In *The Ninth International Conference on Development and Learning* (ICDL 2010),
Ann Arbor, Michigan, August 2010.

Real Time Targeted Exploration in Large Domains

Todd Hester and Peter Stone
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712
{todd,pstone}@cs.utexas.edu

***Abstract*—A developing agent needs to explore to learn about the world and learn good behaviors. In many real world tasks, this exploration can take far too long, and the agent must make decisions about which states to explore, and which states *not* to explore. Bayesian methods attempt to address this problem, but take too much computation time to run in reasonably sized domains. In this paper, we present TEXPLORE, the first algorithm to perform targeted exploration in real time in large domains. The algorithm learns multiple possible models of the domain that generalize action effects across states. We experiment with possible ways of adding intrinsic motivation to the agent to drive exploration. TEXPLORE is fully implemented and tested in a novel domain called Fuel World that is designed to reflect the type of targeted exploration needed in the real world. We show that our algorithm significantly outperforms representative examples of both model-free and model-based RL algorithms from the literature and is able to quickly learn to perform well in a large world in real-time.**

I. Introduction

A important part of developmental learning is being able to learn with minimal supervision. Reinforcement learning (RL) is a popular method in the machine learning community for learning with minimal supervision. However, its ability to be useful for developmental learning has been hampered because most RL algorithms require that the state space be explored exhaustively. In order to apply the power of RL to the learning problems of a developing agent in a vast state space, there needs to be a way to explore the space intelligently. In this paper we present an algorithm that does that.

We illustrate the exploration problem through an example. Imagine you've moved to a new city, and you're trying to navigate from your house to a store at a particular intersection across town. There are some things you know about the problem: you know the city is laid out in blocks, you know how your car works, and you know where your destination is. But you don't know the best path, which roads are faster or have fewer stoplights, which ones are dead ends, or where the gas stations and post offices are. What is the optimal way to explore when presented with this problem?

You most likely want to explore a few possible paths to your destination. But you don't need to explore exhaustively, as eventually the possible cost of exploring (running into a dead end or extremely slow street) will outweigh the possible benefits of finding a faster route. In fact, after you've been on a few roads in a slow neighborhood, you probably want to *avoid* exploring additional paths that go through that area.

RL addresses the problem of finding effective solutions to such sequential decision making problems [1]. The goal of RL algorithms is to maximize a reward signal (a scalar) over time, which can either come from the agent or the external environment. In tasks such as this one, where taking actions is expensive or time-consuming, it is important that an RL algorithm be *sample efficient*: it takes very few actions to learn an effective policy. The size of many real world tasks also require sample efficient algorithms so the agent can learn in a reasonable amount of time.

Model-based methods are a class of RL algorithms that are particularly sample efficient. They learn a model of the domain from their experiences, and then plan on that model to calculate good policies in the domain. To be sample efficient, these methods must acquire the necessary experiences to effectively learn the model. Typical approaches to this exploration problem include exploring *randomly* or *exhaustively*. However, neither of these approaches are effective in practical problems.

In this paper, we present an algorithm that tries to explore the minimal number of states necessary to learn an accurate model of the domain and learn a good policy. It explores states that are promising, but also learns what states *not* to explore when the expected costs of exploration outweigh the expected benefits. Note that because the agent is not exploring exhaustively, there is a chance that the agent will miss a great outcome that it did not think was possible (for example, a wormhole that directly transports the agent across the city). Just as any agent in a large and varied environment (such as the real world) must, we sacrifice this guarantee of optimality for high expected reward in reasonable time frames.

II. Background

We adopt the standard Markov Decision Process (MDP) formalism for this work [1]. An MDP consists of a set of states S, a set of actions A, a reward function $R(s,a)$, and a transition function $P(s'|s,a)$. In many domains, the discrete state s is represented by a vector of n discrete state variables $s = \langle x_1, x_2, ..., x_n \rangle$. 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'. The new state s' is determined from the probability distribution $P(s'|s,a)$.

The value $Q^*(s,a)$ of a given state-action pair (s,a) is an estimate of the future reward that can be obtained from (s,a) and is determined by solving the Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \qquad (1)$$

where $0 < \gamma < 1$ is the discount factor. 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. The optimal policy π is then as follows:

$$\pi(s) = \text{argmax}_a Q^*(s, a) \tag{2}$$

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 plan on this model through a method such as value iteration [1] or UCT [2], effectively updating the Bellman equations for each state using their model. RL algorithms can also work without a model, 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; their sample efficiency is only constrained by how many actions it takes to learn a good model of the domain.

The agent's model of the domain can be learned using a number of techniques. A common approach is to use a tabular model where the agent learns a model for each state-action based on the frequencies of different outcomes at each state. The agent could also learn the model using any supervised learning technique, such as decision trees [3] or Gaussian Process regression [4].

III. Related Work

Our goal in this work is to develop an agent that can explore intelligently by combining an effective model learning method with targeted exploration. There are a number of related algorithms that address the exploration problem.

R-MAX [5] is a typical model-based approach that uses a tabular model and explores thoroughly by driving the agent to visit each state-action m times. It is guaranteed to find the optimal policy in time polynomial in the number of states and actions, but this may still result in an inordinate amount of time spent exploring the domain.

Many methods such as SPITI [6] use ϵ-greedy exploration, where the agent takes what it thinks is the optimal action most of the time, but takes a random action ϵ of the time. Random exploration is guaranteed to explore the entire state space when given an infinite number of actions, but does not attempt to explore in any targeted way.

Bayesian RL methods seek to solve the exploration problem optimally by maintaining a distribution over possible models inside their state representation and taking actions to maximize reward in these models. Doing so, however, requires the agent to include its belief over the model in its state representation, and the transition dynamics have to include both transitions in the actual domain as well as changes in the agent's model. This augmented state representation results in a enormous state space, making the full Bayesian algorithm intractable. Attempts have been made to approximate the full algorithm by parameterizing the model and tying model parameters together [7] or sampling from the model distribution [8], [9], but these methods are still only tested in domains with 5-36 states. In addition to requiring a large amount of time to compute a policy, these methods must maintain a belief state over the model and require the user to create a well-defined model prior.

Simsek and Barto [10] take a similar approach, examining the optimal exploration problem where the agent does not care about external rewards. They derive a second MDP whose reward is the improvement of the value of the greedy policy of the original MDP. Both the original and derived MDPs are learned using Q-learning, resulting in an agent that takes actions in the parts of the statespace where values are improving the most. However, this method does not work with a model-based method learning the original MDP, and does not consider the balance of exploration and exploitation.

Another approach that addresses our problem is Gaussian Process RL. Deisenroth and Rasmussen [4] present one such approach, where the agent maintains a model of the domain using Gaussian Process regression. This model is able to generalize experience to unknown situations as well as represent uncertainty. This approach has achieved great results on motor control problems such as the inverted pendulum and cart-pole problems. However, the algorithm requires ten minutes of computation time for every 2.5 seconds of experience. Also, the algorithm is provided a cost function of how far the agent is from the target state.

Oudeyer et al. [11] present Intelligent Adaptive Curiosity (IAC), a method for providing intrinsic reward to encourage a developing agent to explore. Their approach does not adopt the RL framework, but is similar in many respects. In it, they split the state space into regions and attempt to learn a model of the transition dynamics in each region. They maintain an error curve for each region and use 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. The resulting intrinsic motivation drive is close to what we desire, but their algorithm selects actions only to maximize the immediate reward, rather than the discounted sum of future rewards. In addition, their method has no way of incorporating external rewards or weighing their value in deciding what to explore.

In previous work [3], we presented an algorithm, RL-DT, that uses decision trees to model the transition and reward dynamics in the domain. The algorithm is able to build a model that generalizes experience to unknown states well. However, RL-DT uses a heuristic for exploration: it explores exhaustively until it has found a state-action with at least some percentage of the maximum reward in the domain, and then it switches to exploiting its model. In many cases this algorithm over-explores, and it does not make any attempt to target its exploration on states that will most improve its model.

Knows What It Knows (KWIK) [12] is a learning framework for efficient model learning. A learning algorithm that fits the KWIK framework must always either make an accurate prediction, or reply "I don't know" and request a label for that example. KWIK algorithms can be used as the model learning methods in an RL setting, as the agent can be driven to explore the states the model does not know to improve its model quickly. The drawback of KWIK algorithms is that they often require a large number of experiences to guarantee an accurate prediction when not saying "I don't know."

Here, we present a novel RL algorithm, TEXPLORE, that does not have the drawbacks of the methods presented above. Specifically, it does not explore as much as R-MAX, it can run in real-time on large domains unlike Bayesian or Gaussian Process methods, and it is able to reason about future rewards unlike IAC. It is the first method to perform targeted exploration in real-time in large domains.

IV. Algorithm

TEXPLORE is a model-based algorithm that learns multiple possible models of the domain and averages them to come up with a model that represents its uncertainty in the domain (Figure 1). It is also able to explore areas where it is particularly uncertain (where the models' predictions vary the most) by adding intrinsic rewards for exploration into its model. The agent re-plans its policy on this model at each time step, finding the best action according to its current model and intrinsic rewards.

A. Model Learning

In many domains, actions have similar effects across states. For example, in robotics tasks, actions may move a joint by some number of degrees in each state. The transition effects, or the relative change in a state, are therefore easier to generalize across states than absolute outcomes. For this reason, we choose to learn models of transition effects, $s_e = s' - s$, rather than absolute outcomes.

The algorithm learns a model of the domain by learning a separate prediction for each of the n state features and the reward. Assuming that each of the state variables transition independently, these separate models can be combined to create a complete model of the transition effects. The probability of the transition effect $P(s_e|s,a)$ is the product of the probabilities of each of its n state features:

$$P(s_e|s,a) = \Pi_{i=0}^{n} P(x_e^i|s,a) \qquad (3)$$

Building a model in this way assumes that each state feature can be predicted independently, which may be incorrect in many cases, but has not been a detrimental assumption in the domains we have tested.

We build each model using a random forest [13]. The random forest is a collection of decision trees, each of which is trained on only a subset of the agent's experiences. Similar to the distribution over possible models that Bayesian methods use, each tree represents a possible model of the domain. The final prediction for a state-action is the average of the predictions of each of the trees. For example, if four trees predicted outcome A, and one tree predicted outcome B, the final model would have an 80% probability of outcome A and a 20% probability of outcome B.

Averaging multiple possible models of the domain inherently incorporates uncertainty in the model. If all the models agree on the outcome of a particular state-action, then it is likely to be correct. Moreover, averaging the models in this way also allows the trade-off between exploration costs and potential benefits to be handled naturally. For example, if the models disagree and the average model predicts there is a small chance of a particular high-valued outcome occurring, it may be worth exploring even if there is a low probability that it is real. On the other hand, if this outcome has a large negative value, the possibility that exploring it could be costly should make the agent avoid it.

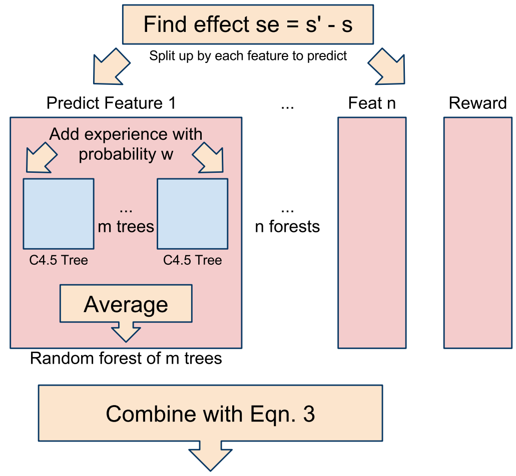

Fig. 1. *Model Learning.* This is how the algorithm learns a model of the domain. The agent calculates the difference between s' and s as the transition effect. Then it splits up the state vector and learns a random forest for each state feature. Each random forest is made up of stochastic decision trees, which get each new experience with probability w. The random forest's predictions are made by averaging each tree's predictions, and then the predictions for each feature are combined into a complete model of the domain.

When the agent takes an action, it saves an *experience*, (s, a, s', r), of the state-action it took, the outcome, and the reward it received. Each tree is updated with the new experience with probability w, taking the (s, a) as input and trying to predict a state feature from s_e or the reward.

Each decision tree is built recursively using the C4.5 algorithm [14]. At each node, the algorithm calculates the gain ratio for each potential split, and splits the tree using the split with the highest gain ratio. If the best gain ratio is not above some threshold t, then the node becomes a leaf node. To increase stochasticity in the models, at each split, each feature is removed from the set used for potential splits with probability f. When making a prediction, the tree finds the leaf corresponding to the given state-action, and outputs a probability for each outcome in the leaf based on the percentage of the leaf's outcomes that it makes up.

As an example, in the forest predicting rewards for the example in the introduction, each tree may disagree on where the 'slow' neighborhood is based on the experiences it has seen. One tree may say that the reward is -10 when the STREET feature is less than 10 and -1 otherwise, while another may say that the split occurs at 32nd street instead.

B. Exploration

There are a number of possible ways to incorporate exploration on top of the learned model of the domain. As stated earlier, we hypothesize that acting greedily with respect to the average of multiple possible models will perform well. Another option is to use information based on the variance of predictions by each model. Similar to [4], we experimented with an approach where the agent is given intrinsic rewards based on the variance of the model's predictions. In this case, the model's reward R is modified as follows:

$$R(s,a) = R_o(s,a) + b\sigma^2(s,a) \quad (4)$$

$$\sigma^2(s,a) = \frac{1}{n+1}[\sigma^2 R(s,a) + \sum_{i=1}^{n} \sigma^2 P(s_i^t|s,a)] \quad (5)$$

where $R_o(s,a)$ is the original prediction of the model, $\sigma^2(s,a)$ is the variance in the model's predictions, averaged over each feature and reward model, and b is a coefficient either adding or subtracting intrinsic rewards from the model based on the variance. By setting $b < 0$, the agent will avoid states that the model is uncertain about; setting $b > 0$ will result in the agent being driven to explore these uncertain states. If $b = 0$, the agent will act greedily with respect to its model. Changing the parameter b affects how aggressive the agent is in trying to improve uncertainties in its model. In addition to these methods, we ran experiments using ϵ-greedy exploration, where the agent takes a random action ϵ of the time.

C. Planning

Each time the agent's model is updated, it needs to re-plan what the best action is based on its model. TEXPLORE plans using a version of the popular UCT algorithm [2] that is modified by incorporating eligibility traces and generalizing values across depths in the search tree. To select an action, UCT searches ahead from the agent's current state in the model to find the best action. It is an anytime algorithm and will keep improving its action choice until it is stopped.

Similar to the prior that is created for Bayesian RL algorithms, we wish to provide our algorithm with some basic knowledge of the structure of the domain. We do this by seeding the agent with a few sample experiences from the domain, which it uses to initialize its model. Since these seeds bias the agent's expectations of the domain, the agent's performance is sensitive to them. We explore this effect empirically in Section VI.

V. EXPERIMENTS

We tested the algorithms in a novel domain called *Fuel World* which we created to examine exploration, shown in Figure 2. 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 with 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 domain has 21×31 cells, each with 61 possible energy levels, and 8 possible actions, for a total of $317,688$ state-actions. The agent does not start with enough fuel 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.

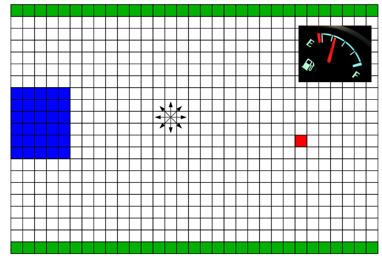

Fig. 2. The Fuel World domain. Starting states are in blue, fuel stations in green, and the goal state is shown in red. The possible actions the agent can take are shown in the middle.

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

$$R(x) = \mathit{base} - (x \bmod 5)a \quad (6)$$

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. We designed two versions of the domain to examine the effect of these costs on exploration. In the first version, $a = 1.0$ and $\mathit{base} = -18.0$ for the bottom row and -21.0 for the top row, resulting in rewards ranging from -18.0 to -25.0. The second version has more variation, as $a = 5.0$ and $\mathit{base} = -10.0$ for the bottom row and -13.0 for the top row, resulting in rewards from -10.0 to -33.0.

We ran experiments comparing versions of TEXPLORE using models with 1, 5, and 15 decision trees. For the methods with multiple decision trees, we ran experiments with intrinsic rewards from Equation 4 with $b = 35, 9, 0$, and -35. Planning was performed using UCT with $\lambda = 0.05$ and planning stopped after 0.1 seconds (so that the agent takes 10 actions per second). Based on informal testing, the experiments were run with $w = 0.6$ and $f = 0.2$. In addition, we compared against two baseline methods, which are typical model-free and model-based approaches: Q-LEARNING [15] and R-MAX [5]. Each of these algorithms strive for optimality; but achieving provably optimal performance requires them to explore the domain thoroughly, which is not practical in large domains such as this one. The discount factor, γ, was set to 0.99.

All of the algorithms (including Q-LEARNING and R-MAX) are given seeding experiences in the domain. They are given two experiences from the goal state, and two transitions from each row of fuel stations, for a total of six seeding experiences. Since Q-LEARNING and R-MAX both employ tabular representations with no generalization, the sample experiences are only useful to them in the exact states they occurred in. In contrast, TEXPLORE's random forest models are able to generalize these experiences across state-actions.

VI. RESULTS

Figure 3 shows the average reward per episode over 30 trials for the baseline methods and TEXPLORE using 15 decision

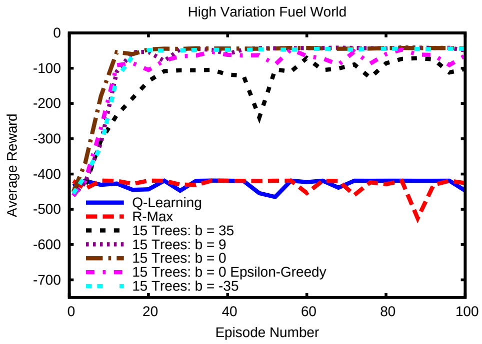

Fig. 3. Average reward over the first 100 episodes in the high variation Fuel World domain. Results are averaged over 30 trials.

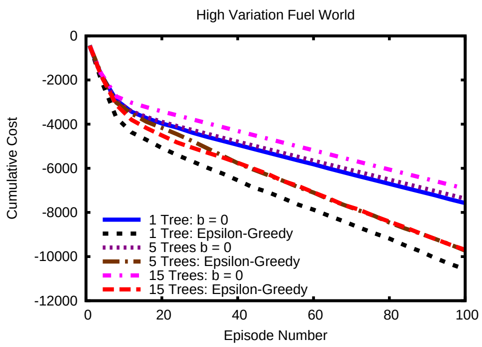

Fig. 4. Cumulative reward over 100 episodes in the high variation Fuel World domain. Results are averaged over 30 trials.

trees in the high variation Fuel World domain. R-MAX and Q-LEARNING both fail miserably, running out of fuel quickly every episode, and all the methods other than $b = 35$ accrue significantly more reward per episode ($p < 0.001$) than them starting in episode 5. The algorithm with $b = 35$ learns to get fuel to stay alive, but over-explores, accruing large negative rewards. The other methods all learn the task quickly and perform well. After 300 episodes, the greedy method ($b = 0$) accrues significantly ($p < 0.001$) more rewards than the others, closely followed by the version with a small incentive to explore uncertain states ($b = 9$).

Next we compared how the best exploration type ($b = 0$) and ϵ-greedy exploration fared with either 1, 5, or 15 decision trees. In the case with one tree, it was updated with all the experiences and the full set of features to split on. The cumulative rewards of these methods over 100 episodes on the high variation Fuel World domain are shown in Figure 4. More decision trees result in better policies, as the greedy method with 15 trees accumulates significantly more reward ($p < 0.001$) than the 5 tree algorithm after the 3rd episode, and the 5 tree method accrues significantly more reward ($p < 0.001$) than the 1 tree method after the 26th episode. In addition, using the average model to explore greedily is better than doing ϵ-greedy exploration, as all the greedy methods have significantly more reward ($p < 0.001$) than all the ϵ-greedy methods after episode 6. These results look similar in the low variation Fuel World.

To further examine how the agents were exploring, we made heat maps showing which states the agents visited. The colors

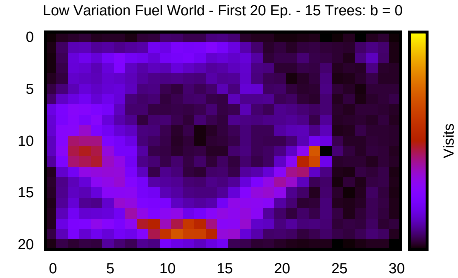

Fig. 5. Heat map displaying the average number of visits the agent with 15 trees and $b = 0$ took to each state over the first 20 episodes in the low variation Fuel World, averaged over 30 trials and all fuel levels.

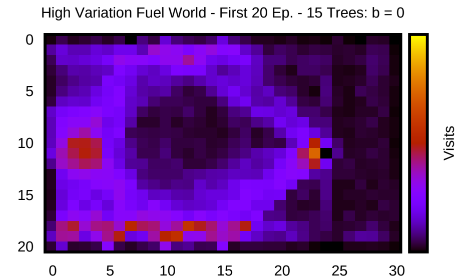

Fig. 6. Heat map displaying the average number of visits the agent with 15 trees and $b = 0$ took to each state over the first 20 episodes in the high variation Fuel World, averaged over 30 trials and all fuel levels.

represent the number of times the agent visited each cell in the domain (averaged over 30 trials and all fuel levels), with brighter colors meaning more visits. Figure 5 shows the visit map over the first 20 episodes for the algorithm with 15 decision trees and $b = 0$ in the low variation Fuel World domain, and Figure 6 shows the same for the high variation domain. First, we can see that both algorithms are mainly exploring states near the fuel stations and the path to the goal, ignoring the space in the middle and right of the domain. Looking at the cells in the bottom row between columns 10 and 15, we can see that the agent in the low variation world explored more of these fuel stations. Because of the extra variation in the costs of fuel stations in the second world, exploration is more costly (it may result in up to -20 more reward) and the agent decides it is not worthwhile to explore the fuel stations as thoroughly.

The effects of the agent's different exploration in these two domains can be seen in its final policy in each domain. Figures 7 and 8 show their visits over the final 50 episodes on the two domains. The optimal policy in both domains is to use the fuel station in column 15 on the bottom row. Going to the station in column 10 is only slightly worse, with an expected total reward only 0.6 less than going to column 15. In the low variation version, the agent explored the stations and largely settled on one of these two policies. In the high variation version, the various trials settled on a number of different policies, all of which are within 7.4 reward/episode of the optimal policy. Since the reward within one fuel row can vary up to 20.0 in this domain, it is not worthwhile to the agent to possibly receive this additional cost for a maximum possible benefit for 7.4.

For comparison, we show the visit map for the algorithm

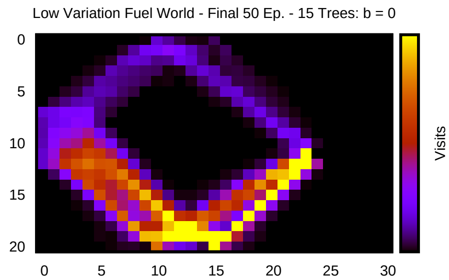

Fig. 7. Heat map displaying the average number of visits the agent with 15 trees and $b = 0$ took to each state over the final 50 episodes in the low variation Fuel World, averaged over 30 trials and all fuel levels.

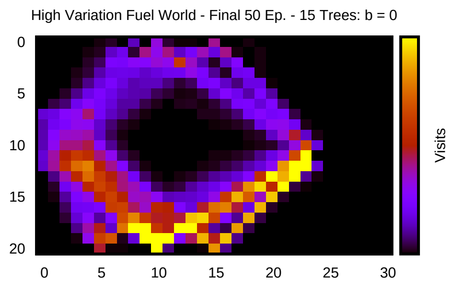

Fig. 8. Heat map displaying the average number of visits the agent with 15 trees and $b = 0$ took to each state over the final 50 episodes in the high variation Fuel World, averaged over 30 trials and all fuel levels.

with 15 trees and $b = 35$ over the first 20 episodes in the low variation domain in Figure 9. Here the agent explored a lot more states in general, and more fuel stations in particular. Still, because of the extra cost of exploring the fuel stations, the agent spent most of its time in the rows adjacent to the fuel stations, presumably navigating between the ones it wants to explore. Due to this extra exploration, in both versions of the domain, this agent settles on better final policies, but accrues significantly more negative reward while exploring.

Finally, we examined how the seeding affects the performance of the algorithm. If given no seeding experiences, the algorithms have no expectation of goal states or fuel stations being possible. The algorithms eventually find the fuel stations, but never find the goal state, simply learning to live a long time near the fuel stations. When we provide the algorithm with a single experience of the goal state, it still has no idea of the existence of fuel stations. In this case, we see that the methods using intrinsic rewards to drive exploration ($b > 0$) are able to find the fuel stations faster and accrue higher cumulative rewards. The algorithm that is driven to avoid uncertain states ($b = -35$) has a lot of difficulty, accruing 13 times more costs

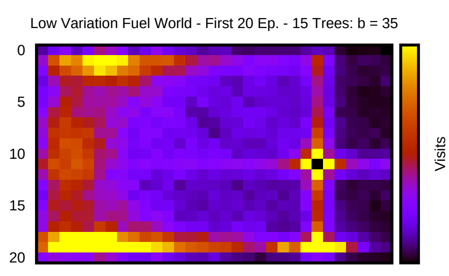

Fig. 9. Heat map displaying the average number of visits the agent with 15 trees and $b = 35$ took to each state over the first 20 episodes in the low variation Fuel World, averaged over 30 trials and all fuel levels.

than the best methods.

VII. Discussion and Conclusion

We have presented an algorithm that explores efficiently by averaging possible models of the domain. Unlike methods that guarantee optimality by exploring exhaustively, TEXPLORE learns quickly by discovering which states *not* to explore. It weighs the expected costs of exploring with the expected benefits of learning a better policy, and can thus learn a task quickly. TEXPLORE also runs in real-time on large domains, which is something that Bayesian methods that try to achieve similar exploration fail to do. The algorithms using a model that generalizes were significantly better than the baseline algorithms. Comparing only the algorithms using this model, significantly more reward was accrued by acting greedily with respect to an average model than using ϵ-greedy exploration or a single model, thus demonstrating that our exploration method is better than these methods under identical conditions.

This work has a number of possible directions for future work. One possible extension is to use a different type of model that does not require the feature independence assumption of Equation 3. We also want to extend the current model learning and planning methods to work in continuous domains and robotics problems in particular. Finally, we are interested to see if using solely the intrinsic rewards from Equation 4 and no external rewards will provide enough intrinsic motivation for a developing, curious agent.

References

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

[2] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *In: ECML-06. Number 4212 in LNCS*. Springer, 2006, pp. 282–293.

[3] T. Hester and P. Stone, “Generalized model learning for reinforcement learning in factored domains,” in *AAMAS*, May 2009.

[4] M. P. Deisenroth and C. E. Rasmussen, “Efficient reinforcement learning for motor control,” in *10th International PhD Workshop on Systems and Control*, Hluboka nad Vltavou, Czech Republic, Sept. 2009.

[5] R. I. Brafman and M. Tennenholtz, “R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning,” in *IJCAI*, 2001, pp. 953–958.

[6] T. Degris, O. Sigaud, and P.-H. Wuillemin, “Learning the structure of factored markov decision processes in reinforcement learning problems,” in *ICML ’06*. New York, NY, USA: ACM, 2006, pp. 257–264.

[7] P. Poupart, N. Vlassis, J. Hoey, and K. Regan, “An analytic solution to discrete bayesian reinforcement learning,” in *ICML ’06*. New York, NY, USA: ACM, 2006, pp. 697–704.

[8] M. Strens, “A Bayesian framework for reinforcement learning,” in *ICML ’00*, 2000, pp. 943–950.

[9] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate, “A bayesian sampling approach to exploration in reinforcement learning,” in *UAI*, 2009.

[10] Ö. Şimşek and A. G. Barto, “An intrinsic reward mechanism for efficient exploration,” in *ICML*, 2006, pp. 833–840.

[11] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, “Intrinsic motivation systems for autonomous mental development.” *IEEE Trans. Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, 2007.

[12] L. Li, M. L. Littman, and T. J. Walsh, “Knows what it knows: a framework for self-aware learning,” in *ICML*, 2008, pp. 568–575.

[13] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[14] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, pp. 81–106, 1986.

[15] C. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, University of Cambridge, 1989.