# Efficient Exploration With Latent Structure

Bethany R. Leffler Michael L. Littman Alexander L. Strehl Thomas J. Walsh
Department of Computer Science
Rutgers University
Piscataway, NJ
Email: {bleffler,mlittman,strehl,thomaswa}@cs.rutgers.edu

***Abstract*— When interacting with a new environment, a robot can improve its online performance by efficiently exploring the effects of its actions. The efficiency of exploration can be expanded significantly by modeling and using latent structure to generalize experiences. We provide a theoretical development of the problem of exploration with latent structure, analyze several algorithms and prove matching lower bounds. We demonstrate our algorithmic ideas on a simple robot car repeatedly traversing a path with two different surface properties.**

## I. Introduction

In many successful applications of mobile robotics such as home vacuuming, scientific planetary exploration, and mapping, a robot has the opportunity to repeatedly visit a set of locations and thereby improve the efficacy of its actions in these locations. Learning effective behavior requires careful balancing of exploration and exploitation (Thrun 1992)—the decision maker needs to try new actions to evaluate their effectiveness, but at the same time, take advantage of what it knows to perform its task adequately.

There has been substantial progress over the past few years on developing provably efficient algorithms for balancing exploration and exploitation in the context of several formal mathematical models. Two notable concrete examples of such formal problems are bandit problems (Berry and Fristedt 1985) and learning in Markov decision processes (MDPs) (Sutton and Barto 1998). We summarize in Section II a notion of efficient exploration and describe what is known about algorithms for achieving efficiency in these two models.

A significant difficulty in applying the known algorithms to problems in robotics or other real-life instantiations is that they fail to exploit commonality of structure within the domain, which is absolutely critical to generalization and rapid learning. As a motivating example, consider a robot learning to traverse an office hallway looking for an unlocked door. Each location contains either a door or a wall. A sensible robot will not check to see if every segment of wall is "unlocked"—once it can distinguish walls and doors, it only needs to check the doors. Said another way, the learner can establish an unobserved, or latent, variable that indicates whether a location includes a door and this additional information can constrain exploration and model building resulting in faster learning. Existing efficient exploration algorithms do not have the capability of generalizing their experience in this way.

In Section III, we present a formal model that can serve as a first step to understanding the problem of efficient exploration with latent structure. We also provide an algorithm that is provably efficient in the context of the formal model. Section IV details our implementation of several exploration algorithms in a simple mobile robot domain to allow for direct comparison and to illustrate feasibility.

## II. Probably Approximately Optimal Exploration

In this section, we describe two well-studied formal models of exploration, define a mathematical concept of computational efficiency of learning in these models, and summarize known results of exploration algorithms for the models.

### A. Bandit problems

Bandit problems (Berry and Fristedt 1985) can be defined as follows. A learner is faced with $k$ actions, or "arms". On each of a sequence of discrete time steps $t = 1, \ldots$, the learner must choose an action to execute, $i_t$. After executing its action, the learner observes a payoff $r_t$ in the range $0$ to $B$. Payoffs are drawn from a stationary distribution that is a function of the selected action where $\mu_i$ is the expected payoff from action $i$. The learner attempts to select actions to maximize its total payoff.

Although algorithms have been defined that can achieve optimal payoff in some types of bandit problems (Gittins 1989), Fong (1995) advocated a weaker objective for learning. In *probably approximately optimal* (PAO) learning, the learner's behavior is parameterized by two values, $\epsilon > 0$ and $\delta > 0$. The learner must identify an action $\hat{\imath}$ that is *probably* (with probability at least $1-\delta$) *approximately* optimal ($\mu_{\hat{\imath}} \geq \mu^* - \epsilon$, where $\mu^* = \max_i \mu_i$). For an algorithm for bandit problems to be considered *efficient*, the PAO property must be achieved after $m$ timesteps, where $m$ is polynomial in $k$, $B$, $1/\epsilon$ and $1/\delta$.

Fong (1995) showed that several algorithms are efficient. The simplest, which he called "naïve", computes a number $m$, polynomial in the required quantities (Section III-C), then executes each action $m/k$ times. The $m$ trials are used to compute the maximum likelihood estimate of each $\mu_i$ (just the empirical average of the payoffs for action $i$) and the action with the highest estimate is PAO. Even-Dar *et al.* (2002) also presented an analysis for naïve, as well as for an improvement they called "median elimination".

The interval estimation (IE) algorithm (Kaelbling 1993) uses the available experience to put $\beta$% confidence intervals on the estimated payoffs of each action. It then chooses the action

with the highest upper confidence, resulting in the choice of either a high-payoff action, or one that results in a substantial reduction of the size of a confidence interval. Fong (1995) showed how to compute $\beta$ to create an efficient algorithm.

In greedy exploration, the empirical average payoffs for each action are computed. The action with the highest estimate is chosen. This approach is *not* PAO because an unlucky trial for a high-payoff action can result in it getting an artificially low estimate and then forever be starved of experience in favor of a lower payoff choice. A simple variation, $\epsilon$-greedy exploration, mitigates the likelihood of this premature convergence by selecting the greedy action with probability $1-\epsilon$ and choosing a uniform random action with the remaining probability. By setting $\epsilon$ appropriately (Strehl and Littman 2004), an $\epsilon$-greedy learner can sample actions often enough to avoid premature convergence and achieve efficient learning.

Of the various algorithms we have evaluated in detail, IE seems best at identifying high payoff actions quickly while avoiding premature convergence, while naïve is the simplest to analyze. In this paper, we build on the analysis of naïve, although we believe more effective exploration techniques will prove superior in future work.

### B. Markov decision processes

In a Markov decision process (MDP) (Puterman 1994), a learner again chooses a series of actions and receives a series of payoffs. In an MDP, however, the payoff process has a notion of *state* that influences the expected payoff of the actions. In particular, before each action choice, the learner is informed of its current state from a set of $l$ possible states. The payoff at time $t$ is sampled from a stationary distribution that is a function of the state at time $t$ and the learner's choice of action at time $t$.

The exploration problem in MDPs, studied in the reinforcement-learning literature (Sutton and Barto 1998), is complicated by the fact that the state evolves according to a stationary transition function that is also a function of the most recent state and action. This means that the most straightforward extension of naïve—try each action in each state $m$ times—is unimplementable because the learner has only limited ability to reach any specific state. Furthermore, extensions of algorithms like $\epsilon$-greedy run into problems as well—randomly chosen actions can result in inopportune transitions to low-payoff states.

Fiechter (1994) described a set of PAO conditions for MDPs and showed that they reduce to the same concept. $E^3$ (Kearns and Singh 2002) (for "explicit explore exploit") was the first algorithm to achieve efficient learning in MDPs. It keeps track of the number of times each action was executed in each state to determine when the estimates associated with the state are sufficiently accurate. The resulting empirical model is used to calculate the expected payoff for behaving greedily and the expected cost in time for reaching an underexplored part of the state space. The algorithm chooses its behavior based on a comparison of these two strategies.

RMAX (Brafman and Tennenholtz 2002) is a simpler algorithm that combines exploration and exploitation into a single decision by assigning underexplored states an artificially high payoff value in the empirical model. Strehl and Littman (2004) point out that RMAX for MDPs is analogous to naïve for bandit problems in that it distinguishes underexplored actions from sufficiently explored actions on the basis of the number of times they have been executed and always prefers underexplored actions when they exist.

Model-based interval estimation (MBIE) (Strehl and Littman 2004) is the analog of IE for MDPs. It creates high-confidence payoff bounds on rewards based on interaction experience and chooses actions that correspond to the upper confidence interval. Several other authors have described algorithms based on similar ideas (Dearden *et al.* 1999; Even-Dar *et al.* 2003; Wiering and Schmidhuber 1998), but only MBIE has been proven efficient.

## III. Efficient Exploration with Latent Structure

Provably efficient exploration algorithms for bandit problems and MDPs have been implemented and several of these algorithms result in practical learning rates in small problems. As mentioned in the introduction, scaling to real-life problems requires understanding how to retain theoretical assurances while taking advantage of the structure these problems offer for generalization.

In this section, we introduce a simple formal model that represents a small step in this direction. In particular, we propose a model that includes latent structure and provide an algorithm that exploits this structure to generalize and learn more efficiently than is possible by an algorithm that does not generalize. The result is strong in the sense that we can prove lower bounds on the experience needed to achieve PAO learning as well as algorithms that meet these bounds (to within constant factors).

Before introducing the model, we briefly mention another attempt at using structure to improve exploration. In a dynamic Bayes network representation of an MDP, each state is described by a set of features and, for each action, the conditional independence of features is explicitly represented. Factored $E^3$ (Kearns and Koller 1999) learns efficiently by modeling payoffs and transitions in terms of these conditional independence relations. Note that the algorithm requires that the conditional independence relations be known in advance. Thus, while factored $E^3$ demonstrates that structure can be leveraged to speed up exploration, the structure is provided explicitly as part of the problem description and is not *latent*, as is the case in most naturally occurring problems. Learning the structure *and* exploiting it is the goal of this paper.

The idea of modeling latent structure in a learning problem was also explored by Sherstov and Stone (2005). The learning setting in that work was an MDP, a more general model than considered here. Their goal was not to speed up learning on a single domain but to facilitate the transfer of learning to similar domains. Our use of abstraction across states is

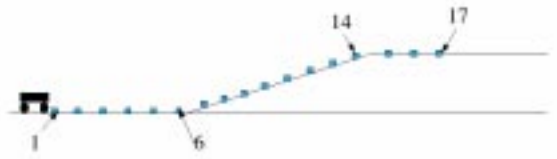


Fig. 1. A schematic view of the robotic testbed used in our experiments.

similar to techniques studied in hierarchical reinforcement learning (Barto and Mahadevan 2003).

### A. A simple learning problem with latent structure

To study problems with latent structure, we created a physical testbed. It consists of a simple robot car that can move forward at any of seven different power settings. On the course depicted in Figure 1, the car moves from Location 1 to Location 17, recording its speed at each location using dead reckoning. Due to the gear structure of the robot, momentum has no influence on velocity across locations. Since the car's sensor is limited to its speed, its objective is to retain a constant goal speed $g$, with per-step reward decreasing with the square of the deviation from this speed. The ideal policy for the robot is to choose a power setting at each location that maximizes its reward.

Notice that each of the locations on the course fall into one of two classes, either flat or sloped. An algorithm that accounts for this latent structure will be able to use its data to generalize its behavior well and thereby solve the learning task more efficiently than if this structure were neglected.

More formally, our example problem lies between bandit problems and MDPs in complexity. The environment can be expressed as a set of $l$ locations (states), numbered 1 through $l$. The locations are arranged in a repeating loop; after each action, the process transitions from its current location $j$ to the next location in the loop $(j+1) \bmod l$, regardless of the action chosen. Like MDPs, the combination of the action and location result in a probability distribution over outcomes (speed, on our robot), and outcomes are associated with rewards. In this paper, we examine outcome distributions over a finite set $W$ of $a := |W|$ possibilities. A parallel theory can be developed for continuous outcome distributions such as Gaussians. Also similar to the MDP case, the learner is informed of the identity of the current location before each action decision.

The latent structure in this problem can be described as follows. There is a set of $n << l$ classes that mediate between the locations and the outcome distributions. That is, each location is assigned a class and each class is associated with a stationary outcome distribution. We use the notation $A_j$ to be the class assignment for location $j$ and $\mu_i^c$ to be the expected payoff for action $i$ in class $c$ (averaged over all outcomes). Thus, selecting action $i$ in location $j$ results in a payoff of $\mu_i^{A_j}$, on average.

We restrict the payoffs to the range 0 to $B$, which we take as a constant, and translate outcomes to rewards using the deterministic function $R(w)$. We write $\rho_i^c(w)$ to be the probability of outcome $w \in W$ resulting from the selection of action $i$ in a location of class $c$. Thus, $\mu_i^c = \sum_w R(w) \times \rho_i^c(w)$.

We make an additional *separability assumption*. We define the L1 distance between outcome distributions for two classes $c_1$ and $c_2$ under the same action $i$ to be

$$||\rho_i^{c_1} - \rho_i^{c_2}||_1 = \sum_w |\rho_i^{c_1}(w) - \rho_i^{c_2}(w)|.$$

The separability assumption states that there exists a constant $\Delta > 0$ such that, for all actions $i$, and classes $c_1$, $c_2$ ($c_1 \neq c_2$), $||\rho_i^{c_1} - \rho_i^{c_2}||_1 > \Delta$. That is, outcome distributions for different classes are well separated.

The two critical pieces of missing information in the model just described are the class–location assignments $A_j$ and the outcome distributions $\rho_i^c(w)$. Although we are most interested in a learning algorithm for the case in which both of these pieces of information are missing, the next sections examine the learning problems resulting from various simplifications. The relationships between these problems help put the full learning problem in its proper context.

### B. Known assignment, known outcomes

When both $A$ and $\rho$ are known, the optimal choice of action is simply to compute $\mu_i^c = \sum_w R(w) \times \rho_i^c(w)$ for each action $i$ in each class $c$. Then, when occupying location $j$, choose $i^* = \text{argmax}_i \, \mu_i^{A_j}$, that is, the action $i^*$ that has the highest expected reward for the class location $j$ belongs to. No learning is needed and the optimization, which we refer to as "known-known" (known assignment, known outcomes), is quite simple.

### C. Known assignment, unknown outcomes

Next, let's consider the problem that results from the assignment function $A$ being known, but the outcome distributions $\rho$ remaining unknown. The learner always knows the class of the current location, but it initially does not know the appropriate action to take in the class. The classes don't provide any information about one another, so the learner actually faces $n$ separate bandit problems, one for each class, which we can solve independently.

The naïve method can be applied to this problem by treating each class as a separate bandit problem. We call this algorithm "known-naïve", since the assignments are known and the unknown outcomes are learned using naïve. By the analysis of Fong (1995), it suffices to try each action $i$ in a fixed class $c$ $O(\frac{1}{\epsilon^2} \ln(\frac{nk}{\delta}))$ times. After this exploration phase, known-naïve estimates the average reward $\hat{\mu}_i^c$ by the sample mean for each action $i$, and chooses the one that appears to achieve the greatest reward, $i^* = \text{argmax}_{i=1}^k \hat{\mu}_i^c$. Fong (1995) has shown that $i^*$ will be an $\epsilon$-optimal arm with probability at least $1 - \frac{\delta}{n}$. The algorithm then proceeds by sampling each action in each class as described.

For simplicity of presentation, we use the notation $\hat{O}(\cdot)$ to represent $O(\cdot)$ where $\epsilon$, $\delta$, $B$, and $a$ are taken as constants.

We also define $\hat{\Omega}(\cdot)$ similarly. These definitions emphasize the dependence of the bounds on the number of locations $l$, classes $n$, and actions $k$. Known-naïve has a total learning complexity $\hat{O}(nk\ln(nk))$ as it samples each action in locations representing each class. It is guaranteed to find an $\epsilon$-optimal action for each class (thus for each location) with probability at least $1-\delta$, which follows from the union bound over all classes and actions.

Section VI shows that $\hat{\Omega}(nk\ln(n))$ samples are needed to achieve the PAO condition for this problem. Thus, known-naïve includes an unnecessary factor of $\ln(k)$ in the bound. Even-Dar *et al.* (2002) provides an algorithm called "median elimination" that removes the dependence on $\ln(k)$. Thus, the "known-median" algorithm matches the lower bound.

In our robot task we used known-naïve because it's simpler and since the number of actions is small making the removal of $k$ from the logarithmic factor unnecessary.

### D. Unknown assignment, known outcomes

This case represents the inverse of the learning problem from the previous section. We take $\rho$ to be known and $A$ to be unknown. That is, although we know how each class behaves, we don't know which locations correspond to which classes.

Note that after selecting action $i$ $m$ times in class $c$, the learner can estimate a vector of length $a$, $\hat{\rho}_i^c$, where $\hat{\rho}_i^c(w)$ is the number of times we have taken action $i$ in class $c$ and observed outcome $w$, divided by $m$. From the results of Weissman *et al.* (2003), we have that

$$\Pr(||\rho_i^c - \hat{\rho}_i^c||_1 \geq \epsilon) \leq 2^a e^{\frac{-m\epsilon^2}{2}}. \quad (1)$$

Setting Equation 1 to be less than $\delta$ yields

$$m \geq \left(\frac{2}{\epsilon^2}\right) \left[a\ln(2) + \ln\left(\frac{1}{\delta}\right)\right]. \quad (2)$$

Because of the separability assumption, to determine the unknown assignment, we only need to sample, at each location, from one action.[1] Since the outcome distributions $\rho$ are known in this case, accurately estimating the outcome distribution for an action reveals its class.

We can choose the action arbitrarily, and one reasonable choice is the action that maximizes expected return, given some prior over the set of classes. If we gather enough samples at any given position to assure that the difference between the empirical outcome distribution and the true outcome distribution is less than $\frac{\Delta}{2}$, then by our separability assumption, we can assign a position to the class whose distribution is closest in L1 distance to the empirical distribution, which is typically called a nearest neighbor approach. We call this algorithm "NN-known" as it uses the nearest-neighbor approach to identify the class of each location and then uses its knowledge of the distributions to select actions.

[1]Weaker assumptions can also be made that still lead to efficient algorithms. However, completely eliminating the separability assumption makes it no easier, in a $\hat{O}(\cdot)$ sense, to solve the problem than if there were no structure at all.

If we set $\epsilon := \frac{\Delta}{2}$ and substitute $\frac{\delta}{l}$ for $\delta$ in Equation 2, we can compute the number of samples for each location to ensure that with probability at least $1-\delta$ we will determine the correct mapping from location to class. Once this is complete, we can immediately implement the optimal policy, since for each class we can compute the action with highest expected reward.

Using the above analysis, we discover that our goal can be accomplished using $\hat{O}(l\ln l)$ samples. Note that the dependence on the number of actions $k$ disappears because only one action needs to be sampled to reliably identify the class. Once again, Section VI argues that this result matches the lower bound for this problem.

### E. Unknown assignment, unknown outcomes

If both assignments and outcome distributions are unknown, one approach that can be taken is to treat each location as an independent bandit problem. This "unmapped-naïve" algorithm, which ignores the class assignments entirely, achieves a bound of $\hat{O}(lk\ln(lk))$. The analogous "unmapped-median" algorithm uses $\hat{O}(lk\ln(l))$ samples and achieves the lower bound for the structure-free case.

An approach that exploits the latent structure of the problem should be able to achieve a smaller bound. The problem in which both assignments and outcome distributions are unknown is at least as hard as the problem considered in Sections III-C and III-D. This suggests that we would be fortunate if we could learn a PAO policy using $\hat{O}(l\ln l + nk\ln n)$ steps of experience—the maximum of the lower bounds derived for the two previous problems. In fact, we can achieve this bound, as described below. Since $n << l$ (far fewer classes than locations), this algorithm is exploiting the latent structure in a manner that improves learning time.

A direct approach to the problem is to learn the unknown assignments and the unknown outcome distributions in two separate phases. In the first phase, locations are grouped by the similarity of their outcome distributions, building on the insights from Section III-D. Since the true outcome distributions are unknown, we cannot use the nearest-neighbor approach, and instead must cluster based only on the relative distances of the empirical distributions. Once an assignment of locations to class is complete, we can use the known-median approach from Section III-C to learn the outcome distributions with sufficient accuracy to make near-optimal decisions.

First, let's consider the clustering step. For any fixed action, we seek to build an estimate of its outcome distribution for each location. If all $l$ estimates are accurate to within an L1 distance of $\epsilon$, we know that the distributions from two locations that are part of the same class cannot be more than an L1 distance of $2\epsilon$ apart (by the triangle inequality). On the other hand, two locations that are part of separate classes will have estimates that are at most $\Delta - 2\epsilon$ apart in L1 distance, since the underlying distributions must be $\Delta$ apart in L1 distance by the separability assumption. We want to use a value of $\epsilon$ so that we'll be able to distinguish locations that belong to the same class from those that do not: $\Delta - 2\epsilon > 2\epsilon$, or $\epsilon < \Delta/4$. Using

Equation 1 and the union bound, we find that $\hat{O}(l \ln l)$ steps of experience will suffice to get accurate estimates with high probability. Specifically, we require that the failure probability, the chance that some empirical distribution differs in L1 distance from its true distribution by more than $\epsilon$, is less than $\frac{\delta}{2}$.

After these distributions have been estimated, they can be used to create a distance matrix and clustered by, for example, single-link hierarchical clustering. If the clustering is stopped when distances exceed $\Delta/2$ or when $n$ clusters have been formed, the resulting clustering is correct with probability at least $1 - \frac{\delta}{2}$. Thus, each location is assigned to "the right" class—one that matches the other locations that are in the same true class.

Next, we can directly apply the known-median approach, given that we have correctly reconstructed the assignment of classes to locations. Again, only $\hat{O}(nk \ln n)$ steps of experience are necessary (Section III-C), so combining the learning from the two phases, a total of $\hat{O}(l \ln l + nk \ln n)$ learning samples are needed, matching our desired result. Again, we require the failure probability be less than $\frac{\delta}{2}$. Thus, using a union bound, the probability of a failure in either one of the two phases is less than $\delta$. Hence, for each location, an $\epsilon$-optimal policy will be found with probability at least $1 - \delta$, and the sample complexity will not exceed a polynomial in the relevant quantities.

It should be noted that the constants absorbed in the $\hat{O}(\cdot)$ notation are larger for this combined approach then for solving the two problems of Sections III-C and III-D independently. This is due to the fact that, in the first step, we require each empirical distribution to be within $\frac{\Delta}{4}$ of the true distribution, in L1 distance, rather than $\frac{\Delta}{2}$, as in the case where the agent is aware of the true distributions. It is also due to the fact that we must divide $\delta$ by two in each of two stages of the algorithm. However, ignoring constants, the result is the same. In our experiments, we call this algorithm, "cluster-median", again noting that, do to its simplicity, we evaluated "cluster-naïve" (with single-link clustering) in our experiments.

## IV. Exploration Experiments

This section presents a physical implementation and evaluation of the algorithms and testbed previously described.

### A. Algorithms Tested

We experimented with different algorithms for solving each of the four classes of problems outlined in Section III. Note that while it is possible to use the separability assumption to compute near-optimal sample sizes, in our experiments we chose sample sizes empirically.

In the "known assignment, known outcomes" case (Section III-B), 14 traversals were performed "offline" to estimate model parameters. Locations were labeled with classes by hand based on their slope, and these class labels were verified against the known outcomes.

Next, for the "known assignment, unknown outcomes" case (Section III-C), location classes were again labeled by hand and then fed to the known-naïve algorithm to record the outcomes and decide on a policy. Estimates for the outcome distributions were compiled using a single traversal, which is sufficiently long to sample each power level in each class and produced near-optimal behavior.

For the "unknown assignment, known outcomes" case (Section III-D), we evaluated two different algorithms. A strategy we call "POMDP-known" used knowledge of the outcome distributions to build a Partially Observable Markov Decision Process (POMDP) for choosing power levels at a location given a 50-50 prior over the two classes. The procedure employed Tony Cassandra's implementation of incremental pruning (Cassandra *et al.* 1997) to calculate a policy for choosing actions for each timestep to maximize the overall expected reward over a large finite horizon. Although this algorithm can require exponential time, it ran quickly and produced a simple policy consisting of an initial intermediate power level followed by a high or low power depending on the speed observed the first time the location was visited.

The second algorithm we applied in this case was the NN-known procedure. The true distributions of outcomes were mined from the same data used to solve the "known assignments, known outcomes" case. Three traversals were then performed at a single power level to record empirical outcome distributions. NN-known was then used to compare the two distributions and map the locations associated with each outcome triple to the appropriate class. Having discerned the classes, the robot executed the optimal action, as computed from the data given to it.

Finally, we implemented three algorithms for the "unknown assignment, unknown outcomes" case (Section III-E). The first, unmapped-naïve, simply ignored the latent structure in the problem and ran each action in each location once, thereafter choosing the action that had the best performance for each location. After one traversal per action, the resulting estimates were good enough to achieve apparently optimal performance.

The second algorithm we considered used Expectation-Maximization (EM) (Dempster *et al.* 1977) as the basis for classifying locations. While there are some variations of EM for which formal guarantees have been proven, we used EM in this study as an *ad hoc* method for estimating latent structure. Our EM implementation classified each of the 17 locations into one of the two classes given three samples of the outcome distributions for each location using a constant action throughout the trials. At this point, learning proceeded as in the "known assignment, unknown outcomes" setting (Section III-C), resulting in the algorithm "EM-naive".

Finally, we evaluated the performance of the clustering algorithm, cluster-naïve. Data was collected in a series of three traversals, just as in the EM implementation.

### B. Experimental setup

To compare the algorithms in a physical setting, we built a robot to traverse our two-slope test course. The robot was constructed using parts from a Lego Mindstorm kit, specifically

Fig. 2. The robotic vehicle used in our experiments.

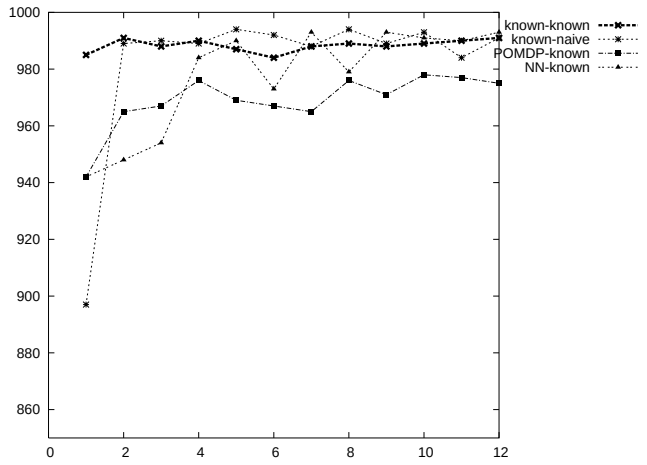


Fig. 3. Reward for each traversal for algorithms that were provided some prior knowledge of the domain.

an RCX 2.0 block, one rotation sensor, one motor with seven power levels, four wheels, and various connecting pieces. The course was made up of three two-by-two boards supported by plywood in the pattern shown in Figure 1. The robot's software was built in the the leJOS programming language. The robot itself is shown in Figure 2.

Each algorithm was run for two epochs, consisting of 12 traversals each. The interval between locations was defined as 100 rotation clicks as registered by the robot's rotation sensor. The reward for a traversal was calculated using the following formula:

$$1000 - \sum_{l} (round(\frac{t_l - t_{l-1}}{100}) - t_g)^2 \qquad (3)$$

where $round$ and division by 100 are used to discretize the outcomes, $t_g$ is the goal time elapsed between points as derived from the goal speed $g$, $t_l$ is the time we reach location $l$ and the subtraction from 1000 is used to map cost to reward.

Since the RCX 2.0 does not possess adequate memory to perform the calculations needed for the aforementioned algorithms, the robot simply stored the observed speeds during each traversal. At the end of the course, the collected data was transmitted to a laptop, which processed the data and calculated the power levels to use for each location in the next traversal as per the algorithm being evaluated.

Evaluating algorithms in the real world invites a host of noise factors that one would not consider in a pure simulation. One such factor in our implementation was the substantial effect battery power had on the robot's performance. As the batteries drained, the effect of the power commands on the movement of the robot changed. Although the learning algorithms were able to adapt to different battery powers, the large variability of the action effects threatened to make comparisons unreasonable. To compensate, brand new disposable batteries were used for each algorithm studied. Thereafter, any decline in battery power appeared consistent across all algorithms.

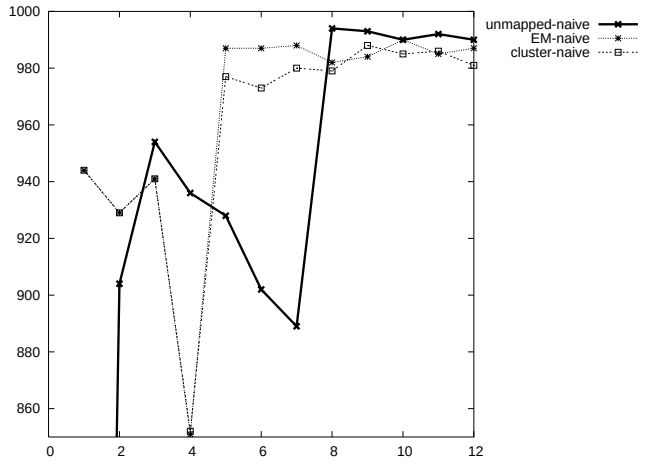


Fig. 4. Reward for each traversal for algorithms in the unknown-unknown case.

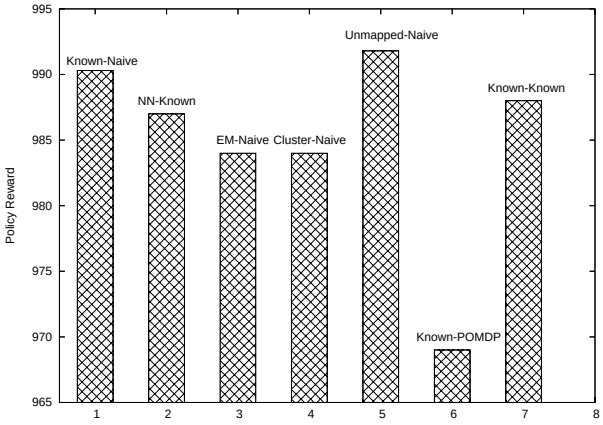


Fig. 5. Average reward obtained by the learned policy for each algorithm.

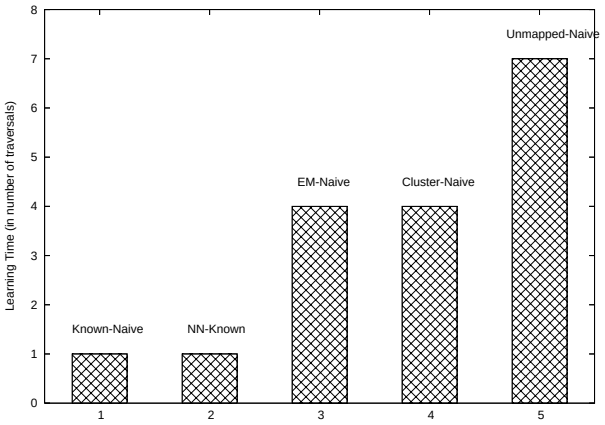


Fig. 6. Traversals needed to learn a near-optimal policy.

### C. Results

Figure 3 and Figure 4 show the total reward each algorithm accrued in each traversal of the first epoch. The graphs for the second epoch were omitted because they did not differ significantly from Epoch 1. Each algorithm learns for some number of traversals, then identifies a final policy that it uses for the remainder of the epoch. The small variations noticeable after learning are due only to noise.

As expected, the known-known strategy performed well in every traversal since no learning was required. Along the same lines, all partial-knowledge algorithms learned a good policy sooner than any of the "unknown assignment, unknown outcomes" algorithms. When comparing the three"unknown assignment, unknown outcome" algorithms, the EM-naïve and cluster-naïve algorithms decided on a policy in almost half the time of the unmapped-naïve algorithm. The convergence times for these three algorithms is displayed with greater granularity in Figure 6. Notice that the learned policies have roughly the same values, except POMDP-known. In fact, the EM-naïve and cluster-naïve implementations converged to the exact same policies. The poor performance of POMDP-known is attributable to the model of the environment it built internally, which was inconsistent with the models used by the remaining algorithms. Figure 5 compares the average reward for all resulting behaviors.

Surprisingly, the known-known strategy does not perform the best. This is because the assumption that only two classes exist in the world (uphill and flat) is not entirely true. In actuality, each position was slightly different, and certain locations were really transitions between the flat and sloped surfaces and needed to be handled differently. So, even though the algorithms that exploit the latent structure quickly discover which positions belong to which classes, they fail to achieve maximum reward. The unmapped-naïve approach was able to represent and exploit the *differences* between the locations, resulting in slower learning but larger reward after learning.

## V. Conclusion

Learning in real-life scenarios is very difficult because of the complexity of natural environments. Successful learners must be prepared to extract and exploit latent structure in their environments to learn efficiently. To study this issue, we have presented a particular learning scenario with latent structure. We showed that this structure can be exploited to improve the rate of learning and demonstrated this fact with (i) algorithms with improved bounds, (ii) matching lower bounds that show the fundamental complexity of the problem, and (iii) empirical demonstration on an implemented robot with improved rates of learning.

A specific technical contribution of the paper is our two-phase procedure for solving the learning problem we posed. It makes productive use of unsupervised clustering to reveal the latent structure, then exploits the structure to learn efficiently. We showed that the cluster-median version of this approach matches the lower bound for the problem, showing that it exploits the latent structure to the maximum degree possible for this problem class.

Natural extensions of our work include learning with an unknown number of classes, developing algorithms for related problems like learning the conditional independences in the dynamic Bayes network setting or learning an abstract model of an MDP, combining the two learning phases so that the problem as a whole may be explored efficiently and jointly, and weakening the separability assumption so that it need not hold for *all* actions.

## VI. Appendix: Algorithm Analysis

The first problem we consider is solving $n$ independent $k$-action bandit problems. Unfortunately, $n$ copies of a PAO bandit algorithm that fails with probability $\delta$ will fail with probability $1-(1-\delta)^n$, making it no longer PAO.

Even-Dar *et al.* (2002) showed that in the PAO model, given $\epsilon$, $\delta$, any algorithm that finds an $\epsilon$-optimal policy with probability at least $1-\delta$ will require at least $O(\frac{k}{\epsilon^2}\ \ln\left(\frac{1}{\delta}\right))$ samples. Thus, any PAO algorithm requires at least $C\ln\left(\frac{1}{\delta}\right)$ samples for some constant $C$, where we have taken $k$, the number of actions, and $\epsilon$ as parameters folded into $C$, since they have no relevance on our analysis.

Now, suppose we have $n$ separate bandit sub-problems to solve. Each sub-problem $i$ has an associated failure probability $\delta_i$. The goal of the overall algorithm is to find an $\epsilon$-optimal solution for each of the subproblems and then bound the event that any of the subproblems fails by $\delta$. Thus, it will require at least

$$C\sum_{i=1}^{n}\ln\left(\frac{1}{\delta_i}\right) \tag{4}$$

samples. We will bound Equation 4 from below, under the constraint that the probability that some sub-problem fails is equal to the overall failure bound:

$$\delta = 1-\prod_{i=1}^{n}(1-\delta_i). \tag{5}$$

First, we sketch an argument that Equation 4 is minimized with all $\delta_i$ values equal to

$$\delta_1 = 1 - (1-\delta)^{(1/n)}.$$

This choice is justified by the fact that, if $\delta_i$ and $\delta_j$ differ, setting them equal to $1 - \sqrt{(1-\delta_i)(1-\delta_j)}$ satisfies Equation 5 and reduces the value of Equation 4. The latter follows from a close examination of the contribution of $\delta_i$ and $\delta_j$ to the function and showing that the minimum value is achieved when $\delta_i = \delta_j$.

We can then show that the definition of $\delta_i$ leads to the desired bound. Define

$$S_1 = n \cdot \ln\left(\frac{1}{1-(1-\delta)^{(1/n)}}\right)$$

and

$$S_2 = n \cdot \ln\left(\frac{n}{\delta}\right).$$

$S_1$ is the bound that comes from Equation 4. $S_2$ is the desired lower bound. With several applications of l'Hospital's rule, it can be shown that

$$\lim_{n\to\infty} \frac{S_1}{S_2} = 1.$$

Thus, the number of samples needed to solve $n$ independent bandit problems in the PAO framework has a lower bound of $\Omega(n\ln\left(\frac{n}{\delta}\right))$, once again ignoring $\epsilon$ and $k$.

Even-Dar *et al.* (2002) provide another useful bound, which we use to prove a lower bound for the problem considered in Section III-D. The bound involves the *coin-bias problem* in which a weighted coin has a probability of heads of either $0.5+\epsilon$ or $0.5-\epsilon$. The task is to determine the coin's bias with probability at least $1-\delta$ given $0<\epsilon<1$ and $0<\delta<1$ as input.

The number of samples required by any algorithm that solves the coin-bias problem (Mannor and Tsitsiklis 2004) is

$$\Omega\left(\frac{1}{\epsilon^2}\ln\left(\frac{1}{\delta}\right)\right). \tag{6}$$

The coin-bias problem is a special case of the unknown assignment problem, specifically where there is a single location and two classes with distributions identical to the distributions that govern the behavior of the two weighted coins. Thus, the lower bound in Equation 6 applies to the location-assignment problem.

Using the argument from the beginning of this section, solving $l$ copies of the coin-bias problem, one for each location, leads to an overall lower bound of $\Omega(\frac{l}{\epsilon^2}\ln\left(\frac{l}{\delta}\right))$. Ignoring constants and taking $a$ (the number of possible outcomes) to be constant, the algorithm NN-known from Section III-D achieves this lower bound (see Equation 2).

## REFERENCES


[Barto and Mahadevan, 2003] Andrew Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems*, 13, Special Issue on Reinforcement Learning:41–77, 2003.

[Berry and Fristedt, 1985] Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Chapman and Hall, London, UK, 1985.

[Brafman and Tennenholtz, 2002] Ronen I. Brafman and Moshe Tennenholtz. R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[Cassandra *et al.*, 1997] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 54–61, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

[Dearden *et al.*, 1999] Richard Dearden, Nir Friedman, and David Andre. Model-based bayesian exploration. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 150–159, 1999.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[Even-Dar *et al.*, 2002] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Pac bounds for multi-armed bandit and Markov decision processes. In *15th Annual Conference on Computational Learning Theory (COLT)*, pages 255–270, 2002.

[Even-Dar *et al.*, 2003] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for reinforcement learning. In *The Twentieth International Conference on Machine Learning (ICML 2003)*, pages 162–169, 2003.

[Fiechter, 1994] Claude-Nicolas Fiechter. Efficient reinforcement learning. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 88–97. Association of Computing Machinery, 1994.

[Fong, 1995] Philip W. L. Fong. A quantitative study of hypothesis selection. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 226–234, 1995.

[Gittins, 1989] J. C. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley-Interscience series in systems and optimization. Wiley, Chichester, NY, 1989.

[Kaelbling, 1993] Leslie Pack Kaelbling. *Learning in Embedded Systems*. The MIT Press, Cambridge, MA, 1993.

[Kearns and Koller, 1999] Michael J. Kearns and Daphne Koller. Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 740–747, 1999.

[Kearns and Singh, 2002] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2–3):209–232, 2002.

[Mannor and Tsitsiklis, 2004] Shie Mannor and John N. Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *Journal of Artificial Intelligence Research*, 5:623–648, 2004.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[Sherstov and Stone, 2005] Alexander A. Sherstov and Peter Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[Strehl and Littman, 2004] Alexander L. Strehl and Michael L. Littman. An empirical evaluation of interval estimation for Markov decision processes. In *The 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2004)*, pages 128–135, 2004.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[Thrun, 1992] Sebastian B. Thrun. The role of exploration in learning control. In David A. White and Donald A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pages 527–559. Van Nostrand Reinhold, New York, NY, 1992.

[Weissman *et al.*, 2003] Tsachy Weissman, Erik Ordentlich, Gadiel Seroussi, Sergio Verdu, and Marcelo J. Weinberger. Inequalities for the L1 deviation of the empirical distribution. Technical Report HPL-2003-97R1, Hewlett-Packard Labs, 2003.

[Wiering and Schmidhuber, 1998] Marco Wiering and Jürgen Schmidhuber. Efficient model-based exploration. In *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior (SAB'98)*, pages 223–228, 1998.