Accepted for publication in the Proceeedings of the Seventeenth International Conference on Machine Learning (ICML-2000), Stanford University, California, June 29–July 2, 2000.

A Bayesian Framework for Reinforcement Learning

Malcolm Strens MJSTRENS@DERA.GOV.UK
Defence Evaluation & Research Agency. 1052A, A2 Building, DERA, Farnborough, Hampshire. GU14 0LX. U.K.

Abstract

The reinforcement learning problem can be decomposed into two parallel types of inference: (i) estimating the parameters of a model for the underlying process; (ii) determining behavior which maximizes return under the estimated model. Following Dearden, Friedman and Andre (1999), it is proposed that the learning process estimates online the full posterior distribution over models. To determine behavior, a hypothesis is sampled from this distribution and the greedy policy with respect to the hypothesis is obtained by dynamic programming. By using a different hypothesis for each trial appropriate exploratory and exploitative behavior is obtained. This Bayesian method always converges to the optimal policy for a stationary process with discrete states.

1. Introduction

Reinforcement learning (RL) is a form of machine learning used to solve problems of *interaction* (Bertsekas & Tsitsiklis, 1996; Kaelbling, Littman & Moore, 1996; Sutton & Barto, 1998). It is a form of trial-and-error learning; an agent starts interacting with the environment with an arbitrary (random) policy for choosing control actions. The agent receives *rewards* when these actions lead to successful performance of the task. As the agent explores the environment and finds routes to high reward its behaviour changes from (near-random) *exploration* to (near-deterministic) *exploitation*. Thus, reinforcement learning acquires a model for the relationship between the state of the environment, the available actions, and the rewards that accrue over a period of time.

To make the decision as to what action to take in a particular state, an agent can draw on previous experience and take the action which, on average, led to the better reward. This "greedy policy" fails when rewards are uncertain; the system can converge to a local minimum where the greedy behavior is sub-optimal because its decisions are based on maximum likelihood estimates (e.g. averages) which are not equal to the true parameters of the underlying process (e.g. the true means of the rewards associated with different actions). The conventional approach to solving this problem is to introduce occasional random actions, or some other way of enhancing exploration (Sutton & Barto, 1998). A theoretically justifiable approach is to retain a notion of uncertainty in the model parameters and to take decisions according to hypotheses for the true model parameters. This cannot be achieved for primitive RL algorithms which are model-free; however interval estimation Q-learning and several other techniques explicitly represent uncertainty in the discounted return. These techniques were compared by Dearden, Friedman and Russell (1998) which forms a baseline for this work.

Consider the case where there *is* an explicit model, and the goal is to represent the uncertainty *in the parameters of this model*, rather than in the discounted returns. This has also been proposed by Dearden et al. (1999). For infinite horizon problems, the Bayesian optimal solution for action selection requires calculation of an intractable integral over courses of action (Martin, 1967). Even the optimal 1-step exploitative action is difficult to compute: it is the action with the highest *expected* return under the posterior distribution on models. Calculating this expectation requires integration over the posterior, which is also prohibitory for all but the most simple models. Therefore approximation methods are required.

Dearden et al. (1999) review several sampling approaches for achieving this approximation and use a myopic value of perfect information (VPI) criterion for action selection. I propose a different and simpler scheme which is similar to their global importance sampling method. However, in their work, resampling is performed as often as possible. In contrast, I argue that preserving the same hypothesis for several time steps has a very important advantage (apart from reducing computation cost): *exploration strategy will be consistent over a period of time*. This observation is intuitive in that human learning operates this way – a hypothesis can be tested through taking a coherent course of action, before generating a new hypothesis. The approach also requires only one model hypothesis to be generated per trial and allows large problems to be solved. A further speed-up is obtained by initializing the value estimates of sampled models using the maximum likelihood solution as a starting point.

As an example, suppose a decision-making system must (repeatedly) choose between two actions which lead to unknown rewards (the 2-armed bandit problem). It could generate a random hypothesis of the form "action 1 is best". If prior observations *are* available, then a hypothesis "action 1 leads to expected reward *A* and action 2 leads to expected reward *B*" can be generated in an unbiased fashion (by obtaining a sample of size 1 from the posterior distribution on the true values of expected rewards). A greedy policy with respect to this joint hypothesis selects action 1 when $A > B$. After taking this action, useful information will be gained, even if it is not the true optimal action. A new hypothesis is drawn from the updated posterior distribution before each action is selected. Choosing actions according to unbiased hypotheses from the posterior ensures both actions will occasionally be taken and will eventually lead to optimal behavior as the distribution collapses to a point over time. (*A* and *B* approach the true expected rewards.) Hence there is a natural way of smoothly moving from exploratory to exploitative behavior. The same principle is applicable to models for extended sequences of interaction involving *state* and *stochastic* transitions, such as Markov decision processes (MDPs).

The remainder of this paper shows how this is achieved. Section 2 introduces RL terminology, primitive learning techniques, and defines the MDP model. Section 3 shows that online dynamic programming can be used to solve the reinforcement learning problem and describes heuristic policies for action selection. Section 4 shows how to represent the prior and posterior probability distributions for MDP models, and how to generate a hypothesis from this distribution. Section 5 describes the proposed algorithm and its implementation. This is compared with some existing methods in section 6. Section 7 identifies further avenues for research and draws conclusions.

2. Primitive Reinforcement Learning

Q-learning (Watkins, 1989) is a widely used reinforcement learning technique, and is very simple to implement because it does not distinguish between "actor" and "critic". *i.e.* the same data structure is used to select actions as to model the benefits of courses of action. Q-learning is described here in order to introduce several important concepts of reinforcement learning and to provide a baseline with which to compare dynamic programming approaches. Like many reinforcement learning algorithms, Q-learning aims to maximize *discounted return*. The discounted return is a sum, from the current time until the end of the trial, of rewards received, but each reward is *discounted*; i.e. rewards received sooner are more significant. Formally, if r_t is the reward received at time *t* and g is the discount rate, the discounted return is given by:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

The quality function, $Q(s,a)$, is defined as the discounted return when action *a* is taken in state *s*, assuming an optimal policy is taken thereafter. If $Q(s,a)$ can be estimated by learning, then the optimal policy in state *s* is to choose the action which maximizes $Q(s,a)$. Q-learning works by keeping running estimates that are updated at each step or at the end of a trial. When action *a* in state *s* leads to state s' with instantaneous reward *r*, the *Q-learning rule* updates the existing estimate for $Q(s,a)$ as follows:

$$\hat{Q}(s,a) \leftarrow (1-\alpha)\hat{Q}(s,a) + \alpha\left(r + \gamma \max_{a'} \hat{Q}(s',a')\right)$$

This is a linear combination between the previous estimate, and an estimate obtained from the most recent observation. The learning rate α determines the proportions in this combination, and serves to average over several forms of uncertainty. These uncertainties are (i) the stochastic nature of state transitions; (ii) the stochastic nature of immediate rewards; (iii) the error in the current estimates of *Q*. The learning rate determines a compromise between likelihood (and accuracy) of convergence and the number of trials required. Q-learning is very effective if a large number of trials can be performed, but much faster learning can be obtained if a more explicit mathematical model of the agent-environment interaction is utilized. The most common model is a Markov decision process.

A Markov decision process (MDP) or *controlled* Markov system (Bellman, 1957a) is used to model an interactive system with evolving state. It is defined by a quadruple (S,A,T,R). S and A are discrete sets of states and actions. T is a stochastic state-action transition function. For each state-action pair (s,a), the reward $R(s,a)$ is a real-valued random variable. At any discrete time *t*, the system state is X_t and the state at the next time step, X_{t+1}, is given by probabilistic transitions determined by T and an action Y_t received by the system from an external agent:

$$T(s,a,s') = P\left(X_{t+1} = s' \,\middle|\, X_t = s, Y_t = a\right)$$

When this transition takes place, the process emits a scalar reward, generated from $R(s,a)$.

The MDP model allows an explicit expression for $Q(s,a)$:

$$Q(s,a) = E[R(s,a)] + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a') \qquad (1)$$

where $E[\,]$ is the expectation operator.

3. Dynamic Programming for RL

Dynamic programming (DP) is a means of solving constraint satisfaction problems through repeated resubstitution of estimates. It is particularly relevant for estimating quantities associated with the nodes of a graph such as the expected discounted reward in the RL problem (Bellman, 1957b; Bertsekas, 1995). DP will be

used here for two different purposes. Firstly, it will be used to solve for the optimal policy under the maximum likelihood MDP parameters (expected rewards and transition probabilities). Secondly it will be used to solve for the optimal policy under each hypothesis drawn from the posterior distribution on MDP model parameters.

3.1 Dynamic Programming on a MDP

If the reward distribution and state transition probabilities are known then equation 1 yields a set of simultaneous nonlinear equations in Q. (One equation for each state-action pair.) The DP value iteration technique solves this set of equations directly by repeated substitution of estimates. Once Q is known, the greedy policy in state *s* is to choose the action *a* for which $Q(s,a)$ is maximum. However, the reward distribution and transition probabilities are not known in many scenarios, and so must be estimated online, while the agent explores the environment. The maximum likelihood estimate of the transition probability $T(s,a,s')$ is the proportion of times that action *a* in state *s* led to state s' (*e.g.* Barto, Bradtke and Singh, 1995). The maximum likelihood estimate of $E[R(s,a)]$ is the average of the rewards received when action *a* was taken in state *s*. Hence, dynamic programming provides a solution to the reinforcement learning problem without the need for a learning rate. A drawback to the DP approach is that it requires an assumption that the underlying reward distributions and transition probabilities are statistically stationary. The implication of this assumption is that retraining of the system is required whenever the environment changes significantly.

The DP solution method requires the storage of very large structures of sparse data. In particular, DP must keep track of the number of transitions between every pair of states, for every possible action. Most of the counts are zero, and the use of hash tables ensures that the space required is proportional to the number of non-zero counts (while retaining random access). This means that it is possible to work with systems having several thousand states. The DP solution method is made efficient by avoiding unnecessary iteration of the DP "backup" operation which propagates estimates of Q through the MDP (equation 1). This is achieved by the prioritized sweeping algorithm (Moore & Atkeson, 1993; Andre, Friedman & Parr, 1998) which updates only the estimates likely to have changed the most after each observation.

3.2 Heuristic Exploration Policies

With most reinforcement learning methods, including Q-learning and online dynamic programming, there remains a problem of finding an appropriate compromise between exploratory and exploitative behavior. In both cases, the *greedy* policy is to choose the action which yields the highest *expected* discounted return in each state. However, if this action is not the true best action, then the greedy policy can lead to learning becoming "stuck" in a sub-optimal local maximum.

Many alternative policies have been proposed to mix exploratory behavior with this exploitative behavior in order to find new routes to reward or become more confident about which policy is best (Kaelbling, Littman & Moore, 1996; Thrun, 1992). The *semi-uniform* policy chooses mostly greedy actions, but selects a random action with a small probability. The Boltzmann policy chooses actions according to a stochastic function of their associated expected rewards. It has a "temperature" parameter which can be reduced during learning, to move from random (exploratory) behavior to deterministic behavior over the course of learning, in a way analogous with simulated annealing. Another policy (Strens, 1999) chooses the *least taken* action in a given state with a probability inversely proportional to a polynomial function of the number of times that the *least taken* action has been taken; otherwise the greedy action is chosen. Kearns and Singh (1998) have proved convergence bounds for a similar policy, polynomial in a measure of the discounting horizon. Fiechter (1997) applied the PAC ("probably approximately correct") method to prove convergence of another algorithm.

The use of these heuristic (*i.e.* rule-based) policies is often very effective, but it is the aim of this work to eliminate the need for such design decisions and to automatically obtain a policy which gives the appropriate mix of exploration and exploitation.

4. Representing MDP Posterior Distribution

I show how to represent the posterior probability density (i.e. posterior distribution) for the MDP model parameters. These parameters fully define the MDP (apart from fixing the sets of states and actions), and so this distribution is defined over the space of MDPs (or "version space") for this machine learning task (Mitchell, 1997). The posterior distribution must represent uncertainty in transition probabilities and in reward distributions, for each state-action pair. The MDP has the Markov property, so each of these distributions is independent. (The reward and transition probabilities associated with a state-action pair are independent of the sequence of states, actions and rewards which led to that state.) The rewards are assumed to be independent of transitions. (This constraint is not valid in state-value models which require an alternative problem formulation.) Hence, the posterior distribution is represented independently for each state-action pair (s,a) as a distribution of immediate reward distribution parameters and a distribution over the transition probability vector $\boldsymbol{p}$ defined as:

$$\boldsymbol{p} \equiv \left(T(s,a,s_1), \ldots, T(s,a,s_N)\right)$$

This vector has one element for each possible successor state, so its maximum length is equal to the total number

N of states in the system. Note that $|\boldsymbol{p}|=1$ and $\boldsymbol{p}$ represents the parameters of a multinomial distribution.

4.1 Prior Probability Density over MDP Parameters

In order to regularize the learning problem, it is desirable to choose a prior probability density for the parameters of the MDP.

There are two possible types of prior distribution for the transition probability vector $\boldsymbol{p}$.

(i) The first option is to make no major assumptions and to use a uniform or Dirichlet distribution over allowable values as the prior. Friedman and Singer (1999) show that a Dirichlet distribution can be used as an incremental parametric posterior for multinomial probabilities.

(ii) The second option is to assume that the transition matrix is sparse. Only a certain number z of the elements of $\boldsymbol{p}$ are non-zero (i.e. some successor states are unreachable). A hierarchical version of the Dirichlet formalism can be used to account for uncertainty about reachability (Friedman & Singer, 1999).

I choose the latter option, but introduce a stricter form of regularization which reduces the posterior itself to be a (sparse) multinomial distribution. For each state-action pair, a lower bound ($z = z_{\min}$) on the value of z is known throughout processing, because any successor state with a non-zero transition count is reachable. Observe that z places an ordering on the complexity of the distributions of $\boldsymbol{p}$. In obtaining the posterior distribution for $\boldsymbol{p}$, I will allow only $z = z_{\min}+1$[1]. Note that $z_{\min}$ will change dynamically as learning proceeds. (Alternatively, if reachable successor states are known *a priori* then z should be limited to this number.) Of the $z_{\min}+1$ successors, all but one are fixed (they are the states with non-zero transition counts), and the last one is chosen uniformly over the remaining states. Thus there are $N - z_{\min}$ possible hypotheses for the set of successor states. After this choice, a uniform prior can be used for the non-zero elements of $\boldsymbol{p}$ (with the constraint $|\boldsymbol{p}|=1$).

There are many possibilities for the model for immediate rewards. Here I consider only the Gaussian case. It is important to choose an appropriate prior on the standard deviation (s) because a uniform prior would lead to asymptotic behavior of likelihood at $s = 0$.

Defining $y = 1/s$, I choose a prior probability density:

$$f(y) \propto y \exp\left(-y^2 s_0^2/2\right)$$

This has a maximum at $s = s_0$. A prior probability density on the mean (m) is given by:

$$f(m) \propto N\left(m_0, s^2\right)$$

The constants (m_0, s_0) incorporate prior knowledge about likely immediate reward distributions. However, m_0 serves only for initialization; its value is given no weight once the first observation has been made.

4.2 Bayesian Posterior Probability Density

The posterior probability density for model parameters is obtained by combination between the prior density and the evidence available from observations. Let $\boldsymbol{n}$ be the actual observation counts for the events represented by the elements of $\boldsymbol{p}$. i.e. n_i is the number of times that successor state s_i has been reached for the state-action pair we are working with. Applying Bayes' Theorem yields the posterior for $f(\boldsymbol{p})$:

$$P(\boldsymbol{p}|\boldsymbol{n}) = \frac{P(\boldsymbol{n}|\boldsymbol{p})P(\boldsymbol{p})}{P(\boldsymbol{n})}$$

Restricting the set from which $\boldsymbol{p}$ is drawn at every time step (as described in section 4.1) the prior $P(\boldsymbol{p})$ over this set is uniform throughout processing. Hence, ignoring constants, the posterior is a multinomial distribution given by:

$$f(\boldsymbol{p}) \propto \prod_{1 \le i \le N} p_i^{n_i}$$

The posterior density for the immediate reward distribution parameters can be obtained by a similar application of Bayes' Theorem, in terms of a distribution for y and a conditionally independent distribution for m. The posterior distribution for y is given by:

$$f(y) \propto y^{n-1} \exp\left(-y^2\left(nd^2 + s_0^2\right)/2\right)$$

after n observations of immediate reward, with sample variance d^2. Then the posterior on m is:

$$f(m) \propto N\left(\bar{x}, s^2/n\right)$$

where $s = 1/y$. (When $n = 0$, the prior is used.) Thus m depends on y, but the converse is not true; this indicates the order in which these two quantities must be sampled.

4.3 Generating a Hypothesis for the MDP

The proposed algorithm requires an unbiased hypothesis to be obtained from the posterior distribution over MDP parameters at the start of each trial or at fixed intervals during learning. (An unbiased hypothesis is equivalent to an unbiased sample of size 1.) Values of s, m and $\boldsymbol{p}$ must be generated for each state-action pair. Sampling $\boldsymbol{p}$

---

[1] This greatly simplifies the posterior density but restricts it to a space which may not contain the true posterior. However this space will expand to contain the true posterior as learning proceeds and the lower bound on z increases. If the true value of z is equal to $z_{\min}$ then learning also converges to the exact solution with one of the non-zero transition probabilities asymptotically approaching zero.

can be decomposed into a chain of conditionally independent sampling operations, one for each element of $\boldsymbol{p}$. m is sampled from a Gaussian distribution. Samples of s and $\boldsymbol{p}$ are obtained by mapping the uniform distribution $U[0,1]$ through cumulative probability density functions obtained by adaptive numerical integration. For numerical reasons, the probability density of the dimensionless quantity given by equation 2 can be integrated to sample s. (The full details of the sampling of $\boldsymbol{p}$ are not given here.)

$$1/\left(1+y^{2} s_0^2\right) \tag{2}$$

5. Overview of the New Learning Method

I have described how the agent can (i) solve for the optimal policy of a particular MDP using dynamic programming; (ii) estimate the posterior distribution over MDP process parameters during learning; (iii) sample from this distribution to obtain a particular example (or hypothesis) for an MDP which could explain the set of observations made so far. These components are now combined into an algorithm for reinforcement learning. The algorithm must yield an appropriate policy for action-selection at each interaction step. Quantities associated only with efficient implementation are ignored in this overview.

5.1 Maximum Likelihood MDP

The following quantities are accumulated or calculated for each state, and updated (if necessary) after each learning step:

A1. Visit count.
A2. ML[2] estimate of discounted return $\hat{V}(s)$.
A3. List of allowable successor states.

The following quantities are associated with state-action pairs and also kept up-to-date:

B1. Visit count.
B2. Sum of immediate rewards.
B3. Sum of squares of immediate rewards.
B4. List of known successor states.
B5. Transition counts $\boldsymbol{n}$.
B6. ML estimate of expected immediate reward $\hat{m}$.
B7. ML estimate of transition probabilities $\hat{\boldsymbol{p}}$.
B8. ML estimate of discounted return $\hat{Q}(s,a)$.

The ML estimates are kept approximately up-to-date using a few iterations of the prioritized sweeping algorithm at each step. (These ML estimates will be used to reduce the computational cost of calculating discounted returns for each MDP hypothesis.)

[2] *ML* is shorthand for *Maximum Likelihood*.

5.2 Hypothesis Generation

The agent must generate a new hypothesis on a regular basis. For learning in trials of finite length, this is done at the start of each trial. (Otherwise, it takes place every *N* steps, where *N* is related to the number of state transitions the agent is likely to need to plan ahead.) The hypothesis is generated (as described in section 4.3) and the following additional quantities are stored with each *state-action* pair:

B9. Hypothesis for transition probabilities $\boldsymbol{p}^{*}$.
B10. Hypothesis for expected immediate reward m^{*}.

To obtain greedy behavior with respect to this hypothesis rather than the maximum likelihood values ($\hat{\boldsymbol{p}}$ and $\hat{m}$), the following quantities must also be calculated.

For each state:

A4. Discounted return for current hypothesis $V^{*}(s)$.

For each *state-action* pair:

B11. Discounted return for current hypothesis $Q^{*}(s,a)$.

These quantities are approximated by application of prioritized sweeping, using the hypothesized MDP parameters, after initializing them with the maximum likelihood estimates (which greatly reduces computational cost.) The greedy action $\hat{a}$ in state *s* is then given by:

$$\hat{a} = \arg\max_a \left(Q^{*}(s,a)\right)$$

6. Experimental Comparison

Three standard problems with discrete state spaces (Dearden, Friedman, Russell, 1998) were simulated and learning performance was compared using the following methods:

(i) Q-learning with a semi-uniform policy.

(ii) Q-learning with a Boltzmann policy.

(iii) Interval Estimation Q-learning 'plus' ("IEQL+").

(iv) Bayes 'value of perfect information' with mixture updating ("Bayes VPI+Mix").

(v) Dynamic programming with a heuristic policy.

(vi) The proposed algorithm ("Bayesian DP").

In (i) the learning rate and semi-uniform mixing coefficient were decayed uniformly during learning (from 0.02 to 0.00). Methods (ii)-(iv) are primitive learning and local-sampling methods tested by Dearden, Friedman and Russell (1998). Their results (with tuned parameters) are shown here for comparison. In (v), the action selection policy is that of Strens (1999) mentioned in section 3.2, with the probability of choosing the least-taken action a_L given by $4/(4+n^2)$, in which *n* is the number of times that action a_L has been tried previously.

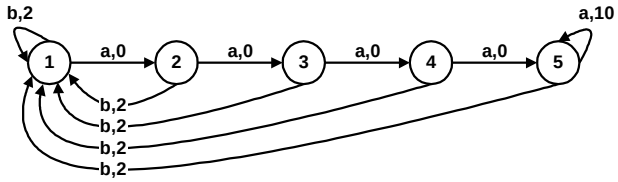

Figure 1. The "Chain" problem

6.1 Problem Descriptions

Figure 1 shows the 5-state "Chain" problem. The arcs are labeled with the actions that cause that state transition, and the associated rewards. However the agent has only abstract actions $\{1,2\}$ available. Usually abstract action 1 causes real-world action a to take place, and abstract action 2 causes real-world action b. With probability 0.2, the agent "slips" and its action has the opposite effect. The optimal behavior is to always choose action 1 (even though this sometimes results in the transitions labeled with b). Once state 5 is reached, a reward of 10 is usually received several times before the agent slips, and starts again at state 1. This problem requires effective exploration and accurate estimation of discounted reward.

Figure 2 shows the "Loop" problem which involves two loops of length 5 joined at a single start state. Two actions are available and transitions are deterministic. Taking action a repeatedly causes traversal of the right loop, yielding a reward of 1 for every 5 actions taken. Conversely, taking action b repeatedly causes traversal of the left loop, yielding a reward of 2 for every 5 actions taken. This problem requires a difficult compromise between exploration and exploitation.

Figure 3 shows the "Maze" problem. The agent can move left, right, up or down by one square in the maze. If it attempts to move into a wall, its action has no effect. The problem is to move from the start (top-left) to the goal (top-right) collecting the flags on the way. When it reaches the goal, the agent receives a reward equal to the number of flags collected, and is returned to the start immediately. The problem is made more difficult by assuming that the agent occasionally "slips" and moves in

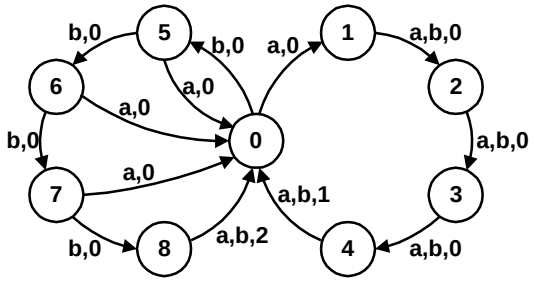

Figure 2. The "Loop" problem.

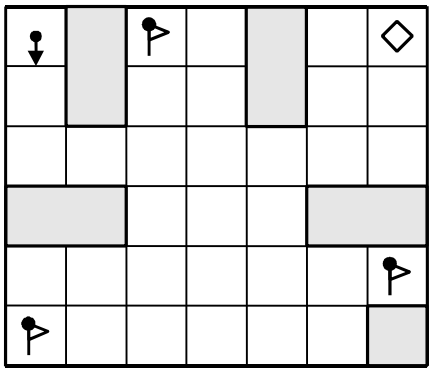

Figure 3. The "Maze" problem.

a direction perpendicular to that intended (with probability 0.1). There are 33 reachable locations in the maze (including the goal) and there are up to 8 combinations for status of the flags at any time. This yields 264 discrete states. The agent was given limited layout information (identifying the immediate successors of each state) in order to reduce the complexity of the posterior distribution for the Bayesian DP approach.

6.2 Results

The experimental results show accumulated totals of reward received over learning phases which consist of 1000 steps for Chain and Loop, and 20000 steps for Maze. Averages were taken over 256 runs for Chain and Loop, and 16 runs for Maze. *Table 1* summarizes comparative performance after 1, 2, and 8 phases of learning. (Note that these results are pessimistic in that they show the rewards actually received during learning rather than the rewards which could be received with the instantaneous greedy policy.) In the Bayesian DP method, a new hypothesis (for the MDP) was drawn each time the system entered the starting state. In Maze, a new hypothesis was also obtained every 24 steps because there is no guarantee that the agent will return to the start in finite time.

An optimal deterministic policy would yield average rewards of 3677 in Chain and 400.0 in Loop. The optimal policy for Maze is not obvious due to the effect of slipping. Without slipping, the optimal policy would yield 2143. I estimate that the true optimal policy with slipping would yield between 1860 and 1900.

The results show that the dynamic programming approaches are significantly better than the primitive learning approaches for these problems, except for Loop where Q-learning also eventually achieves near-optimal performance. The Bayesian approach is significantly better than the Heuristic DP after 8 phases of Loop and Maze, and performs similarly for Chain. Heuristic DP is significantly better than Bayesian DP in phases 1 and 2 of Maze, but this is at a cost of worse performance in later

Table 1. Comparison of accumulated rewards. Results for phases 1 and 2 which are best by a significant margin shown in bold-type. Results for phase 8 which are near the optimal performance for the problem are also shown in bold-type.

CHAIN	PHASE 1	PHASE 2	PHASE 8
QL SEMI-UNIFORM	1594 ± 2	1597 ± 2	1602 ± 2
QL BOLTZMANN[3]	1606 ± 26	1623 ± 22	N/A
IEQL +[3]	2344 ± 78	2557 ± 90	N/A
BAYES VPI+MIX[3]	1697 ± 112	2417 ± 217	N/A
HEURISTIC DP	2855 ± 29	3450 ± 21	**3635 ± 18**
BAYESIAN DP	**3158 ± 31**	**3611 ± 27**	**3643 ± 24**
LOOP	PHASE 1	PHASE 2	PHASE 8
QL SEMI-UNIFORM	337 ± 2	392 ± 1	**399.4 ± 0.1**
QL BOLTZMANN[3]	186 ± 1	200 ± 1	N/A
IEQL+[3]	264 ± 1	293 ± 1	N/A
BAYES VPI+MIX[3]	326 ± 31	340 ± 31	N/A
HEURISTIC DP	314 ± 3	376 ± 2	394 ± 1
BAYESIAN DP	**377 ± 1**	**397.5 ± 0.1**	**399.5 ± 0.1**
MAZE	PHASE 1	PHASE 2	PHASE 8
QL SEMI-UNIFORM	655 ± 24	1135 ± 14	1147 ± 12
QL BOLTZMANN[3]	195 ± 20	1024 ± 29	N/A
IEQL+[3]	269 ± 1	253 ± 3	N/A
BAYES VPI+MIX[3]	818 ± 29	1100 ± 38	N/A
HEURISTIC DP	**1508 ± 17**	**1800 ± 4**	**1856 ± 3**
BAYESIAN DP	750 ± 6	1763± 7	**1864 ± 3**

phases. (For all three problems Bayesian DP converges to a near-optimal strategy by phase 8.)

7. Discussion and Conclusions

I have presented a powerful approach to the reinforcement learning problem. Rather than using a maximum likelihood estimate for the underlying process, the posterior distribution over process parameters is fully represented, and greedy behavior, with respect to a sample from this posterior, is used. My approach is novel in that each hypothesis is retained over a period of time, ensuring goal-directed exploratory behavior without the need to use approximate measures such as myopic VPI (Dearden et al., 1998), or heuristic exploration bonuses combined with backpropagation of uncertainty (Meuleau & Bourgine, 1999). The number of steps for which each hypothesis is retained limits the length of consistent exploration sequences, and need not be constant. The result is an automatic way of obtaining behavior which moves gradually from exploration to exploitation, without making heuristic design decisions. Accurate convergence is guaranteed because all uncertainty is represented explicitly, unlike primitive learning techniques where there is a difficult compromise between computational cost, likelihood of convergence, and accuracy of convergence.

A generalization of the algorithm is to obtain a sample of size greater than 1 from the posterior distribution. Each hypothesis in this sample could potentially yield a different greedy action and different discounted return. There are several ways to combine this information in order to choose the best action to take:

1. Choose the action with highest average predicted discounted return. This gives a more exploitative policy because it approximates an integral over the posterior. This could improve learning rates because it favors simpler models, at the cost of taking longer to find more complex, but less likely ones.
2. Choose the action with the most votes. This is similar to (1) in that it leads to more exploitative policies.
3. Choose the action corresponding to the maximum of the predictions of discounted return. This is analogous to an "optimistic" approach and may lead to more exploratory behavior.

However, using multiple hypotheses in this way may have an adverse effect on the goal-directed exploratory behavior which results from retaining hypotheses for a period of time. Exploration of these issues is an important topic for future research.

The computational complexity of Bayesian DP, as described, is relatively high compared with primitive learning techniques. This is a result of needing to generate hypotheses and perform prioritized sweeping at the start of each trial. However, for complex problems, the main computational cost is usually in the simulation rather than the learning, and so the gain through learning in fewer trials is much more important. For larger state spaces, it is possible to convert the learning process I have described into a purely local process. This is achieved by recursively traversing the MDP from the current state and resampling at any state-action pairs which have not been resampled in the last N steps, where N is indicative of the depth of (exploratory) planning likely to be required. Similarly, the greedy policy can be obtained locally by recursive traversal of the MDP. This observation leads to a natural interpretation of Bayesian DP as "hypothetical planning", connecting the rigorous Bayesian approach with a more intuitive psychological viewpoint.

Complex problems have large, continuous-valued state-spaces, and so require approximation architectures to represent state transitions and rewards. The same Bayesian approach should also be applicable to such architectures, but several new problems are introduced by

[3] These results are from Dearden, Friedman and Russell (1998).

the continuous state-space. Representations must be chosen for the continuous, stochastic state transition function and the immediate reward function. It is also necessary to calculate the discounted return from hypotheses for these functions. If these problems can be overcome, then the approach described in this paper would be much more broadly applicable.

Acknowledgements

I thank the anonymous reviewers for their detailed comments and references, which have led to significant clarification of the work in this paper.

References

Andre, D., Friedman, N., & Parr R. (1998). Generalized prioritized sweeping. *Advances in Neural Information Processing Systems 10*. Cambridge, MA: MIT Press.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, Special Volume: Computational Research on Interaction and Agency, 77*, 81-138.

Bellman, R. E. (1957a). A Markov decision process. *Journal of Mathematical Mechanics, 6*, 679-684.

Bellman, R. E. (1957b). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control.* Belmont, MA: Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming.* Belmont, MA: Athena Scientific.

Dearden, R., Friedman, N., & Andre D. (1999). Model based Bayesian exploration. *Proceedings of Fifteenth Conference on Uncertainty in Artificial Intelligence*. San Francisco, Morgan Kaufmann.

Dearden, R., Friedman, N., & Russell S. (1998). Bayesian Q-learning. *Proceedings of Fifteenth National Conference on Artificial Intelligence.* Menlo Park, CA: AAAI Press.

Fiechter, C-N. (1997). Expected mistake bound model for on-line reinforcement learning. *Proceedings of Fourteenth International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Friedman, N., & Singer, Y. (1999) Efficient Bayesian parameter estimation in large discrete domains. *Advances in Neural Information Processing Systems 11*. Cambridge, MA: MIT Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research, 4*, 237-285.

Martin, J., J. (1967). *Bayesian decision problems and Markov chains.* New York: John Wiley.

Meuleau, N., & Bourgine, P. (1999). Exploration of multi-state environments: local measures and back-propagation of uncertainty. *Machine Learning, 35*, 117-154.

Mitchell, T. M. (1997). *Machine learning.* Columbus, OH: McGraw-Hill.

Moore, A. W., & Atkeson, C., G. (1993). Prioritized sweeping: reinforcement learning with less data and less time. *Machine Learning, 13*, 103-130.

Kearns, M., & Singh, S. (1998). Near-optimal performance for reinforcement learning in polynomial time. *Proceedings of Fifteenth International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Strens, M. J. A. (1999). *Learning, cooperation and feedback in pattern recognition.* Ph.D. Thesis, Physics Department, King's College London, London.

Sutton, R. S., & Barto, S. (1998). *Reinforcement learning.* Cambridge, MA: MIT Press.

Thrun, S., B. (1992). *Efficient exploration in reinforcement learning* (Technical Report CMU-CS-92-102), School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph.D. Thesis, Psychology Department, Cambridge University, Cambridge, U.K.