Multi-Armed Recommendation Bandits for Selecting State Machine Policies for Robotic Systems

Pyry Matikainen,[1] P. Michael Furlong,[1] Rahul Sukthankar,[2,1] and Martial Hebert[1]

***Abstract*— We investigate the problem of selecting a state-machine from a library to control a robot. We are particularly interested in this problem when evaluating such state machines on a particular robotics task is expensive. As a motivating example, we consider a problem where a simulated vacuuming robot must select a driving state machine well-suited for a particular (unknown) room layout. By borrowing concepts from collaborative filtering (recommender systems such as Netflix and Amazon.com), we present a multi-armed bandit formulation that incorporates recommendation techniques to efficiently select state machines for individual room layouts. We show that this formulation outperforms the individual approaches (recommendation, multi-armed bandits) as well as the baseline of selecting the 'average best' state machine across all rooms.**

I. INTRODUCTION

For many classes of robotics applications it is difficult to generate good policies for individual instances, either because it is expensive to evaluate policies on the instances, or because the possible space of policies is too large to effectively optimize. For example, a robotic vacuum cleaner like the Roomba might be driven by a relatively simple state machine, yet directly learning even such a simple state machine for a particular room layout would be prohibitively expensive. Instead, these applications tend to pool instances together to learn generally good policies that perform well on average, although they may be far from optimal for any individual instance. Instead we consider another possibility: given a library of policies specialized for different instances, could a strategy be devised to simply *select* the best policy out of the library for a particular instance?

Policy selection is applicable to a wide range of robotics applications, for evaluation and illustrative purposes we consider the following simulated (yet plausible) scenario in this paper: a fleet of simple, state-machine driven robotic vacuum cleaners are purchased by individual users and set to work in their respective rooms whose layouts are unknown. If we consider the robot of a specific user then each time it is run the robot may pick a policy (a state machine) to execute from a library of such state machines and, after that run, it is told how much of the room it covered. Additionally, it may also access a central database to query how well the other state machines in the library have performed on *other* room layouts. The objective is to find a high coverage state machine as quickly as possible. Complicating the problem is the fact that each run starts in a randomized position in the room and that the state machines may be randomized themselves; thus, the coverage returned at the end of the run is only a noisy estimate of the true quality of that particular state machine (see Fig. 1 for an example of a simulated run of a state machine on a room).

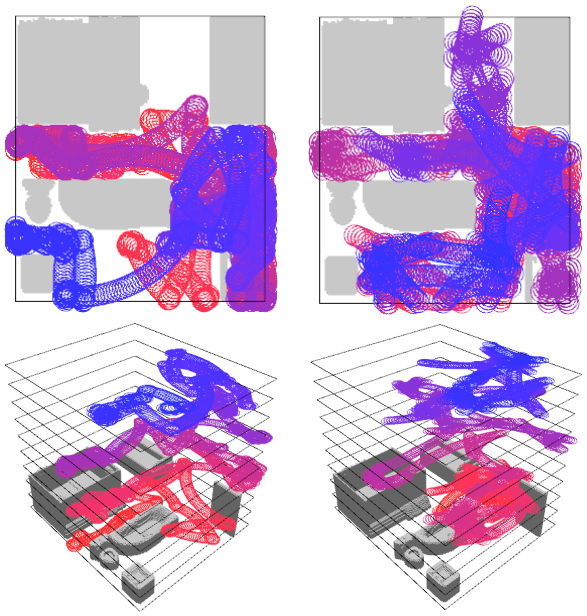

Fig. 1. Runs of two different state machines on the same room layout. Time is represented by both the vertical axis and color (red→blue). The average best state machine across all rooms (left) takes large sweeping curves that are good for open rooms, but which makes it navigate tight spaces less effectively. In contrast a state machine that is good for navigating tight spaces (right) does well on this room but poorly overall. The problem is how to effectively pick the state machine suited for each room.

There are two broad ways of approaching the problem: first, one might consider the multi-task aspect and how to use the reported coverages of the state machines across different unknown room layouts to improve the state machine choice for each individual room. Recently, Matikainen *et al*. [1] proposed a related approach for computer vision: using collaborative filtering techniques as seen in recommender systems (*e.g*., Netflix, Amazon.com) to recommend classifiers from a large library for a computer vision task. Analogously to the way that Netflix uses the collective *ratings* of a large pool of users to recommend movies to individuals based on their personal ratings, collaborative filtering techniques can be applied to selecting or *recommending* a state machine from a large library for a robotic vacuum cleaner in a new, unknown room configuration, given a database of how well those state machines perform on other room layouts. Then,

[1]The Robotics Institute, Carnegie Mellon University [2]Google Research

just as a Netflix user does not have to rate every movie in the library to receive recommendations, individual vacuuming robots do not have to try every state machine in the library.

Alternatively, the problem can be viewed as a k-armed bandit [2]: if there are a finite number (k) state machines, then each state machine can be seen as an arm of the bandit. At each trial or run, the robot can pick an arm to pull, and receive a randomized payout according to that arm's distribution. The goal is to converge on the arm with the highest long-term payout (expected payout) as quickly as possible. However, in the traditional multi-armed bandit formulation, the algorithm has to try each arm at least once to get some estimate of its quality [3]; if the state machine library is large, then even trying each option once could take prohibitively long.

While the recommendation approach is able to use the collective knowledge of the robots in order to collaboratively select policies, it makes those selections offline without tackling the sequential nature of the choices and the exploration-exploitation trade-off. Conversely, interpreting the problem as a k-armed bandit addresses the sequential nature and fundamental exploration-exploitation trade-off, but does not take advantage of the shared knowledge between robots operating on parallel tasks. In this paper we propose to merge the two interpretations, we combine collaborative filtering and k-armed bandits to produce an approach that outperforms either approaches individually.

II. RELATION TO COLLABORATIVE FILTERING

Applying collaborative filtering to this type of problem hinges on the relation of users to room layouts, items to robot state machines, and item ratings to the performance metric of the robot state machines on the room layouts. In brief, if there are a set of 'items' and 'users', and there is some way for a user to 'rate' an item (assign a numerical score of quality), collaborative filtering techniques address the problem of predicting the ratings a specific user will assign to items, based on how that user has rated a subset of the items, and a database of how *other* users have rated the items. Mechanically, this can be seen as a kind of vector completion: given a (possibly incomplete) matrix of ratings R with dimensions Items $\times$ Users and an incomplete column vector r of a new user's ratings of the items, predict the user's complete ratings vector r'.

For example, a broad class of collaborative filtering techniques (called 'factorization' methods) attempt to predict ratings by making the assumption that R is low-rank: $R \approx TU$, where T and U are rank k matrices produced by various factorization schemes. Since r' must then lie on a k dimensional subspace such that $r' = Tx$, it is possible to predict r' given at least k known ratings in r [1]. However, there is a broad foundation of collaborative filtering literature that can be drawn from to tackle this and related problems [4], [5], [6].

Matikainen *et al*. [1] made the connection that the labels 'users' and 'items' were mere conventions, and that computer vision tasks could be seen as 'users' and classifiers as 'items', so long as there was a way to assign ratings. In the case of classifiers, the natural rating of a classifier on a task was the *accuracy* of the classifier, and thus the mapping allowed collaborative filtering techniques to be used unaltered to predict the performance of classifiers for computer vision tasks. In this paper, we equate users to room layouts and items to state machines, with a room's 'rating' of a state machine being its average cleaning coverage on that room.

III. RELATED WORK

There is a significant body of work on multi-task policy optimization focusing on learning good policies for Markov Decision Processes (MDPs) or Partially Observable Markov Decision Processes (POMDPs). However, these types of approaches tend to be restricted to relatively small grid worlds (30×30) because the approaches become intractable for large state spaces [7], [8], [9], [10].

Attempts have been made to deal with large state spaces in various ways; for example, Sallans *et al*. [11] try to learn an approximator function for the value of a state-action pair. However, it may still take thousands of iterations to learn a good policy. An early work by Parr and Russell [12] considered reinforcement learning over partially specified state machines that could 'call' each other as sub-modules; this allowed larger state spaces (60×60), but the larger grid-world maps relied on explicitly repeated obstacles.

On the other end of the spectrum, rather than attempting to generate policies from a pure learning perspective, there has been much work on complete coverage problems for mobile robots [13], [14]. However, these works tend to require accurate localization of the robot whereas we consider a largely blind robot with only rudimentary sensing. The capabilities of our robot more closely resemble the early state-machine driven work of MacKenzie and Balch [15].

The execution and selection of state machines can be viewed as the sequential selection of experiments that provide greatest return (coverage). Robbins [2] introduced the multi-armed bandit formulation to address experiment selection. In our setting the experiments are executions of robot state machines on room layouts and the result is coverage of the room by the robot.

Lai and Robbins [16] value candidate experiments by summing the mean and standard deviation of observed rewards. By combining expected reward with an upper bound on expected reward [16] address the exploration/exploitation trade off. Auer *et al*. [17] refined the approach of Lai and Robbins further with the Upper Confidence Bound (UCB) algorithm. UCB values experiments by summing the expected reward and an index that increases with an decrease of relative evaluations of a given experiment relative to the total number of runs. For state machine i the UCB value can be written as $\mathbb{E}\left[\text{coverage}_i\right] + \sqrt{\frac{2 \ln n_i}{N_T}}$, where n_i is the number of times state machine i has been executed and N_T is the total number of times state machines have been evaluated. These approaches only need to know the result of a given experiment and not its operation. Bandit approaches are particularly suitable for the task of evaluating robot state

machines because they do not need to understand the internal workings of the robots, which for our technique can be considered black boxes.

In recent work, Yue *et al*. apply this upper confidence bound heuristic to a recommendation problem, where the objective is to select which news articles a user is likely to 'like' [18]. Each news article is parametrized into a feature space, and a user's preferences are modeled as a linear function of that feature space. Then, by keeping track of the uncertainty on the user's preference weight vector w, they are able to obtain uncertainties on the predicted rating a user will give to incoming articles, and they select the article with the highest upper confidence bound. By using the typical factorization assumption that users' preference vectors w lie on a low-dimensional subspace of all possible preference vectors, they are able to speed up the estimation of a particular user's w by using the w vectors of other users.

Compared to Yue *et al*.'s work, we take a similar, but more general approach in that we clearly encapsulate the recommendation system away from the bandit algorithm, allowing the use of any recommendation framework that is able to make recommendations with uncertainty bounds. Since the 'arms' of our bandits are known (unlike with news articles, where each article can be seen as a combination of 'arms' according to its representation in feature space), we do not need to assume an underlying feature representation, and in fact, we found in preliminary experiments that a neighborhood based recommendation system performs better in this application than a factorization based one.

IV. METHOD

A. Problem definition

The evaluation task is a simulated vacuuming robot. The robot runs a simple state machine and the objective is to cover (vacuum) as much of the floor of a room as possible within a fixed time limit.

Floor-plans are produced from the Google sketchup database of 3D models by finding models of furnished rooms and computing a rasterized 2D grid representation of the traversable floor area, discretized to approximately 40 cells per meter (each grid cell is 2.54 cm per side). A typical room might be 300×180 grid cells (7.6m × 4.6m). Although these are not strictly real floor plans, they are designed by humans and likely share the same general features as the arrangement of furniture within real rooms (See Fig. 2).

Out of the approximately 1500 floor plans obtained from Google Sketchup, we use approximately 1000 as the set used to build the database of coverages, and hold out 526 as the testing set.

B. Simulator

We model a simple robot operating in a 2D room layout. The robot is modeled as a disk with a 17.8 cm radius which can drive forward and backward (subject to maximum velocity constraints) and turn (the robot can turn completely in place), subject to maximum turning rate constraints. The robot is assumed to vacuum continuously, and a grid cell is 'covered' (vacuumed) if any part of the robot ever touches it. The robot cannot drive through furniture.

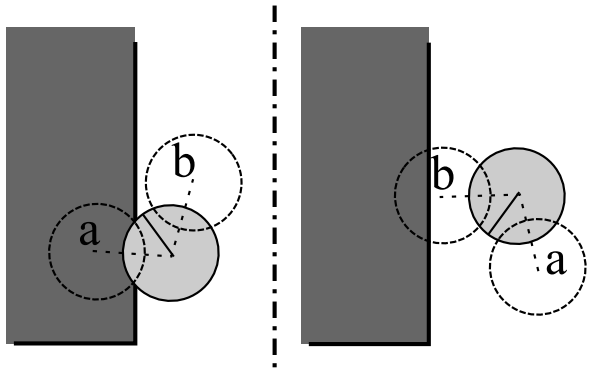

Fig. 3. Simulated robot sensors. Left: collision and bump sensor a are activated. Right: only bump sensor b is activated.

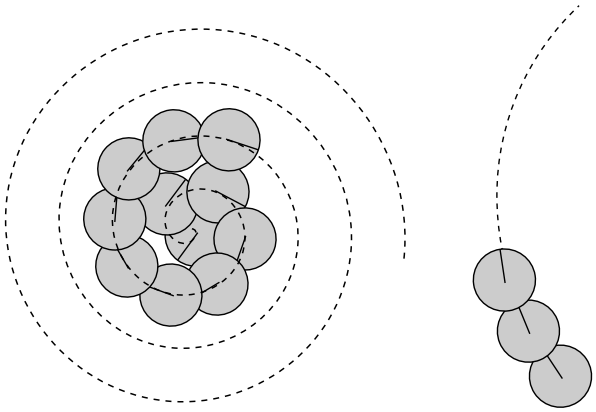

Fig. 4. Simulated robot drive modes. Left: "spiral" (robot drives at constant velocity in an Archimedean spiral), right: "drive" (robot drives at constant velocity and turn rate, producing a circular arc).

The robot has three types of sensors: it can detect when it has directly hit something (a collision), it has a number of 'bump' sensors, and it has timers. The 'bump' sensors are implemented as detecting whether an object of the same size as the robot would collide at a fixed relative location to the robot (see Fig. 3). Each timer senses whether a fixed amount of time has elapsed in the current state.

The robot runs a simple state machine. At each simulator tick t, the robot starts in some state s_t. The robot then evaluates all its sensors (collision, bump, and timers). Let E_t be the set of sensor events that occurred in the tick; a static transition table governs the state transitions, so that $T(s_t, e)$ returns the possible set of new states the robot could take after observing event e in state s_t. Note that there can be multiple outgoing transitions from a single state-event pair: in this case, the robot chooses one of the new states randomly with uniform probability. Since multiple events might fire in a single tick, let the total set of possible new states be the union of all the possible transitions, $P_t = \bigcup_{e \in E_t} T(s_t, e)$. Then, the robot randomly picks a new state from P_t.

Each state is associated with a fixed output 'instruction'.

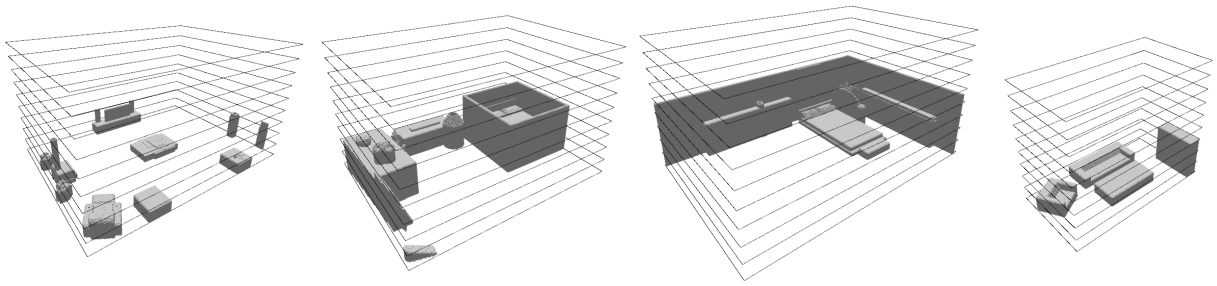

Fig. 2. Example room layouts obtained from the Google Sketchup library

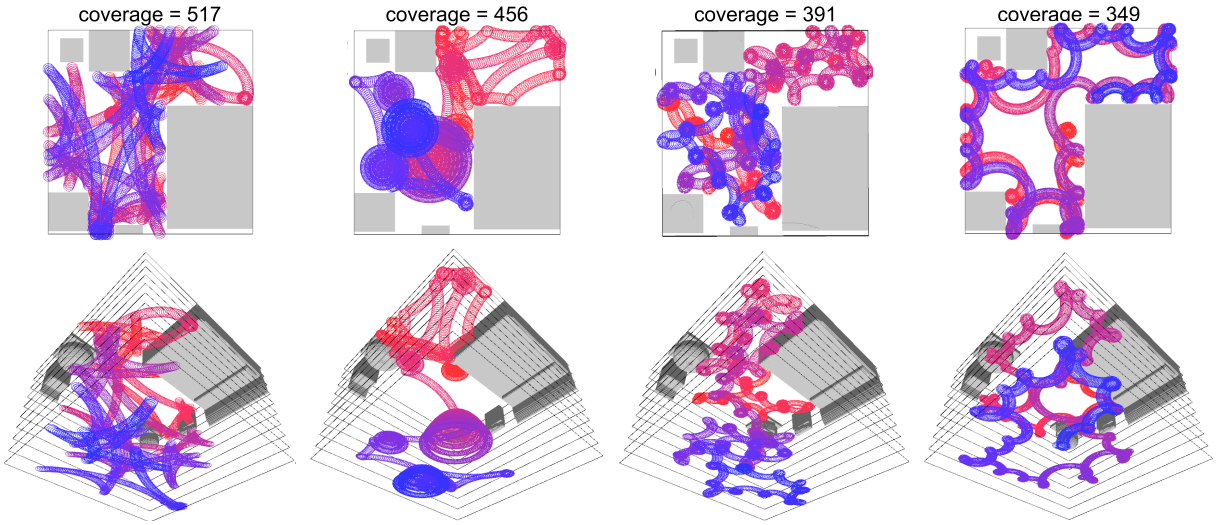

Fig. 5. Examples of different state machines running on the same room, one run of each state machine viewed from both overhead (top row, color is time: red→blue) and perspective (vertical axis and color are time). Despite the simplicity of the state machine representation, a wide range of behaviors is possible, and the interaction of these behaviors with room layouts is difficult to characterize analytically.

There are two classes of instructions: 'drive' and 'spiral'. In drive mode, the output specifies the linear velocity and turning rate, as fractions of the max (possibly negative). The maximum linear velocity of the robot is 35.6 cm/s and the maximum turning rate 12 rad/s. In spiral mode, the robot's linear velocity is fixed, and the turning rate is governed by the equation $v_\theta = \frac{v_0}{(\delta t+1)^{1/2}}$, which will drive the robot in an Archimedean spiral whose direction and spacing is governed by the output parameter v_0; δt is the amount of time elapsed in the current state. See Fig. 4.

Despite the simplicity of this state machine, there is a wide range of possible behaviors for the robots, a few of which are exhibited in Fig. 5.

Given this state machine definition, we generate a library of 200 state machines by picking 200 rooms from the training set, and for each room trying 700 random state machines and retaining the best for the library. That is to say, each state machine in the library is the best performer on a specific room in the training set.

C. Coverage Prediction through Collaborative Filtering

As described earlier, we map from the users and items of traditional collaborative filtering to room layouts and state machines. Then, to make the predictions, collaborative filtering methods need a database of the coverage of the state machines in the library on a number of rooms, or essentially a $|R| = \text{Models} \times \text{Rooms}$ matrix of coverages. We try each of the 200 models in the library on the 1000 training set rooms once to produce a 200×1000 recommendation database.

We denote the set of state machines whose coverages have been measured on the current room by C; these are the entries in the current room's ratings vector that are known. Based on these visible dimensions of the ratings vector, the room's ratings vector can be compared to the ratings vectors of the rooms in the database (the columns of R).

We used a k nearest neighbor approach to predict a coverage score based on state machines performance on rooms. Where N is the set of k nearest neighbors of the columns of R according to the ratings vector comparison (simple l^2 distance). The predicted mean coverage and variance for a model can then be computed according to Alg. 1. While Koren [5], [6] suggests that the correlation coefficient between ratings might be used to determine the neighbors, in practice we find that the simple Euclidean distance outperforms the correlation coefficient. Matikainen *et al.* [1] obtained the best results using a factorization based

collaborative filtering approach, for the scenario in this paper we find that a simple k-nearest-neighbor technique provides better results.

This can be seen as a special case of the neighborhood techniques used in the literature, whose complexity derives mainly from having to deal with missing ratings in the store R [4], [5] and scale to very large datasets [6]. Note that similar recommendations with uncertainty can be made with factorization based techniques, and Yue *et al*. [18] give an example of how to do this, although they do not describe their factorization procedure in terms of making recommendations.

Algorithm 1 k-Nearest-Neighbor Collaborative Filtering

function PREDCOVERAGE(C, m_i)
  $N \leftarrow \text{Neighbors}(C, R, k)$
  return $\mu_p = \frac{\sum_{q \in N} r_{iq}}{|N|}$
end function

function PREDVARIANCE(C, m_i)
  $N \leftarrow \text{Neighbors}(C, R, k)$
  $\mu_p \leftarrow$ PREDCOVERAGE(C, m_i)
  return $\sigma_p^2 = \frac{\sum_{q \in N} (r_{iq} - \mu_p)^2}{|N| - 1}$
end function

D. Evaluation Scenario

We evaluate strategies on an indefinite horizon scenario that is common to a wide range of robotics applications. In this scenario, a strategy is run on a test room for a number of trials which is not revealed to the robot using the strategy. The performance of a strategy is measured as the average accumulated fractional regret. For each trial's performance the fractional regret is computed as

$$\frac{\text{coverage}_o - \text{coverage}_t}{\text{coverage}_o},$$

where coverage_o denotes the coverage obtained by the optimal choice of state machine for that room and coverage_t the measured coverage of a state machine on trial t. Note that the optimal coverage coverage_o is not the optimal coverage out of all possible state machines, but rather optimal in the sense that it is the highest achievable coverage by any state machine *in the library*. It is coverage of the optimal choice out of the library, not the optimal coverage out of all possible controllers. The average accumulated fractional regret for a room on trial t is computed as

$$\frac{1}{t} \sum_{i=1}^{t} \frac{\text{coverage}_o - \text{coverage}_i}{\text{coverage}_o}.$$

Note that a strategy does not know its regret while running.

Successfully balancing the exploration/exploitation tradeoff involves minimizing a combination of the time (number of trials) it takes to converge to a choice and the regret of that choice, as it can be better to converge more quickly to a higher regret than to take a long time to obtain only slightly better asymptotic performance.

E. Strategies

Strategies are methods or algorithms for choosing state machines across iterated trials.

Strategies may use two pieces of information about each state machine: its MEASUREDCOVERAGE, which is the mean coverage (and variance) obtained by that state machine on the current room over all the times it was chosen (if a state machine has never been chosen, its measured coverage is undefined). Strategies may also use the PREDCOVERAGE(C, m) of a state machine m, which is the coverage predicted by the recommendation system given the measured coverages of the chosen state machines in the set C. Note that if $C = \emptyset$, that is, if a prediction is requested without having actually tried any state machines on the current room, then the recommender returns the mean coverage of the requested state machine across the database.

Before each trial t, a strategy may choose one state machine $m_t \in M$ from the library, after which the measured coverage of that state machine is available to the strategy.

We explore four different strategies:

1) Random Search: At each iteration, choose a random state machine that has not been previously chosen, and evaluate its coverage.

Algorithm 2 UCB Bandit

function INITIALIZE($M, t_{\max}$)
  $\mu \leftarrow \emptyset$
  $\sigma \leftarrow \emptyset$
  $n \leftarrow \emptyset$
end function

function CHOOSESM(t)
  if $|\mu| < |M|$ **then**
    $m_t \leftarrow m : m \in M \bigwedge m \notin \mu$
    $\mu_m \leftarrow$ MEASUREDCOVERAGE(m_t)
    $n_m \leftarrow 1$
  else
    $m_t \leftarrow \arg\max_{m \in M} \left[\mu_m + \sigma_m + \sqrt{2 \frac{\ln t}{n_m}} \right]$
    $n_m \leftarrow n_m + 1$
  end if
  $c_t \leftarrow$ MEASUREDCOVERAGE(m_t)
  $\mu_m, \sigma_m \leftarrow$ UPDATESTATS(μ_m, σ_m, c_t)
end function

2) Upper Confidence Bound (UCB) Bandit—without recommendation: The bandit uses the Upper Confidence Bound decision [17] rule to select the next state machine and has no input from the predictions of the recommender function. By necessity this strategy must first try every state machine at least once before it can begin to determine which state machine provides the best coverage. The strategy then computes a score for the state machines based on a running sample mean and sample standard deviation which are in turn derived from observations of the state machines running in a given room. See Alg. 2.

3) Recommender Bandit: This version of the bandit uses the recommender function and the history of observed cov-

Algorithm 3 Recommender Bandit

function INITIALIZE($M, t_{\max}$)
  $P \leftarrow \{\text{PREDCOVERAGE}(\emptyset, m) : m \in M\}$
  $C \leftarrow \emptyset$
  $n \leftarrow \{1 : m \in M\}$
end function

function CHOOSESM(t)
  $m_t \leftarrow \arg\max_{m \in P} \left[\begin{array}{l} \text{PREDCOVERAGE}(C, m) + \\ \text{PREDDEVIATION}(C, m) + \\ \sqrt{2\frac{\ln t}{n_m}} \end{array} \right]$
  $n_m \leftarrow n_m + 1$
  $C \leftarrow C \cup (m_t, \text{MEASUREDCOVERAGE}(m_t))$
end function

erages to estimate the mean and variance of the coverage for the different state machines. By relying on the recommender function for these quantities the bandit need not engage in lengthy periods of time exploring the space of models. See Alg. 3.

Algorithm 4 Neo-Bandit

function INITIALIZE($M, t_{\max}$)
  $P \leftarrow \{\text{PREDCOVERAGE}(\emptyset, m) : m \in M\}$
  $C \leftarrow \emptyset$
  $n \leftarrow \{1 : m \in M\}$
end function

function CHOOSESM(t)
  $m_t \leftarrow \arg\max_{m \in P} \left[\begin{array}{l} \text{BESTESTUPPBOUND}(m) + \\ \sqrt{2\frac{\ln t}{n_m}} \end{array} \right]$
  $n_m \leftarrow n_m + 1$
  $c_t \leftarrow \text{MEASUREDCOVERAGE}(m_t)$
  $\mu_m, \sigma_m \leftarrow \text{UPDATESTATS}(\mu_m, \sigma_m, c_t)$
end function

function BESTESTUPPBOUND(m)
  $\mu_p \leftarrow \text{PREDCOVERAGE}(C, m)$
  $\sigma_p^2 \leftarrow \text{PREDVARIANCE}(C, m)$
  if $n_m < 2$ **then**
    $\mu_b \leftarrow \mu_p$
    $\sigma_b^2 \leftarrow \sigma_p^2$
  else
    $\alpha = \frac{\sigma_p^2}{\sigma_p^2 + \sigma_m^2}$
    $\mu_b = \alpha \cdot \mu_m + (1 - \alpha) \cdot \mu_p$
    $\sigma_b^2 \leftarrow \alpha^2 \sigma_m^2 + (1 - \alpha)^2 \sigma_p^2$
  end if
  return $\mu_b + \sigma_b$
end function

4) Neo-Bandit: The Neo-Bandit is like the Recommender Bandit except that it does not fully trust either the predicted coverages or the measured coverages, but instead takes a minimum-variance weighted mean of the two to produce a best-estimate of each state machine's mean coverage (with uncertainty). Note that the algorithm uses the *variance of the sample mean* for the measured coverage variance (*i.e.*, as the number of measured samples goes to infinity, the uncertainty on the measurement should go to zero). See Alg. 4.

V. RESULTS

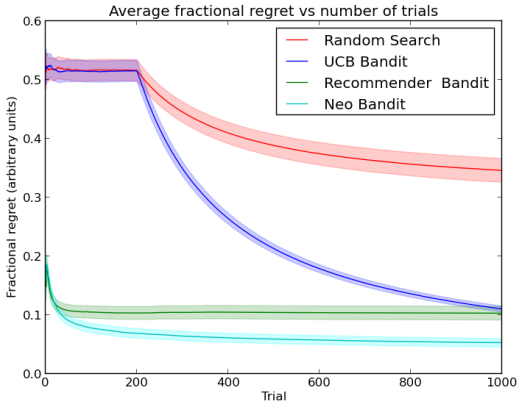

Fig. 6. Results for the indefinite horizon experiments. The solid lines represent the regret averaged over the test rooms the strategies were evaluated on. The shaded region represents a 95% confidence interval around that mean. The UCB bandit spends the first 200 trials examining the different state machines and is not significantly different from random search during that time. Once the bandit completes its initial exploration over the state machines its performance begins to converge towards the performance of the other bandit algorithms. The Recommender and Neo-Bandit algorithms both converge very quickly to a low average regret. The Neo-Bandit converges to a much lower average regret and with a tighter confidence interval. It is clear to observe that the Neo-Bandit is the superior algorithm to employ in this setting.

The results for the indefinite horizon experiments can be seen in Fig. 6. During their exploration phase the random search and the UCB bandit are not statistically different from each other. It is only after the initial exploration phase when the UCB bandit has the freedom to refine its estimates of coverage for the state machines that the differences in performance begin to emerge. Similarly the Recommender Bandit and the Neo-Bandit are initially not statistically different from each other, and both outperform the other two strategies well before the random search and UCB bandit finish their initial exploration. The UCB bandit had not converged before the end of the indefinite horizon experiment while the Recommender and Neo-Bandits have. Clearly using prior knowledge significantly reduces time to convergence.

By acting as a variance weighted combination of the UCB and Recommender bandits, the Neo-Bandit outperforms both, as can be seen in Table I. The Neo-Bandit had both the lowest average regret, approximately half that of the next best algorithm, and its performance is statistically significantly different from the other algorithms. Further the Neo-Bandit converges faster than the UCB bandit which does not converge before the end of the indefinite horizon experiment. All of the bandit algorithms produce more

reliable results than the random search, as implied by the standard error.

TABLE I
THE FINAL AVERAGE REGRET AND STANDARD ERROR OF REGRET FOR THE INDEFINITE HORIZON EXPERIMENT (ARBITRARY UNITS).

Algorithm	Final fractional regret	Final standard error
Random Search	0.346	0.0104
UCB Bandit	0.110	0.0026
Recommender Bandit	0.103	0.0059
Neo-Bandit	0.053	0.0038

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have demonstrated a method for quickly selecting a robot policy (state machine) for a cleaning vacuuming robot from a library of such policies. However, this method is general to robotics applications where directly learning a policy is difficult yet a library of reasonable policies can be generated, and where it is possible to quantify the quality of those policies on individual instances of the problem. Since the method considers the actual policies as black boxes and only directly considers the *ratings* of the policies, the underlying policies can take virtually any form: state machines, neural networks, or even hand-coded programs are all possibilities. The method could be applied to selecting gaits for different terrains, strategies for robotic soccer teams, or even grasps for different objects.

Furthermore, although we have presented results on simulations, the specific problem we have evaluated against is not a completely abstracted toy problem, as there are only a few practical hurdles involved with implementing the method on real cleaning robots. Foremost is the question of how coverage might be effectively measured in a real robot: while outfitting a robot with motion-capture markers or the like would certainly solve the problem, once accurate localization is a possibility, simply mapping the room directly becomes an option. One possibility is that a human being could directly rate the robot's performance on a simplified 'cleanliness' scale. These human-provided ratings would be extremely noisy, but collaborative filtering techniques were developed first to deal with noisy human ratings, so there is still hope that in aggregate there would be enough information to still make useful recommendations.

ACKNOWLEDGMENTS

This work was partially funded by the Army Research Laboratory under Cooperative Agreement #W911NF-10-2-0061. The views and conclusions are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation herein. We thank Shumeet Baluja for his helpful comments on the paper.

REFERENCES

[1] Matikainen, P., Sukthankar, R., Hebert, M.: Model recommendation for action recognition. In: CVPR. (2012)
[2] Robbins, H.: Some aspects of the sequential design of experiments. Bulletin of the American Mathematical Society **58** (1952) 527–535
[3] Berry, D., Fristedt, B.: Bandit Problems: Sequential Allocation of Experiments. 1985. Chapman and Hall, London (1985)
[4] Bell, R.M., Koren, Y.: Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In: ICDM. (2007)
[5] Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: ACM KDD. (2008)
[6] Koren, Y.: Factor in the neighbors: Scalable and accurate collaborative filtering. ACM Transactions Knowledge Discovery Data **4** (2010) 1:1–1:24
[7] Li, H., Liao, X., Carin, L.: Multi-task reinforcement learning in partially observable stochastic environments. Journal of Machine Learning Research (2009)
[8] Rosman, B., Ramamoorthy, S.: A multitask representation using reusable local policy templates. In Proc. AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI (2012)
[9] Wilson, A., Fern, A., Ray, S., Tadepalli, P.: Multi-task reinforcement learning: A hierarchical bayesian approach. In: Proceedings of the International Conference on Machine Learning. (2007)
[10] Tanaka, F., Yamamura, M.: Multitask reinforcement learning on the distribution of MDPs. In: Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation. Volume 3. (2003) 1108 – 1113
[11] Sallans, B., Hinton, G.E., Mahadevan, S.: Reinforcement learning with factored states and actions. Journal of Machine Learning Research **5** (2004) 1063–1088
[12] Parr, R., Russell, S.: Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems 10, MIT Press (1997) 1043–1049
[13] Wong, S., MacDonald, B.: A topological coverage algorithm for mobile robots. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Volume 2. (2003) 1685 – 1690 vol.2
[14] Choset, H., Pignon, P.: Coverage path planning: The Boustrophedon cellular decomposition. In: International Conference on Field and Service Robotics. (1997)
[15] Mackenzie, D., Balch, T.R.: Making a clean sweep: Behavior based vacuuming (1993)
[16] Lai, T., Robbins, H.: Asymptotically efficient adaptive allocation rules. Advances in applied mathematics **6** (1985) 4–22
[17] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine learning **47** (2002) 235–256
[18] Yue, Y., Hong, S.A., Guestrin, C.: Hierarchical exploration for accelerating contextual bandits. In: ICML. (2012)