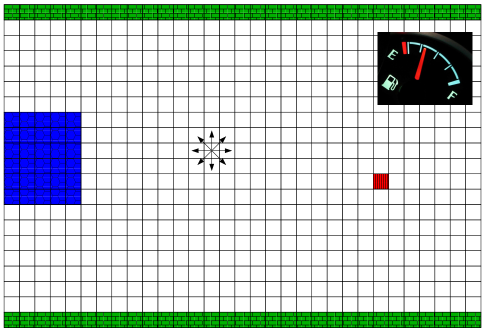

Figure 1: The Fuel World domain. Starting states are blue, fuel stations are green, and the goal state is shown in red. The agent's possible actions are shown in the middle. The fuel stations are the most interesting states to explore, as they vary in cost, while the center white states are easily predictable.

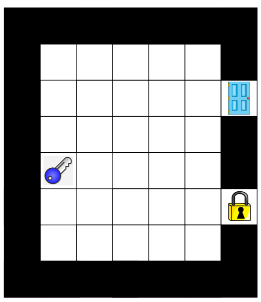

Figure 2: The Light World domain. In each room, the agent must navigate to the key, pickup the key, navigate to the lock, press it, and then navigate to and exit through the door to the next room.

receives a reward of -1 each step until it leaves the room, when it receives a reward of $+10$. The agent starts in a random state in the room. Since each of the objects has a related sensor feature, we hypothesize that a few different exploration strategies will work well on this task. Strategies that reward higher sensor features may help drive the agent to the correct objects, and strategies that utilize the NOVELTY reward have been shown to promote useful exploration as the objects have unique sensor values [11].

The third domain is called the **Increasing Rewards** domain, shown in Figure 3. In this task, the agent starts on the left side of the grid in one of the blue states. It can navigate through the grid with the usual actions: NORTH, SOUTH, EAST, and WEST, each of which move the agent in the desired direction with probability 0.8 and in either perpendicular direction with probability 0.1. There are 5 goals in the domain, and with escalating rewards for the farther goals. The agent receives -1 reward each step until it reaches a goal, when its episode terminates with the specified reward (either $1, 2, 4, 8$, or 16). The idea behind this domain is that the agent must set its exploration rewards high enough to drive it to explore past the closer lower-rewarding goals that are easier to find.

The last two domains are similar in nature, but the best exploration for each of them varies. In both domains, the agent is in a 11 by 12 gridworld with the same four navigation actions and action dynamics as the **Increasing Rewards** domain. However, in this task, there is a goal state that is in a *different* random location each episode. Essentially, each new episode is a new exploration problem for the agent. It can use what it has learned from past episodes about *which* exploration strategies are the best, but none of its knowledge about the locations of the goal in the previous episodes translate to the current episode. The agent receives a reward of -1 each step until reaching the goal state, when its episode terminates with a reward of $+2$. In the first version of the task, called the **Sensor Goal** task, the agent's state is made up of 6 state features: the agent's X and Y location in the domain, and four sensor features telling it the distance to the goal in each of the four cardinal directions. In this version of the task, both the strategies that reward minimizing these sensor features and the strategy rewarding novel states should be successful. In the second version of the domain, called **Arbitrary Goal**, the agent has no sensors of the goal's location, but instead has a state feature indicating the version of the domain it is in without providing any information about the goal location. In this version of the domain, the best exploration strategy is to visit every state in the domain until it finds the randomly located goal.

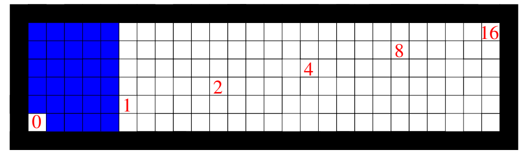

Figure 3: The Increasing Rewards domain. The agent starts in one of the blue states on the left side of the task. There are five goals in the domain, indicated by the red numbers. Goals that are farther away provide exponentially more rewards.

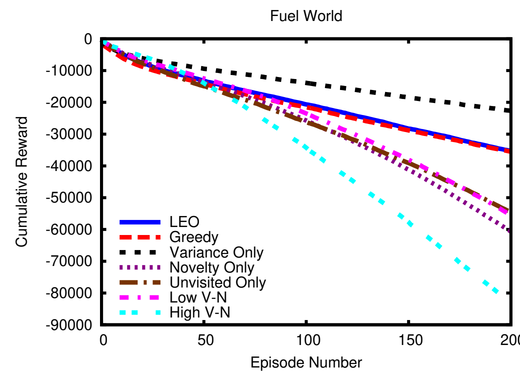

Figure 4: Cumulative rewards for the 7 methods on the Fuel World domain, averaged over 30 trials.

4.3 Results

In this section, we show the results for the algorithms across the five domains. All significance results were calculated using a Student's t-test. Figure 4 shows the cumulative rewards accrued by the algorithms over 200 episodes on the **Fuel World** domain. As expected for this task, the best method is the VARIANCE only method, which drives the agent to explore its various hypotheses about the costs of the fuel stations. The next best methods are LEO and Greedy. All three of these methods accrue significantly more rewards than the others ($p < 0.001$).

Figure 5 shows the weights LEO learned for the different

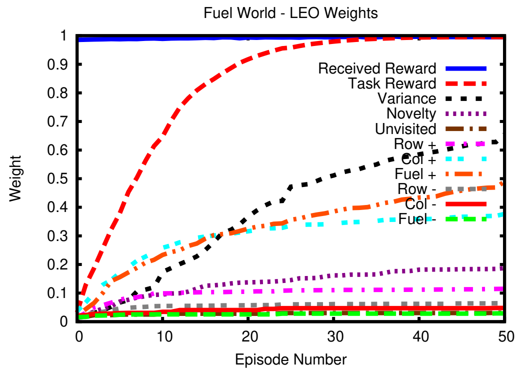

Figure 5: Weights learned by the LEO algorithm on the Fuel World domain, averaged over 30 trials.

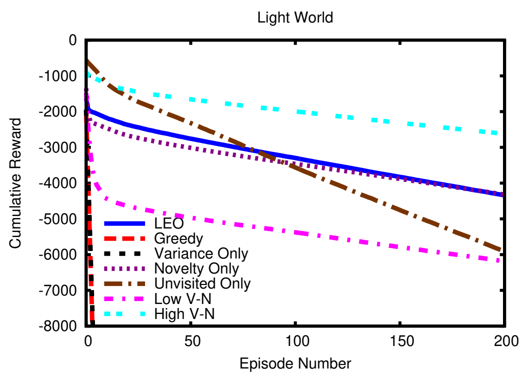

Figure 6: Cumulative rewards for the 7 methods on the Light World domain, averaged over 30 trials. LEO performs reasonably well on this task, while the best algorithms on Fuel World fail completely on this task.

strategies over the first 50 episodes. LEO learns the highest weight for the model of task reward. This is followed by the variance strategy, which makes sense as it performed the best on the domain. The third highest weight is on maximizing the FUEL feature, as LEO has learned to keep the fuel level high to accrue rewards. It also puts positive weight on the strategy of maximizing the COL feature, which will lead it closer to the goal from its start state.

The cumulative rewards of the algorithms on the second domain, **Light World**, are shown in Figure 6. On this task, the HIGH V-N parameters of TEXPLORE-VANIR performed the best, followed by NOVELTY only and LEO. While LEO does not perform the best on this task or **Fuel World**, comparing Figures 4 and 6 show that it is the only method to perform well on both domains. The two methods that performed similar to or better than LEO on **Fuel World** (VARIANCE only and Greedy) fail completely here, never learning to accomplish the task. Conversely, the two methods that perform similar to or better than LEO on **Light World** (NOVELTY only and HIGH V-N) perform the worst on **Fuel World**. The two domains require completely different exploration strategies, and only LEO is able to perform well on both tasks.

The cumulative rewards of the algorithms on the **Increasing Rewards** domain are shown in Figure 7. On this task, all of the algorithms perform well and accrue a similar amount of rewards. LEO does not accrue a significantly different amount of rewards than the other algorithms. Fig-

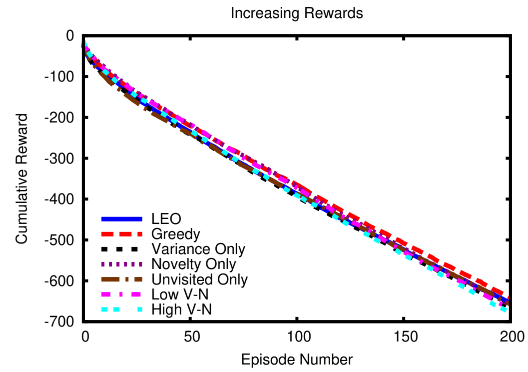

Figure 7: Cumulative rewards for the 7 methods on the Increasing Rewards domain, averaged over 30 trials.

ure 8 shows the weights learned by LEO on this task. LEO puts high weight on four strategies: 1) the task reward; 2) maximizing the COL feature; 3) novelty; and 4) variance. Maximizing the COL feature in this domain makes sense as the higher valued rewards are to the far right in the domain.

Figure 9 shows the cumulative rewards on the **Sensor Goal** domain. On this task, LEO performs the best, accruing significantly more rewards than the other algorithms ($p < 0.005$). Finally, cumulative rewards for the **Arbitrary Goal** domain are shown in Figure 10. As expected, on this task, the best strategy was to explore unvisited states to find the goal (the UNVISITED only strategy). Following this, the NOVELTY only and LOW V-N strategies did well, followed by LEO. While LEO is out-performed by these algorithms on this task, none of them did significantly better than LEO on the other four tasks.

In addition to cumulative rewards, a successful algorithm should learn good final policies. Table 1 shows the average rewards each method received on its final five episodes in each task, as well as how that average reward ranked compared with the other six methods for that task. LEO has the best average reward across the five domains, as it was the best on the **Sensor Goal** and **Increasing Rewards** tasks. LEO was only significantly out-performed by other algorithms on one domain, **Light World**, where the methods with NOVELTY rewards performed the best. These results demonstrate that LEO performs well across a set of domains requiring various exploration strategies. None of the other methods perform well across all five domains. Instead, performing well on these domains would require a user to hand-tune the exploration parameters for each domain. In contrast, LEO is more robust, not requiring hand-tuning and capable of learning the best exploration strategy for each domain. In addition, it can adapt its strategy parameters on-line as its model changes.

5. RELATED WORK

LEO's selection of each strategy according to its weight is similar to the bandit based approaches described in Section 2.1, as all of these methods are based on the EXP4 algorithm [2] for solving multi-armed bandit problems when given a set of experts. The EXP4 algorithm estimates the total return from each expert on the problem and uses a soft-max distribution over the total returns to select which expert to follow.

Domain	LEO		Greedy		VARIANCE only		NOVELTY only		UNVISITED only		LOW V-N only		HIGH V-N only	
	Reward	Rank	Reward	Rank	Reward	Rank	Reward	Rank	Reward	Rank	Reward	Rank	Reward	Rank
Fuel World	-127.4	3	-121.7	2	-86.1	1	-405.6^{*}	6	-308.1^{*}	4	-392.3^{*}	5	-481.8^{*}	7
Light World	-10.2	4	-1735.6^{*}	6	-1794.1^{*}	7	-8.9^{+}	3	-22.9^{*}	5	-8.1^{+}	2	-7.0^{+}	1
Increasing Rewards	-2.5	1	-2.7	3	-2.7	4	-3.0	7	-2.8	5	-2.9	6	-2.6	2
Sensor Goal	-53.1	1	-53.8	2	-98.2^{*}	3	-406.5^{*}	6	-140.3^{*}	4	-408.0^{*}	7	-159.2^{*}	5
Arbitrary Goal	-313.5	2	-538.4	5	-548.1	6	-401.7	4	-308.5	1	-323.7	3	-975.7^{*}	7
Average	-101.3	1	-490.4	6	-505.8	7	-245.1	4	-156.5	2	-227.0	3	-325.3	5

Table 1: This table shows the final performance achieved by each algorithm by presenting the reward each algorithm achieved on the final five episodes of each task, averaged over the 5 episodes and 30 trials. * indicates that LEO received significantly more rewards than this method ($p < 0.01$) and $^{+}$ indicates methods that received significantly more rewards than LEO ($p < 0.01$). The table also shows the rank of each average reward compared to the other methods for each task.

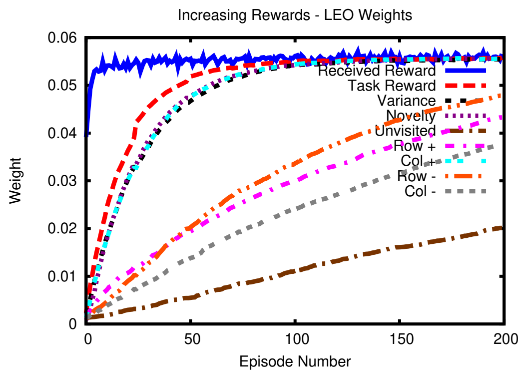

Figure 8: Weights learned by the LEO algorithm on the Increasing Rewards domain, averaged over 30 trials.

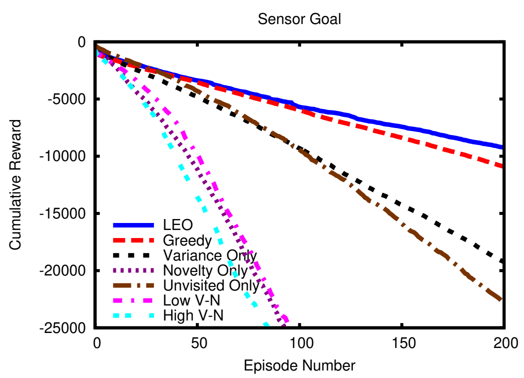

Figure 9: Cumulative rewards for the 7 methods on the Sensor Goal domain, averaged over 30 trials.

LEO is different from these previous works in several respects. After selecting a strategy, LEO takes a sequence of actions rather than a single arm pull, since it must complete an entire episode to receive a payoff. In addition, during an episode, LEO selects different strategies. Thus, contrary to other bandit methods, LEO updates the weights of *all* of the strategies using an importance sampling approach, rather than only updating the weight of a single strategy.

The work with the most similar goal to ours is the policy gradient reward design algorithm (PGRD). This method also learns the best intrinsic rewards on-line, however only for cases where the true reward function is given and the agent is *limited* in some way [6, 20]. PGRD uses its knowledge of the true reward function to calculate the gradient of intrinsic rewards to agent return. Using this gradient, intrinsic rewards

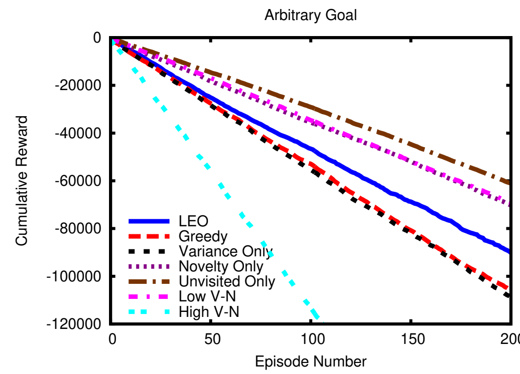

Figure 10: Cumulative rewards for the 7 methods on the Arbitrary Goal domain, averaged over 30 trials.

are found that enable the best agent performance given its limitations. For example, if the agent has a limited planning depth, then even with the true reward function, it cannot perform well. However, the right intrinsic rewards can make up for this deficiency. PGRD does not apply to agents without limitations, as then the given reward function means the problem is effectively solved. Similarly, in tasks where the reward function is not given, then the gradient cannot be calculated and this method does not work.

There are also other methods that are focused on exploration. For example, R-IAC [4] maintains an error curve for different regions of the domain and uses the slope of this curve as the intrinsic reward for the agent, driving the agent to explore the areas where its model is improving the most. An alternative is to learn a separate predictor of the change in model error and use its predicted values as the intrinsic reward to drive exploration [19]. Similarly, ZETA-R-MAX extends R-MAX to classify states as known based on the empirical measure of progress in model learning and provides convergence guarantees [15].

There is related work for doing off-policy value updates (i.e. calculating the value of policies other than the one being followed). Our method for calculating the value of the other exploration policies for each episode is similar to other off-policy approaches [21]. There are also methods that update the off-policy values every step by estimating their values for each state-action [17]. However, these methods would not apply to some of our experiments. For example, in the two domains with the goal in a random location each episode, none of the states visited in one episode will be seen again in the next episode. Therefore, learning off-policy values for those states will not be useful, but it is still possible to learn the values of a policy on a per-episode basis.

All of these methods for doing off-policy updates utilize *importance sampling*, which is a method for estimating one distribution while sampling from another distribution [18]. The importance sampling estimator is used to estimate the value of a target policy while following a different behavior policy. To do so, the received rewards are multiplied by the likelihood of the action under the target policy and divided by the likelihood under the followed policy. In our work, we compare between different exploration policies, and thus the likelihood under the followed policy is the same across all of these policies and cancels out.

6. DISCUSSION AND CONCLUSION

RL has the potential to solve many relevant sequential decision making processes. One difficulty with applying RL to these problems is the need to hand-tune exploration parameters. This tuning is required both to get the right type of exploration for the given domain and algorithm and to determine the proper exploration-exploitation trade-off for the task. In this paper, we have presented the novel LEO algorithm for learning from among several possible exploration strategies on-line. We demonstrated that this approach receives high rewards across a set of several domains that each require a different exploration strategy. In contrast, no single parameterized exploration approach performs well across all the domains. By learning exploration strategies on-line, LEO removes the need for users to hand-tune exploration parameters for each domain.

In this work, our goal is to maximize on-line rewards, and therefore we evaluate the quality of an exploration strategy based on the rewards received while following it. The received rewards indicate how quickly the exploration led the agent to find the rewarding transitions in the domain. While this approach works well in practice, it would be ideal to evaluate an exploration strategy based on the long-term rewards received *after* following it. One challenging possibility for future work is to separate exploration and exploitation, and evaluate exploration strategies by the agent's performance on a later evaluation episode where it exploits the model it learned while exploring. Another alternative is to evaluate the exploration strategies by how much they improve the agent's model accuracy, addressing the pure exploration problem. However, both of these alternatives have an off-line phase; we believe that the approach taken by LEO makes the most sense when the goal is to maximize on-line rewards.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin and at the FLOWERS group at INRIA Sud-Ouest. LARG research is supported in part by NSF (IIS-0917122), ONR (N00014-09-1-0658), and the FHWA (DTFH61-07-H-00030). Thanks to Pierre-Yves Oudeyer for valuable discussions on this work.

7. REFERENCES

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002.

[2] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(68), 2000.

[3] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. *JMLR*, 5:255–291, 2004.

[4] A. Baranes and P. Y. Oudeyer. R-IAC: Robust Intrinsically Motivated Exploration and Active Learning. *TAMD*, 1(3):155–169, Oct. 2009.

[5] R. Brafman and M. Tennenholtz. R-Max - a general polynomial time algorithm for near-optimal reinforcement learning. In *IJCAI*, 2001.

[6] J. Bratman, S. P. Singh, J. Sorg, and R. L. Lewis. Strong mitigation: nesting search for good policies within search for good reward. In *AAMAS*, 2012.

[7] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[8] D. Chakraborty and P. Stone. Structure learning in ergodic factored MDPs without knowledge of the transition function's in-degree. In *ICML*, June 2011.

[9] C. Diuk, L. Li, and B. Leffler. The adaptive-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *ICML*, 2009.

[10] T. Hester and P. Stone. Real time targeted exploration in large domains. In *ICDL*, August 2010.

[11] T. Hester and P. Stone. Intrinsically motivated model learning for a developing curious agent. In *ICDL*, November 2012.

[12] M. Hoffman, E. Brochu, and N. de Freitas. Portfolio allocation for bayesian optimization. In *UAI*, pages 327–336, 2011.

[13] J. Z. Kolter and A. Ng. Near-Bayesian exploration in polynomial time. In *ICML*, 2009.

[14] G. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, 2007.

[15] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *NIPS*, Tahoe, USA, 2012.

[16] M. Lopes and P.-Y. Oudeyer. The strategic student approach for life-long exploration and learning. In *ICDL*, November 2012.

[17] D. Precup, R. S. Sutton, and S. P. Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, 2000.

[18] R. Rubinstein. *Simulation and the Monte Carlo Methods*. John Wiley & Sons, Inc., 1981.

[19] J. Schmidhuber. Curious model-building control systems. In *International Joint Conference on Neural Networks*, 1991.

[20] J. Sorg, S. P. Singh, and R. L. Lewis. Optimal rewards versus leaf-evaluation heuristics in planning agents. In *AAAI Conference on Artificial Intelligence*, 2011.

[21] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.