Computing Exploration Policies via Closed-form Least-Squares Value Iteration *

Lawrence A. Bush, Brian Williams and Nicholas Roy
bushL2@csail.mit.edu, williams@mit.edu, nickroy@mit.edu
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA, 02139

Abstract

Optimal adaptive exploration involves sequentially selecting observations that minimize the uncertainty of state estimates. Due to the problem complexity, researchers settle for greedy adaptive strategies that are sub-optimal. In contrast, we model the problem as a *belief-state* Markov Decision Process and show how a non-greedy exploration policy can be computed using least-squares value iteration. The key to generating such a policy is our use of a non-standard reward function for which we derive a closed form expected reward expression, given a set of sensing actions. Furthermore, by carefully selecting features to approximate the value function, we can compute value iteration backups in closed form. We apply our approach to adaptive occupancy mapping and demonstrate our method on problems of up to 10^3 grid cells, solving example problems within 1% of the optimal policy.

Introduction

We are very good at collecting lots of information. Our research is about collecting the right information. The optimal adaptive exploration problem is to sequentially select observations that minimize the uncertainty of a state estimate. A common example of this problem is occupancy map exploration, in which a sensor seeks to minimize the uncertainty over which grid cells are occupied, using a finite set of sequential observations. The tools we have at our disposal include a sensing platform, a generic signal processor, and a pattern recognition engine to detect activities of interest.

A concrete example is autonomous ocean sampling (Monterey Bay Aquarium Research Institute, 2006), where networked sensing platforms detect activity of interest over a geographic area (Figure 1). For example, a satellite could collect ocean surface color data, which provides indications of algae bloom formation. The data could be automatically analyzed and aggregated into an algae bloom formation activity map.

In this paper, we generate a synoptic activity map. Our probablistic activity map provides a holistic approach to characterizing ocean observations. The map can subsequently be used to direct the attention of other platforms (e.g. autonomous underwater vehicles (AUVs)) for focused exploration.

The method of searching the area affects the quality of the map, and the problem is to generate the most informative search plan. Due to the overall problem complexity, researchers settle for sub-optimal adaptive greedy strategies for choosing sensing locations. In other words, rather than considering the combined consequences of the current and potential future actions, only the first action is evaluated. While greedy strategies perform reasonably well (Guestrin et al., 2005), with a worst case optimality of $(1 - 1/e)$ (Krause & Guestrin, 2005), for many applications expensive resources are deployed based on the information collected, amplifying the impact of sub-optimality. Therefore, a better solution is very desirable. In this paper, we show how domain specific problem structure can be exploited, and a better sensing strategy can be efficiently learned.

Figure 1: The AOSN in Monterey Bay integrates a variety of modern platforms (e.g. satellites and AUVs) to produce a comprehensive regional view of the ocean. $http//www.mbari.org/muse/images/MUSE\_2001.jpg$

Methods: We review the use of open-loop feedback control (Bertsekas, 2005) and discrete state policy learning. Although both methods would improve optimality, they are intractable. Specifically, they solve problems infinitesimal in size relative to relevant real-world problems.

Thus, our approach is to model the problem as a *belief-state* Markov Decision Process (MDP) and learn a non-

*Research supported in part by The Boeing Co. grant MITBA-GTA-1.

myopic approximate policy via least-squares value iteration (LSVI). Our key contribution is the closed form backup, enabling rapid learning and rapid *rollout*. Specifically, we perform LSVI efficiently by employing a non-standard reward function for which we derive a closed form expected reward expression (for a given set of sensing actions). By carefully selecting the features to approximate the value function, we can compute a full value iteration backup in closed form. Specifically, we compute an entire value iteration backup for all state samples and control actions as a single compact matrix multiplication. We then use a full *rollout algorithm* to select the control action, which maximizes the expected outcome with respect to the base policy.

Organization: Our paper is organized as follows. We first describe our formulation of the problem and characterize the optimal solution. We review possible approaches and why they are intractable. We then describe our learning algorithm and performance results. Specifically, we show how the expected reward can be computed in closed form, and demonstrate how this enables efficient value function learning. We conclude with a demonstration of our approach on a set of example activity detection problems of up to 1,089 grid cells, and show that we are able to find policies efficiently that are within 1% of optimal.

Problem Model

Our task is to map the activity in an area and sequentially decide where to sense next. We model the environment as an occupancy grid, as shown in Figure 2. An occupancy grid divides the area into discrete locations. The activity is assumed to be located at specific "occupied" grid cells. Without loss of generality, we assume that the activity does not change location, and the sensor can be steered from one aimpoint to any other aimpoint (i.e., the sensor is not constrained to follow a contiguous path).

We model the sensor as a 3×3 grid; that is, when the sensor is aimed at a particular grid cell i, it observes that grid cell and the 8 surrounding cells. Modeling the sensor in this way allows two actions to partially overlap. Therefore, the sensor can plan two actions that deliberately overlap in order to collect more information at the overlapping grid cell. Each of the 9 measurements created by a single sensing action is modeled as an independent Bernoulli distribution, with some probability of detecting the activity of interest and some probability of false alarm.

The stochastic sensor measurements cause uncertainty regarding activity map state. Therefore, we infer the probability of activity in a given cell using Bayes filtering, i.e.

$$p_i = P(i|w_i) = \alpha P(w_i|i)P(i) \tag{1}$$

where p_i represents the posterior probability that a grid cell is occupied, w_i is the sensor measurement for cell i, $P(w_i|i)$ is the sensor model and $P(i)$ is the prior probability of activity at cell i. We refer to the probabilities of the entire grid as a "belief map."

The exploration process of building an accurate activity map works as follows. We take a sensing action, which returns a measurement for each of the 3×3 observed grid cells. For each sensed cell, we then update the belief map p_i using equation (1). We then use the updated belief map to select the next most informative sensing action.

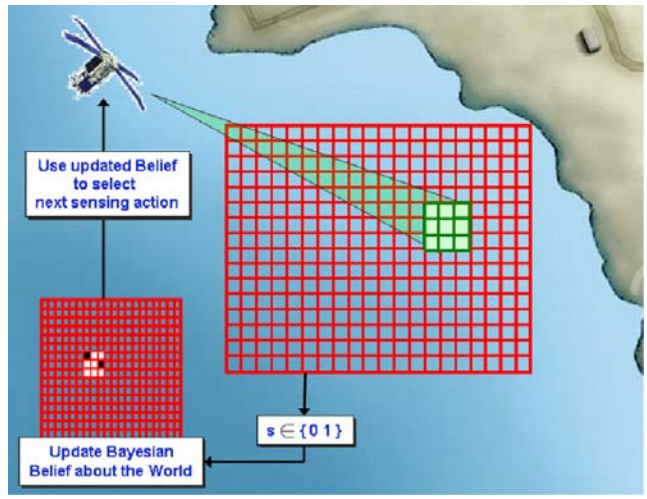

Figure 2: Occupancy grid and Bayesian belief representation of our satellite sensing problem.

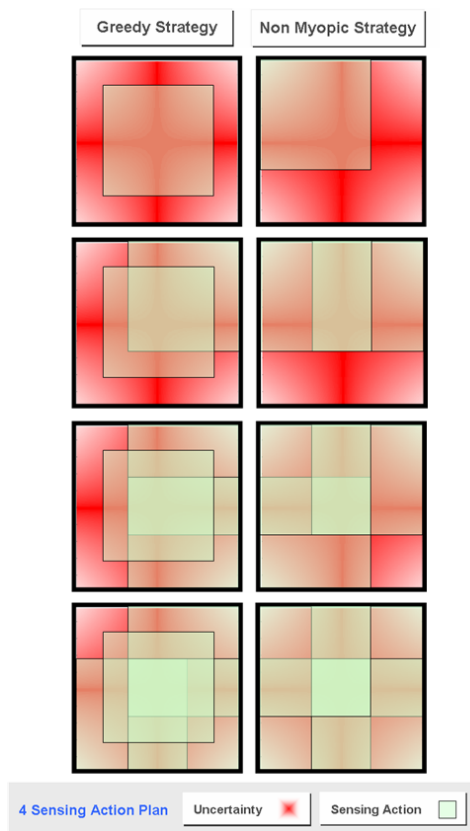

Figure 3: Example comparison of greedy and non-myopic sensing action selection strategies.

Motivation For Our Approach

In this section we motivate our approach by first comparing greedy and non-myopic strategies. We then discuss why a non-myopic open-loop feedback control approach is intractable. Finally, we formulate our problem as a *belief-state* Markov Decision Process and analyze the complexity of finding an optimal control policy.

Greedy Strategy

Our objective is to find the most informative sensing plan, which minimizes map error. Since the true activity map state is unobservable, we can instead minimize the uncertainty of our belief regarding the activity in the area. A common sub-optimal approach to the problem is a greedy strategy, which looks one step ahead and selects the sensing action that reduces uncertainty the most. In the example shown in Figure 3, the boxes represent the map and red indicates the amount of uncertainty. The greedy strategy takes a sensing action right in the middle of the map, where uncertainty is greatest. A non-myopic strategy, on the other hand, considers the entire mission duration, consisting of 4 sensing actions in this example. As the problem proceeds, we can see that the greedy strategy made a poor initial decision, in the end, leaving part of the map completely unobserved. Consequently, the non-myopic strategy leads to reduced map uncertainty over the entire mission duration.

Open-loop Feedback Control

One non-myopic approach to map exploration is open-loop feedback control (OLFC). To select a sensing action, OLFC generates an open loop plan (via search) for the mission duration, which minimizes uncertainty. After the first action is executed and the belief is updated, a new open loop plan is generated for the remaining mission. The process continues for the mission duration.

While OLFC is effective, it is also intractable. For example suppose we have a problem with 1,000 locations and we make 100 sensing actions per mission. In this case, there would be 10^{300} possible plans. We cannot possibly search over that many plans. For example, even if a fast computer could evaluate 1 trillion plans per second and an efficient algorithm could prune away 99.9999% of the plan space, it would still take billions of years to consider the remaining plans. To make matters worse, the detections are probabilistic, which makes evaluating plan outcomes more expensive. Furthermore, we need to update our belief about the world after each sensing action, which requires us to re-plan in real-time, based on our updated belief. As OLFC is far from computationally tractable, the current state of the art uses a randomized greedy strategy (Krause & Guestrin, 2005). Our approach provides a better, non-myopic, solution while maintaining tractability. Next we formulate the problem as a *belief-state* MDP and analyze the complexity of finding an optimal control policy.

Exploration as a *Belief-state* MDP

In this subsection we formulate the previously described occupancy grid model as a *belief-state* MDP, using basic problem notation (Bertsekas, 2005). The underlying imperfect state information problem can be viewed as a perfect state information problem whose probabilistic *belief-state* x is conditioned on the information available (Bertsekas, 2005). The *belief-state* x is comprised of the probability of occupancy of each grid cell. Thus, p represents the probability that a particular grid cell contains the activity of interest. The probability can be any real number between 0 and 1. Thus, the state space is an n-dimensional continuous space.

Following definition 10.7 from Bertsekas & Shreve (1978), a *belief-state* MDP is defined as the tuple $\{X, U, g, f, T\}$, respectively, the state space (X), control actions (U), reward function (g), system dynamics (f) and problem horizon (T). These variables are as follows:

- State space: The *belief-state* $x = [p_1 \ p_2 \ \dots \ p_n]$ is the vector of the probability of activity in each grid cell (p_i), such that $x \in X = [0,1]^n$, where n is the number of cells.
- Control actions: The set of actions is the set of possible sensing locations, $u \in U = \{1, 2, 3, \dots, n\}$, that is, we can select any grid cell as the sensor aim-point, and the sensor will observe that cell and the surrounding cells.
- System dynamics: We assume that each action u generates a set of sensor measurements w,

$$w = \{w_1 \dots w_9\} \quad (2)$$

$$\text{such that } w_j = \{0, 1\} \quad (3)$$

  and each w_j is drawn according to the sensor model $P(w|\rho(j))$, where $\rho(j)$ maps the j-th observation to some grid cell i. The transition from x_t to x_{t+1} is given by

$$x_{t+1} = f(x_t, u, w) \quad (4)$$

  and the state transition probability is the probability of getting sensor measurement w, given our current *belief-state* and control action u.

$$P(x_{t+1}|u, x_t) = P(w|u, x_t) \quad (5)$$

  Thus, the system dynamics are dictated by a Bayesian update given w. For example, we take control action u and get sensor mearurements w; the collection of 9 sensor measurements, one for each observed grid cell. We then perform a Bayesian update according to equation (1) to generate the next state. Hence, the probability of ending up in state x_{t+1} is just the probability of getting sensor measurements w, given the previous state and action u.
- Reward: Our objective is to minimize map error; the difference between our *belief-state* and the truth. Since we cannot compute the actual error, a common approach is to maximize entropy reduction.

$$g(x, u, w) = h(x) - h(f(x, u, w)) \quad (6)$$

  where

$$h(x) = -\sum_i (p_i \log_2 p_i + (1 - p_i) \log_2 (1 - p_i)) \quad (7)$$

  In other words, our reward is the decrease in entropy caused by control action u.

Our task is to develop a strategy (also known as a policy) for selecting control action u, whose outcome depends on sensor return w, which in turn depends on the probability of detection and whether or not a grid cell is occupied. Naturally, we do not know if a grid cell is occupied; however, we maintain a distribution over occupancy, which we call our belief. Therefore, we have a certain expectation over how much information, control action u will provide.

The optimal policy maximizes the expected reward over the entire mission, w.r.t. the stochastic sensor returns.

$$J(x_0) = \max_{u_t} E_{w_t} \left[\sum_{t=0:T-1} g(x_t, u_t, w_t) \right] \quad (8)$$

Naturally, we can formulate the problem as a dynamic program, which maximizes the expected one-stage reward g plus the expected value J (Bellmann, 1957),

$$J(x) = \max_u E_w \left[g(x, u, w) + J(f(x, u, w)) \right]. \quad (9)$$

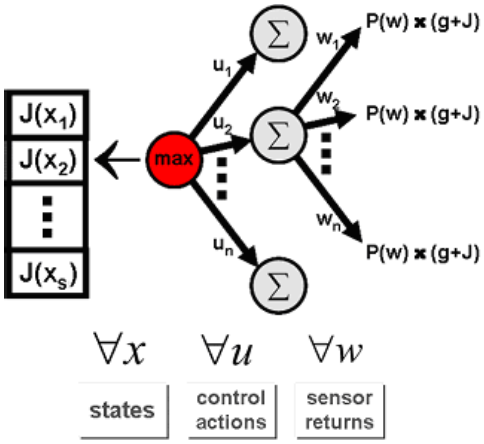

Figure 4: Value iteration backup diagram (discrete state-space problem).

The dynamic programming equation is recursive. However, given the optimal future reward function J, we could compute the optimal control action u. Therefore, knowing the value function J is tantamount to knowing the optimal control policy. Our objective is, therefore, to find the optimal value function J.

Policy Learning Complexity

Unfortunately, solving the described MDP is intractable. To be specific, there are no known methods for solving a continuous *belief-state* MDP exactly. Let us then consider a discrete version of the problem. For example, suppose our problem has 1,000 grid cells and we represent our belief at each grid cell as 10 discrete probability intervals. We would then have a huge number (10^{1000}) of discrete states. The current state of the art for solving discrete MDPs includes two methods (SPUDD and APRICODD) based on Abstract Decision Diagrams (Hoey et al., 1999; St-Aubin et al., 2000) and one method based on Factored MDPs (Guestrin et al., 2003), which can solve problems with 10^{40} states, a small fraction of the size of our problem [1]. Alternatively, our approach is to approximate J as a parametric function, and learn an approximation of J using dynamic programming.

Closed Form Least Squares Value Iteration

We outline our closed form LSVI approach in this section. We begin our discussion by outlining exact discrete value iteration backup. We then discuss computational complexity and our approach to addressing it. Specifically, we detail our function approximation architecture and alternate reward function.

Our main contribution is that we can compute a value iteration backup in closed form (with respect to the current and approximate value functions). We prove our claim with a derivation.

Value Iteration

Our objective is to learn value function J. For a discrete state space, J is represented as a table of values (one record for each discrete state). The table of values can be initialized arbitrarily and improved iteratively. To iteratively improve the value function estimate, for each x, we find control action u that maximizes the expected current reward g plus the expected future reward J (using our current value function table J^i).

$$J^{i+1}(x) \leftarrow \max_u E_w \left[g(x,u,w) + J^i\left(f(x,u,w)\right) \right] \tag{10}$$

The sensor measurements w are stochastic; therefore, the expectation entails summing over the rewards, times their probability of occurring.

$$J^{i+1}(x) \leftarrow \max_u \sum_{w \in \{0,1\}^9} \left(g(x,u,w) + J^i\left(f(x,u,w)\right) \right) P(w) \tag{11}$$

We compute this summation for each control action. We then replace the current table entry $J(x)$ with the new maximum value corresponding to the best control action. The process (a value iteration backup) is shown in Figure 4. The above process constitutes one iteration. We repeat the process until $J(x)$ converges.

The main problem with this approach is that we must perform a value iteration backup for every single state, which would take far too long (i.e. for 10^{1000} discrete states). Furthermore, we cannot store the huge table representation of J in memory, as no computer can hold (10^{1000}) values. We address these complexities by representing J as a parametric function. Furthermore, we employ a better reward function, which serendipitously allows us to compute the summation over w in closed form.

Linear Architecture: Formally, a linear architecture approximates $J(x)$ by first mapping the state x to feature vector $\Phi(x) \in \Re^k$ and by computing a linear combination of those features $\Phi(x)\beta$, where β is a function parameter vector. We compose $\Phi(x)$ by applying basis function $\phi()$ to the probability of occupancy of each grid cell, such that

$$\Phi(x) = \left[\begin{array}{ccc} \phi(p_1) & \ldots & \phi(p_n) \end{array} \right] \tag{12}$$

We also use an intercept that our description ignores for simplicity. We compute an improved value estimate $\hat{J}(x)$ for state sample set X_s via value iteration backups, such that

$$\hat{J}(x) = \max_u E_w \left[g(x,u,w) + J_{approx}\left(f(x,u,w)\right) \right], \tag{13}$$

where $J_{approx}(\cdot) = \Phi(\cdot)\beta$.

We then update our estimate of J_{approx} using regression, where

$$\Phi(X_s) = \left[\begin{array}{c} \Phi(x_1) \\ \vdots \\ \Phi(x_\nu) \end{array} \right] \tag{14}$$

is a set of feature vectors for all state samples and

$$\hat{J}(X_s) = \left[\begin{array}{c} \hat{J}(x_1) \\ \vdots \\ \hat{J}(x_\nu) \end{array} \right] \tag{15}$$

[1] POMDP techniques have analogous complexity issues. Conventional POMDP techniques maintain a large set of α-vectors representing the policy value, and back up each α-vector by computing an expectation over all possible sensor outcomes. It is exactly this computation which we perform efficiently while avoiding the POMDP formulation. Although *approximate* POMDP algorithms have improved dramatically, the exploration problem complexity still precludes using them for all but the simplest problems.

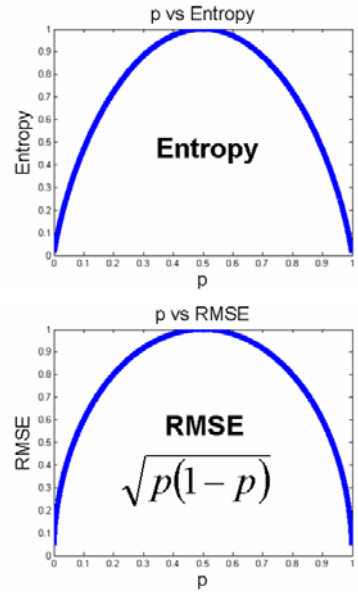

Figure 5: RMSE versus Entropy.

is the updated value estimate for all state samples.

We re-estimate the parameters β of our function as

$$\beta = \left(\Phi\left(X_s\right)^T \times \Phi\left(X_s\right) + \lambda I\right)^{-1} \Phi\left(X_s\right)^T \hat{J}\left(X_s\right) \quad (16)$$

where ridge penalty λ ensures invertiblity and can be increased to reduce estimation variance at the expense of increased bias. Our updated $J_{approx}\left(x\right) = \Phi\left(x\right)\beta$ now estimates our new target value $\hat{J}\left(x\right)$ in a least squares sense.

To summarize, we simulate a next state and plug that state into J_{approx}, our approximate value function. As the sensor measurements are stochastic, we must sum over the rewards of each possible future state. In the next subsections, we show how a better reward function allows us to compute the summation (i.e. expectation) over g in closed form. Furthermore, if we carefully select our features, we can also compute the summation over J_{approx} in closed form. The better reward function we are referring to is root mean squared error (RMSE).

Better Reward Function : Root Mean Squared Error

This subsection motivates why RMSE makes sense as a reward function. Recall that our initial reward function was entropy, which is a measure of uncertainty. Likewise, RMSE is also a measure of uncertainty, specifically it is the standard deviation of a binomial distribution. Our system is well-modeled by a binomial distribution, in that a grid cell is either occupied or unoccupied, and we maintain a distribution over occupancy, called our belief.

Figure 5 shows that RMSE looks very much like entropy; entropy being the traditional measure of uncertainty and the basis of information theory. However, entropy is not directly related to our intended decision. On the other hand, RMSE is directly related to expected map error, computed as the true occupancy $\o \in \{0,1\}$ minus our estimate p squared, multiplied by the probability with respect to occupancy.

$$E[(\o - p)^2] = \sum_{\o \in \{0,1\}} (\o - p)^2 P(\o) = (0-p)^2(1-p) + (1-p)^2 p = p(1-p) \quad (17)$$

Thus, RMSE is not just a reasonable proxy for entropy, it is actually a more appropriate reward as it is directly related to our intended decision. However, the main benefit of using RMSE is its computational properties.

Closed Form Value Iteration Backup

During a value iteration backup, for each control action we compute the expected current plus (approximate) future reward, as a weighted sum over potential sensor outcomes. Using RMSE, rather than Entropy, we can compute the expected current reward g in closed form. Furthermore, if we carefully select the features of our function J_{approx}, we can also compute the sum over J_{approx} in closed form. Therefore, using RMSE, we can compute the entire expectation with respect to w as a single dot product.

Put another way, we avoid the complexity of a stochastic transition function. We instead use a deterministic expression with respect to current and approximate future rewards, which gives us the exact same result. We extend this concept to every control action u and every state sample s, to compute a backup for all samples, states and actions, as a single matrix multiplication. This is a very unusual property for an interesting distribution and useful reward function, which speeds up our computation considerably.

Closed Form E[RMSE]: Here we derive a closed form expression for the expected RMSE, given a set of t sensing actions and our prior belief x. We then show how this general derivation can be applied to a value iteration backup.

We first observe that the expected reward per grid cell is conditionally independent given t. Therefore, from t sensing actions we count the n observations of a particular grid cell due to the overlapping 3×3 footprints. We then compute the E[RMSE] for that cell. We define the following notation:

- n: number of observations at a given cell
- k: number of detections
- $a = P(detection)$
- $b = P(false\text{-}alarm)$
- $p = P(occupancy)$, where

$$P(occupancy|detection) = \frac{P(detection|occupancy)P(occupancy)}{P(detection)}. \quad (18)$$

Thus, a Bayesian update, computing the posterior probability that a cell is occupied given just one single detection $(n = 1, k = 1)$, is expressed using the above notation as:

$$p_{post|n=1,k=1} = \frac{ap}{ap + b(1-p)} \quad (19)$$

Next we will compute a Bayesian update for a particular cell, given a possible *set* of n sensor returns. We observe that the computed posterior is the same regardless of the sensor return order. Thus, the posterior probability that a cell is occupied given k detections (p_{post}) is

$$p_{post} = \frac{(1-a)^{(n-k)} a^k p}{(1-a)^{(n-k)} a^k p + (1-b)^{(n-k)} b^k (1-p)} \quad (20)$$

Similarly, the posterior probability a cell is unoccupied is

$$1 - p_{post} = \frac{(1-b)^{(n-k)} b^k (1-p)}{(1-a)^{(n-k)} a^k p + (1-b)^{(n-k)} b^k (1-p)} \quad (21)$$

Thus, E[RMSE] may be computed as the sum over the posterior RMSE times the likelihood of getting k detections and $n-k$ non-detections, given prior p. We weight the likelihood given occupancy by the prior probability of occupancy (and vice versa), then multiply by the binomial coefficient (the number of k-element subsets of an n-element set) to get

$$E\left[RMSE\right] = \sum_{k=0}^{n} \left\{ \begin{array}{l} \left((1-a)^{(n-k)} a^k p + (1-b)^{(n-k)} b^k (1-p)\right) \\ \times \frac{n!}{k!(n-k)!} \sqrt{p_{post}\left(1-p_{post}\right)} \end{array} \right\} \quad (22)$$

Notice the normalizing denominator of a Bayesian update is also the likelihood of the data given the model. Therefore, when we spell out the posterior RMSE

$$\sqrt{p_{post}\left(1-p_{post}\right)} = \frac{\sqrt{(1-a)^{(n-k)} a^k p (1-b)^{(n-k)} b^k (1-p)}}{\left((1-a)^{(n-k)} a^k p + (1-b)^{(n-k)} b^k (1-p)\right)} \quad (23)$$

we see that its denominator conveniently cancels out the observation likelihood, resulting in:

$$E\left[RMSE\right] = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} \sqrt{(1-a)^{(n-k)} a^k p (1-b)^{(n-k)} b^k (1-p)} \quad (24)$$

The key reason we are able to reduce equation (22) to equation (24) is that both posterior denominators are the same and $\sqrt{d^2} = d$, a reduction is particular to RMSE. We then pull out the prior and reorganize the sensor parameters.

$$E\left[RMSE\right] = \sqrt{p(1-p)} \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} \sqrt{ab}^k \sqrt{(1-a)(1-b)}^{n-k} \quad (25)$$

We then apply the Binomial Theorem:

$$(x+y)^n = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} x^{n-k} y^k, \quad (26)$$

If we let $x = \sqrt{ab}$ and $y = \sqrt{(1-a)(1-b)}$ then

$$E\left[RMSE\right] = \sqrt{p(1-p)} \left(\sqrt{ab} + \sqrt{(1-a)(1-b)}\right)^n \quad (27)$$

Set $c = \sqrt{ab} + \sqrt{(1-a)(1-b)}$ (a constant of our sensor model). Note that equation (27) allows us to compute the expected RMSE reduction (our reward) for a given cell after n measurements. The computation is simply our prior RMSE times $(1-c^n)$. Therefore, when performing a value iteration backup, we can quickly compute the expected current reward for the entire grid by applying $(1-c) \times \sqrt{p_i(1-p_i)}$ at the observed grid cells. Furthermore, we show that, using RMSE as our basis functions, we can compute the sum over J_{approx} in closed form. As it turns out, we also achieve good results by doing so.

Closed Form E[$J_{approx}()$]: $J_{approx}()$ is our approximate value function, and β is the parameter set of our linear function over our features. Our features (or basis functions) are defined as $\phi(\cdot) = \sqrt{\cdot(1-\cdot)}$. Specifically, we compute the RMSE per grid cell of the resulting posterior state. Serendipitously, these features allow us to approximate the value function reasonably well. Furthermore, they allow us to compute the expectation in closed form. We compute $E_w\left[J_{approx}\left(f\left(x,u,w\right)\right)\right]$ on a per-cell basis as follows.

$$\underset{w}{E}\left[J\left(f(p,u,w)\right)\right] = \underset{w}{E}\left[J\left(p_{post}\right)\right] = \underset{w}{E}\left[\beta\phi\left(p_{post}\right)\right] = \underset{w}{E}\left[\beta\sqrt{p_{post}\left(1-p_{post}\right)}\right] = \beta \underset{w}{E}\left[\sqrt{p_{post}\left(1-p_{post}\right)}\right]. \quad (28)$$

As shown above, we can compute $E_w\left[\sqrt{p_{post}\left(1-p_{post}\right)}\right]$ in closed form as $c \times \sqrt{p(1-p)}$. Consequently, we can combine the closed form expectation of $g + J_{approx}$ into a single expression, such that

$$\sqrt{p(1-p)} + E_w\left[-\sqrt{p_{post}(1-p_{post})} + \beta\sqrt{p_{post}(1-p_{post})}\right] \quad (29)$$

$$\begin{aligned} &= \sqrt{p(1-p)} + -c\sqrt{p(1-p)} + c\beta\sqrt{p(1-p)} \\ &= \sqrt{p(1-p)}\left((1-c+\beta c)\right) \\ &= \phi(p)\left((1+c(\beta-1))\right) \end{aligned}$$

The above expressions are given for a single grid cell. We can compute a backup for all grid cells, given action u, as a dot-product by representing $\Phi(x)$ as a row vector and by constructing a column vector, containing a 1 for each unobserved grid cell and $\sqrt{ab} + \sqrt{(1-a)(1-b)}$ for each observed grid cell. Thus we can concatenate the multiplier vectors $\forall u$ into a single square matrix C and compute an entire backup $\forall u$ as a max over a single dot product:

$$\hat{J}(x) = \max \Phi(x)^T \left((1-C) + (\otimes\beta)\right) \quad (30)$$

where $\otimes$ is an element-wise multiplication. Finally, if we concatenate our feature vectors vertically, we can compute an entire backup for all *belief-state* samples X_s as a max over a single matrix multiplication (note addition of γ).

$$\hat{J}(X_s) = \max_{rows} \Phi(X_s)^T \left((1-C) + (C \otimes \beta\gamma)\right) \quad (31)$$

For additional efficiency, we approximate the finite stage problem as an infinite horizon problem with discount rate $\gamma = 1/T$, where T is the mission horizon. The mean of a geometric distribution with success probability $1/T$ is T. Thus, the discount rate was chosen such that the equivalent stochastic shortest path problem would have an average of T stages or actions. The infinite horizon approximation allows us to compute just one value function J_{approx}. Our results show that the approximation is effective. Algorithm 1 shows the closed form LSVI pseudo-code.

Algorithm 1 Closed Form Least Squares Value Iteration

Input: $\{X, U, g, f, T\}$
Initialize $\beta = [0...0]^T$
Construct C
repeat
  $X_s = samples(X)$
  $\hat{J}(X_s) = max_{rows}\Phi(x)\left((1-C) + (C \otimes \beta \times \gamma)\right)$
  $\beta = \left(\Phi(X_s)^T \Phi(X_s) + \lambda I\right)^{-1} \Phi(X_s)^T \hat{J}(X_s)$
until near-convergence
$return(\beta)$

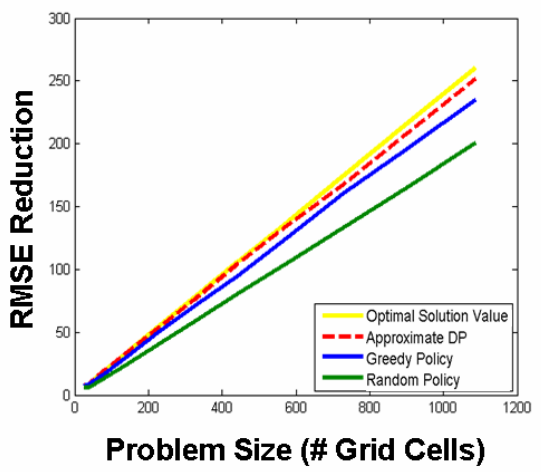

Figure 6: Un-normalized results, Least Squares Value Iteration (approximate dynamic programming) approach.

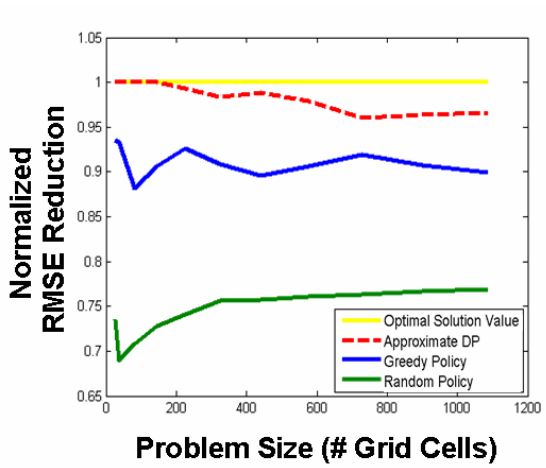

Figure 7: Normalized results, Least Squares Value Iteration (approximate dynamic programming) approach (red dotted line). Performed optimally up to problem size ~ 200. Performance tapers and levels off.

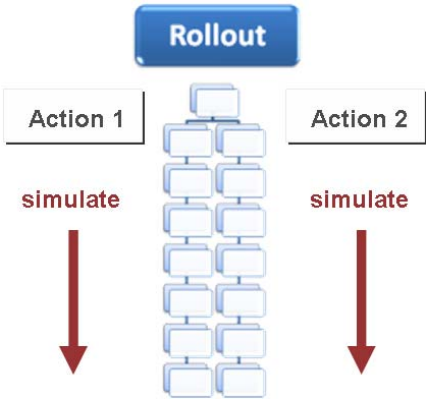

Figure 8: Rollout considers each action and simulates where the approximate policy will lead.

Algorithm 2 Rollout Algorithm

Input: $\{X, U, g, f, T\}, x_t$,base-policyμ_β

$\tilde{J}_{u_t}(x_t) = \underset{w_t}{\mathrm{E}}\left[g(x_t, u_t, w_t)\right]$

$+ \underset{w_k}{\mathrm{E}}\left[\sum_{k=t+1:T-1} g(x_k, \mu_\beta(x_k), w_k)\right]$

$return(argmax_u(\tilde{J}_u(x_t)))$

LSVI Results

Our results demonstrate the significant benefit of our approach. We tested our method on problems with uniform prior beliefs (see Appendix A for details). Our method scaled from 25 grid cells to 1,089 grid cells. We measure performance in terms of RMSE reduction over the mission horizon. Our method is shown in Figure 6 as the red dotted line. As one can see, our method performed better than greedy (blue line) on all problems. We can normalize the results in order to inspect them more closely. As shown in Figure 7, our method performed optimally up to problem size ~ 200. The performance then tapers gracefully and levels off. These initial results are very encouraging; however, we are able to improve upon them using rollouts.

LSVI + Rollout Results

Rollout is a method for extracting a policy at runtime, which uses some additional computation but tends to make better action selections (Bertsekas & Castanon, 1999). For the full details of rollout, see (Bertsekas, 2007). Here we provide an intuitive explanation. Typically, we determine the policy at runtime by finding the control action u which maximizes the dynamic programming equation (13), with respect to our approximate value function J_{approx}. On the other hand, the rollout algorithm selects the control action that maximizes the expected outcome with respect to a multi-stage rollout of the base policy. A multi-stage rollout computes (via simulation) the expected outcome of the base policy over the remaining mission horizon (Algorithm 2). In other words, rollout maximizes over the initial action choice based on where the base policy will lead (see Figure 8). By doing so, rollout makes better control action selections. Rollout is somewhat expensive for stochastic systems (Bertsekas & Castanon, 1999); however, our closed form expected reward expression makes rollout quick.

Figure 9 shows our results using rollout, indicated by the magenta dotted line. As shown, our closed form LSVI plus rollout performed within 1% of optimal on all problems. These near-optimal results are much better than expected and leave little room for improvement. Our closed form version of LSVI + rollout generated the results efficently on problems where all other exact methods would never finish.

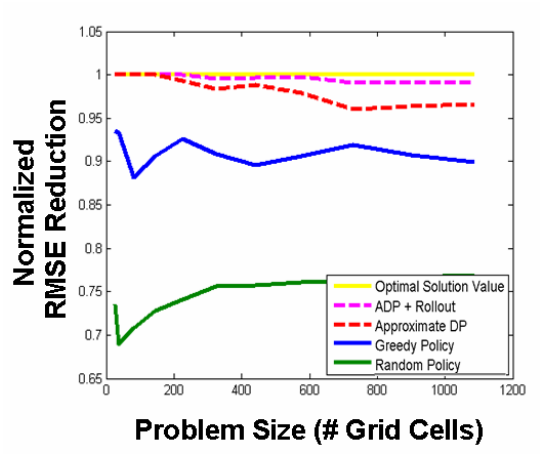

Figure 9: Normalized results: LSVI + Rollout (magenta dotted line) performed within 1% of optimal on all problems.

Contributions

Current methods employ greedy sensing action selection. The non-mypoic approach has been avoided as no exact or approximate MDP solver has been shown to work on relevant problem sizes in this domain. Our contribution is a *belief-state* MDP sensor tasking problem formulation and an efficient function approximation dynamic programming scheme for control policy learning. The result is efficient non-myopic sensing action selection.

Our key innovation, which makes function approximation dynamic programming efficient, is our derivation of a closed form expected reward expression for the alternate reward function RMSE. Likewise, we enable a closed form value iteration backup by selecting basis functions for our value function architecture, which coalesce with the current reward function. We further improve the performance of the resulting policy using rollout. Our method solved large state space problems within 1% of optimal, garnering significant improvement over the state-of-the-art of both optimal exploration and dynamic programming.

References

Bellmann, R. (1957). *Dynamic programming*. NJ: Princeton University Press.

Bertsekas, D. P. (2005). *Dynamic programming and optimal control vol. i, 3rd ed.* Cambridge, MA: MIT Press.

Bertsekas, D. P. (2007). *Dynamic programming and optimal control vol. ii, 3rd ed.* Cambridge, MA: MIT Press.

Bertsekas, D. P., & Castanon, D. A. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics, 5*, 89–108.

Bradtke, S. J., & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Mach. Learn., 22*, 33–57.

Burgard, W., Fox, D., & Thrun, S. (1997). *Active mobile robot localization* (Technical Report IAI-TR-97-3).

Cassandra, A. R., Kaelbling, L. P., & Kurien, J. A. (1996). Acting under uncertainty: discrete bayesian models for mobile-robot navigation. *In Proceedings of the International Conference on Intelligent Robots and Systems*.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research, 19*, 399–468.

Guestrin, C., Krause, A., & Singh, A. P. (2005). Near-optimal sensor placements in gaussian processes. *ICML '05: Proceedings of the 22nd international conference on Machine learning* (pp. 265–272). New York, NY, USA: ACM.

Hoey, J., St-aubin, R., Hu, A., & Boutilier, C. (1999). Spudd: Stochastic planning using decision diagrams. *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (pp. 279–288). Morgan Kaufmann.

J. Boyan, A. M. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Tesauro, Touretzky and Leen edition of Advances in Neural Information Processing Systems*.

Krause, A., & Guestrin, C. (2005). Near-optimal nonmyopic value of information in graphical models. *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)* (pp. 324–33). Arlington, Virginia: AUAI Press.

Monterey Bay Aquarium Research Institute (2006). *Autonomous ocean sampling network*. http://www.mbari.org/aosn/default.htm.

Roy, N., Choi, H.-L., Gombos, D., Hansen, J., How, J., & Park, S. (2007). Adaptive observation strategies for forecast error minimization. *Lecture Notes in Computer Science*.

Sim, R., & Roy, N. (2005). Global a-optimal robot exploration in slam. *Proceedings of the International Conference on Robotics and Automation*. Barcelona, Spain.

St-Aubin, R., Hoey, J., & Boutilier, C. (2000). APRICODD: Approximate policy construction using decision diagrams. *NIPS* (pp. 1089–1095).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Belmont, MA: Athena Scientific.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*.

Appendix A

Test Problems: It is important to measure the optimality gap reduction of our approach with respect to a greedy approach. Yet it is generally impossible to know the optimal solution value. In order to test for optimality we used a unique problem instance, whose optimal solution value can be calculated via Lagrange relaxation with our closed form $E[RMSE]$ expression.

Our test problems have **uniform** .5 prior beliefs for all grid cells and each problem allows n sensing actions which can evenly cover the map. For example, we tested a 33×33 problem allowing 121 sequential sensing actions. Thus, the optimal open loop plan is to evenly tile the sensing area with 11 rows of 11 sensor footprints. We can prove this property via Lagrangian relaxation. General problems with unequal priors and an arbitrary number of sensing actions do not have an elegant solution. Our policies were trained on arbitrary problems. They performed well on both arbitrary and special case problem, with respect to the greedy solution. However, we can only measure the optimality gap for our special problems.

Lagrange Relaxation: We solve the test problems using Lagrange relaxation, by relaxating the sensor footprint and integrality constraints. Thus, the relaxed problems allow a single sensing action to cover non-adjacent cells as well as a non-integer number of sensing actions per cell. The Lagrangian relaxation solutions satisfy the original constraints, thus are optimal. Note that our test problem is a special case, with solution properties which do not hold for virtually every other instance. Consequently, the general problem class requires our more sophisticated policy based approach. Yet the test problems exhibit all of the relevant complex decisions of the general problem class. Consequently, our test problems offer a valid solution optimality comparison.