Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

A SAMPLED FICTITIOUS PLAY BASED LEARNING ALGORITHM FOR INFINITE HORIZON MARKOV DECISION PROCESSES

Esra Sisikoglu

The University of Missouri
E3437 Lafferre Hall
Columbia, MO 65211, USA

Marina A. Epelman

The University of Michigan
1205 Beal Ave.
Ann Arbor, MI 48109, USA

Robert L. Smith

The University of Michigan
1205 Beal Ave.
Ann Arbor, MI 48109, USA

ABSTRACT

Using Sampled Fictitious Play (SFP) concepts, we develop SFPL: Sampled Fictitious Play Learning — a learning algorithm for solving discounted homogeneous Markov Decision Problems where the transition probabilities are unknown and need to be learned via simulation or direct observation of the system in real time. Thus, SFPL simultaneously updates the estimates of the unknown transition probabilities and the estimates of optimal value and optimal action in the observed state. In the spirit of SFP, the action after each transition is selected by sampling from the empirical distribution of previous optimal action estimates for the current state. The resulting algorithm is provably convergent. We compare its performance with other learning methods, including SARSA and Q-learning.

1 INTRODUCTION

In this paper, we propose a learning algorithm based on Fictitious Play concepts for finding optimal policies in homogeneous discrete Markov Decision Problems (MDPs) with unknown transition probabilities. MDPs are characterized as sequential decision making problems that involve choosing actions, making observations of the system, and then modifying these actions based on the new information (Powell 2007, Puterman 1994). The nature of these problems makes it possible to propose solution approaches even when the dynamics of the system (*e.g.*, cost of decisions, probability distributions of random disturbances in the system, etc.) are unknown. By designing a learning algorithm that simultaneously makes observations about the aspects of the system unknown to the decision maker and chooses actions using information obtained via these observations, the optimal solution of the problem can be approximated. Ideally such an algorithm uses simple calculations to generate approximate solutions in each iteration. For this reason we believe that Fictitious Play (FP) concepts can be very useful in this framework. FP is an iterative algorithm used to imitate analysis by players engaged in learning each others' behaviors in a game to find a Nash Equilibrium solution (Brown 1951, Robinson 1951). Our approach is to model an MDP problem as a game (by associating players, strategies, and payoffs with components of the MDP) such that the optimal solution of the problem is a Nash Equilibrium of this game, and apply a variant of FP to the game to search for the optimal solution of the MDP, resulting in a new learning algorithm.

Learning algorithms for MDPs have been extensively studied in the literature. Actor-critic methods, Reinforcement Learning, Neuro-Dynamic Programming (Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998, Szepesvári 2010), Approximate Dynamic Programming (Powell 2007) are well known frameworks for such algorithms. For example, Temporal-Difference learning methods, including Q-learning and SARSA and their extensions, have been very popular for problems that are sufficiently small to allow lookup-table representation of Q-values, *i.e.*, estimates of expected values of state-action pairs (Sutton and Barto 1998). These algorithms can be applied to a variety of problems, and are suitable for interactive learning, where learning occurs while interacting with the real system rather than a simulation (cf. Szepesvári 2010). The algorithms estimate the Q-values and update these estimates as more information becomes available. Multiple versions of learning algorithms, including extensions that allow for approximate solutions of large-scale MDPs, are discussed in the survey publications cited above, and references therein.

In designing (interactive) learning algorithms one needs to specify a learning policy, *i.e.*, the rules for selecting actions in order to explore the state/action space, as well as a value function approximation update method. The most straightforward approach to learning is to uniformly sample actions in each state. Biasing the sampling procedure to choose "better" actions, *e.g.*, using a greedy approach that chooses the best action based on the current value estimates, can significantly improve performance (Chang et al. 2007). However, a purely greedy method has too much bias and might force the algorithm to get "stuck" at a local optimal solution. One of the most common approaches to dealing with this problem is to use ε-greedy sampling (in which at each iteration, with probability ε, a randomly sampled action is used instead of the greedy choice) or Boltzmann sampling, or another randomized method that has a bias towards greedy selection of actions. As an alternative, Chang et al. (2005) designed an adaptive sampling algorithm for finite horizon stochastic dynamic programming problems in which they incorporate regret analysis developed for the multi-armed bandit problems into the action sampling process. With this adaptive approach, they save the effort of sampling "poor actions" (which would significantly improve the algorithm performance if the sampling is relatively time consuming) without sacrificing the convergence properties of the algorithm. Adaptive sampling (and its potential advantages) is inherent in the algorithm we propose in this paper through the utilization of "best action" histories, which are part of the FP framework.

Another important aspect of a learning algorithm's design is the relationship between the value update method and the learning policy, *i.e.*, the states visited and actions deployed. For example, Q-learning is an *off-policy* learning algorithm, *i.e.*, the update of the value of a state-action pair is independent of the learning policy followed, whereas SARSA is an *on-policy* learning algorithm, in which the policy that is being followed is directly used in the value updates (Sutton and Barto 1998). Singh et al. (2000) establish convergence results for single-step on-policy algorithms, such as SARSA(0). They also argue that analysis of on-policy algorithms is more difficult compared to their off-policy counterparts, since in the later the learning policy and value updates can be separated in the analysis. For instance, Kearns and Singh (1999) study the convergence and convergence rate of Q-learning (as well as an indirect learning algorithm), and Kearns and Singh (2002) establish polynomial bounds on the resources required to achieve near-optimal return by a variation that simulates the system until enough information is collected on a set of states and then uses optimization routines to update the value of these "known" states.

In this paper we design a learning algorithm based on Sampled Fictitious Play (SFP) that not only keeps estimates of the optimal value and the optimal action for each state but also estimates system parameters (the transition probability distribution). *We assume for simplicity that the randomness in the system depends on the state but is independent of the action taken.* This assumption holds for many practical problems. For example, in inventory control problems (Puterman 1994) the demand is typically independent of the production amount. In dynamic location problems (Rosenthal et al. 1978), the action (positioning of the equipment trailer) does not influence the random disturbance (the location of the next work request). Even if this assumption is not satisfied (*i.e.*, the random disturbance depends on not only the state but also the action), the algorithmic approach we propose in this paper can still be used with slight modifications and the convergence analysis can be adapted for the modified approach.

In designing our algorithm we use (Sampled) Fictitious Play concepts. FP is an iterative algorithm used to imitate analysis by players engaged in learning each others' behaviors in a game (Brown 1951, Robinson 1951) and it has been shown to converge to a Nash Equilibrium for identical interest games (Monderer and Shapley 1996). At every iteration of FP, each player (or decision maker) calculates ("plays") the *best reply*, *i.e.*, the action that optimizes his expected utility, assuming that the other players select their actions independently at random according to the empirical distributions of their own best replies from the previous iterations. This approach is computationally time consuming since finding, for each player, a best reply that optimizes his expected utility can potentially require calculation of utility values of all combinations of actions of all players. Lambert III et al. (2005) proposed a Sampled Fictitious Play (SFP) algorithm, in which best replies to a *sample* of actions drawn from the history of best replies are calculated, and showed that it converges to a Nash Equilibrium of identical interest games if the sample size is increased at an appropriate rate. Computational requirements of each iteration of the SFP algorithm are dramatically less than that of FP, especially while the sample sizes are sufficiently small.

To facilitate application of these concepts in our algorithm, we define a game on a homogeneous infinite horizon MDP by treating the states and feasible actions in each state as the players and actions available to each player, respectively. In this framework, a collection of actions played by all players defines a stationary policy for the MDP problem. The identical payoff each player receives is then naturally defined to be the expected total discounted reward (or cost) of this policy. (A related approach has been used in applying SFP to finite horizon model-free stochastic DPs in Epelman et al. (2011).) A direct application of SFP in this setting (using a sample of size 1 for illustration) would be to have each player sample an action from his history of best replies and then to calculate, for each player, a payoff-optimizing action given the policy induced by actions sampled by the other players. However, since we consider problems for which the system dynamics are not readily available, estimating the relevant costs associated with each policy would require extensive system observations or simulations in each iteration of SFP. Instead, in the spirit of interactive learning, in the proposed algorithm we maintain estimates of the optimal values of the states, along with estimates of relevant probability distributions, and use these estimates in best reply calculations at each iteration.

Specifically, throughout the algorithm we maintain estimates of the probability that a given outcome of state-dependent random disturbance occurs, and an estimate of the optimal value, as well as a *history* of past best reply actions, for each state. At each iteration, we implement an action sampled from the best reply history for the current state, and then observe the random disturbance. Based on this observation, we update the estimate of the random disturbance probability, and then, using these updated estimates, as well as the current value estimates, we calculate the new best reply action for the current state and add it to the history. Finally, the value estimate of the current state is updated using the best reply action. The system then transitions to the next state according to the implemented action and observed realization of the disturbance. The resulting method is an off-policy algorithm, since the implemented actions are different from the ones used in value estimate updates. On the other hand, the empirical distribution of best replies can be viewed as a probabilistic estimate of the optimal action in that state, and adding the new best reply action to the history serves as an update of this estimate, increasing the probability that this action is sampled next time the system re-enters the state. This connection between the learning policy and the value update method provides an unique balance between exploration and exploitation within the reinforcement learning framework.

An additional distinction of the proposed algorithm is that it does not require the user to select and "tune" any input parameters (*e.g.*, step sizes in value updates or parameters of sampling distributions). This makes the algorithm easier to implement, and possibly more robust in its performance.

This paper is organized as follows: in Section 2, notation and assumptions are given. In Section 3, the algorithm is described. Section 4 discusses convergence. The algorithm is tested on dynamic location and windy gridworld problems in Section 5, with conclusions following in Section 6.

2 NOTATION AND ASSUMPTIONS

We consider an infinite horizon, discrete time, homogeneous Markov Decision Process (MDP) defined by the collection of objects $(S; A_s, p(\cdot|s,a), c(s,a),\ \forall s \in S)$, where (see, *e.g.*, Puterman 1994)

- S is the set of states ($|S| < \infty$, where $|S|$ is the cardinality of set S);
- A_s is the set of feasible actions in state $s \in S$ ($|A_s| < \infty$);
- $p(s'|s,a) = \text{Prob}(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability the system transitions into state s' at time $t+1$ given the system is in state s and the action $a \in A_s$ is chosen at time t;
- $c(s,a) \in \mathbb{R}$ is the immediate cost of choosing action a in state s.

Once an action a is chosen in state s, an exogenous random disturbance is realized and the system transitions into state s'. We assume that the random disturbance depends on the current state s but is independent of the action a chosen. As mentioned earlier, inventory control problem (Puterman 1994) and dynamic location problems (Rosenthal et al. 1978) are examples of problems that satisfy this assumption. In both of these examples, however, an action needs to be taken *before* the outcome of the random disturbance is observed, which is the structure we assume throughout the paper. We use the following notation to represent the random disturbance and the transitions in the system:

- $\omega(s)$ denotes the total random disturbance, which is assumed to be independent of action taken in state s. Sample space $\Omega(s)$ of outcomes of $\omega(s)$ is assumed to be finite for every state s.
- $p(\omega(s) = \hat{\omega}(s)|s)$ denotes the probability that realization $\hat{\omega}(s) \in \Omega(s)$ of random disturbance $\omega(s)$ occurs when the system is in state s. We will use the shorthand $p(\hat{\omega}(s))$ to represent this probability.
- $f(s,a,\hat{\omega}(s))$ is the system evolution function. That is, given a realization $\hat{\omega}(s)$ of $\omega(s)$ and action $a \in A_s$, the system will transition from state s to the (deterministic) state $f(s,a,\hat{\omega}(s))$.

Using the notation defined above, the transition probabilities in the MDP can be written as follows:

$$p(s'|s,a) = \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s)) \mathbf{1}_{\{f(s,a,\hat{\omega}(s))=s'\}},$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function.

In this paper, we focus on finding a stationary policy that minimizes discounted expected cost in the above MDP, with $\gamma \in (0,1)$ denoting the discount factor. (Contrast this with *online learning*, where the total discounted cost incurred during learning is considered, Szepesvári 2010.) It is well-known (see, *e.g.*, Puterman 1994, Bertsekas and Tsitsiklis 1996) that the optimal *state values* (or costs to go) $J(s),\ s \in S$, satisfy Bellman's equation

$$J^\star(s) = \min_{a \in A(s)} \left[c(s,a) + \gamma \sum_{\hat{\omega}(s) \in \Omega(s)} p(\hat{\omega}(s))\ J^\star(f(s,a,\hat{\omega}(s))) \right],$$

and the actions attaining the minimum form an optimal deterministic stationary policy.

We assume that feasible states, actions in each state of the problem and costs of these actions are known, but little or no information is available about the probability distribution of the random disturbance. That is, the distributions of random variables $\omega(s),\ s \in S$, are not known, but a realization of $\omega(s)$ can be observed every time the system visits state s. Thus, this problem fits into the reinforcement learning framework as described by Sutton and Barto (1998).

3 SFP BASED LEARNING ALGORITHM (SFPL)

A learning approach for solving the problem in this setting is to observe and learn about the system dynamics under an implemented policy, while updating this policy as more information about the system becomes available. To make the approach suitable for learning while interacting with the real system, updates are coordinated with observations of transitions; therefore, the iteration count of the algorithm coincides with the real time during which the system evolves. We use subscript t to refer both to the iteration of the algorithm and the real time.

The following notation is used to describe the SFP-Based Learning Algorithm (SFPL):

- *MDP* denotes the infinite horizon homogeneous Markov Decision Process under consideration.
- $J_t(\cdot) : S \to \mathbb{R}$ is the *estimate* of the optimal value function for *MDP* returned by the algorithm at iteration/time t. (With a slight abuse of notation, we also refer to $J_t \in \mathbb{R}^{|S|}$ as a value vector.)
- $p_t(\cdot|s,a)$ and $p_t(\hat{\omega}(s))$ denote *estimates* of, respectively, $p(\cdot|s,a)$ and $p(\hat{\omega}(s))$ at iteration/time t.
- MDP_t is the Markov Decision Process defined on the same state and action space as *MDP* with the difference that the transition probabilities are given by $p_t(\cdot|s,a),\ \forall s \in S,\ a \in A_s$. (Note that $J_t(s),\ s \in S$ *do not* denote the optimal state values in MDP_t.)
- $I_t(s)$ is the number of times the process has visited state s up to (and including) iteration/time t.
- $H(s)$ is the history of "best reply" actions for state s (see algorithm description for more details).
- $W(s)$ is the vector of realizations of $\omega(s)$ observed up to (and including) the current iteration.

By definition, the size of vectors $W(s)$ and $H(s)$ at iteration t is equal to $I_t(s)$ for all s except possibly the current state of the system (for which the associated objects are being updated during the current iteration). Moreover, estimates $p_t(\cdot|s,a)$ and $p_t(\hat{\omega}(s))$ are related by

$$p_t(\hat{\omega}(s)) = \frac{\text{\# of times } \hat{\omega}(s) \text{ is observed in state } s \text{ up to and including iteration } t}{I_t(s)} = \frac{\sum_{i=1}^{I_t(s)} \mathbf{1}_{\{W_i(s)=\hat{\omega}(s)\}}}{I_t(s)}. \quad (1)$$

SFPL is initialized at a random state (unless the starting state is already identified) and all histories are set to be empty. In each iteration $t \geq 1$, an action is selected by sampling (uniformly) an element from the history vector for the current state s_t (if the history is empty, a feasible action is sampled) and the random disturbance in the system is observed. Based on this observation, the estimates of the random disturbance probabilities $p_t(\hat{\omega}(s_t))$ are updated and a best reply $a^\star$ for state s_t is calculated based on estimates $p_t(\hat{\omega}(s_t))$ and $J_{t-1}(\cdot)$. The best reply action is then added to the history of s_t, and a new value estimate $J_t(s_t)$ is calculated utilizing $a^\star$. Finally, the system transitions into the next state given by $f(s_t, a^\star, \hat{\omega}(s_t))$. Notice that this is an *off-policy* method: the action sampled from history is implemented and thus affects the transition, whereas the action used in updating the estimate of the value is $a^\star$.

SFPL is designed for problems which satisfy the assumption that the probability distribution of the random disturbance is independent of the action. However, the algorithm can be modified if this assumption is not satisfied, albeit possibly at the cost of additional storage requirements. For instance, post-decision states (Powell 2007) may be used, or the count of random disturbance realizations will need to be recorded for each state-action pair. Convergence analysis will need to be modified accordingly. Although these modifications may increase the memory requirements of the algorithm, the computational requirements of the algorithm would be similar.

3.1 Outline of Algorithm SFPL

Below, notation $X \leftarrow [X, x]$ represents the concatenation operation.

- Initialization:
  - Assign initial values to $J_0(s)$ for all $s \in S$. Set $H(s) = [\,]$, $I_0(s) = 0$, $W(s) = [\,]$ for all $s \in S$.
  - Set $t = 1$ and observe $s_1 \in S$. (This is the state where the process starts.)
- Iteration/time $t \geq 1$ (below, $s_t \in S$ represents the state at the beginning of iteration):
  State:

- $I_t(s_t) = \begin{cases} I_{t-1}(s_t)+1 & \text{if } s = s_t, \\ I_{t-1}(s_t) & \text{if } s \neq s_t. \end{cases}$

Action:

- Select an action a_t uniformly at random among elements of $H(s_t)$ (if $H(s_t)$ is empty, select an action a_t from A_{s_t} uniformly at random).
- Observe the realization $\hat{\omega}(s_t)$ of the random disturbance $\omega(s_t)$; update $W(s_t) \leftarrow [W(s_t),\ \hat{\omega}(s_t)]$.

Computation:

- Calculate the "best reply":

$$a^\star = \operatorname*{argmin}_{a \in A_{s_t}} \left\{ c(s_t, a) + \gamma \frac{1}{I_t(s_t)} \sum_{i=1}^{I_t(s_t)} J_{t-1}(f(s_t, a, W_i(s_t))) \right\}, \tag{2}$$

  where $W_i(s_t)$ is the ith component of vector $W(s_t)$.
- Update the history $H(s_t) \leftarrow [H(s_t),\ a^\star]$.
- Set

$$J_t(s) = \begin{cases} c(s,a^\star) + \gamma \sum\limits_{\hat{\omega}(s) \in \Omega(s)} p_t(\hat{\omega}(s))\ J_{t-1}(f(s,a^\star,\hat{\omega}(s))) & \text{if } s = s_t, \\ J_{t-1}(s) & \text{otherwise,} \end{cases}$$

  where $p_t(\hat{\omega}(s))$ is given by (1).

Transition:

- Set $s_{t+1} = f(s_t, a_t, \hat{\omega}(s_t))$.
- Update $t \leftarrow t+1$.

4 CONVERGENCE OF SFPL

The following convergence result is proven in Sisikoglu (2009):

Theorem 1 Assuming that, regardless of the policy employed, every state is visited infinitely often due to the inherent randomness in the system, $\lim\limits_{t\to\infty} J_t(s) = J^\star(s)$, with probability $1, \forall s \in S$.

The assumption is satisfied, *e.g.*, if the markov chain induced under any stationary deterministic policy in the MDP is irreducible. The proof of Theorem 1 uses the infinitely many visits to each state to show that the estimates of probability distributions of the random disturbance in each state become asymptotically accurate, leading in turn to asymptotically accurate estimates of each state's optimal value.

There are various problem classes that satisfy the assumption of Theorem 1. For example, for the version of the windy gridworld problem (Section 5) described in Şimşek and Barto (2006), any state neighboring the current one can be visited in the next iteration with a constant probability. In a capacitated inventory control problem, if negative demand represent returns and can be at any feasible level with positive probability, this assumption would also be valid. A relaxed version of the assumption, where only states that are recurrent under an optimal *MDP* policy are visited infinitely often is sufficient to prove that value estimates for those relevant states converge to optimality.

Intuition suggests that SFPL is likely to find the optimal actions in the states recurrent under the optimal policy by setting $J_0(s) < J^\star(s)$, $\forall s \in S$, and/or populating histories of each state by all feasible actions. Since the action implemented at each iteration is sampled from history, each player/state will try to explore and choose diverse actions initially. The numerical results on an instance of gridworld problem given in Section 5 supports this intuition: SFPL visits the states in the recurrent class of the markov chain induced by the optimal policy more often than other states, and quickly discovers the optimal actions and optimal values of these states.

5 NUMERICAL RESULTS

We use the dynamic location problem adapted from Rosenthal et al. (1978) and the windy gridworld problem described in Sutton and Barto (1998) to test performance of the SFPL algorithm, and compare it to simple versions of Q-learning and SARSA algorithms. All computations were performed in MATLAB on a laptop with 2 GB RAM and 2.0 Ghz CPU (Intel Core II Duo Processor).

5.1 Dynamic Location Problem

The dynamic location problem analyzed in this section is adapted from Rosenthal et al. (1978); it is also referenced in Puterman (1994). A workforce that serves Q facilities moves between these facilities according to a stationary probability distribution given by probability transition matrix P, where P_{ij} is the probability that the workforce moves from facility i to facility j. An equipment trailer can be located at any of the facilities in Q. The cost of moving the trailer from i to j is denoted by R_{ij}. If the workforce is located at facility i and the trailer is at facility j, then U_{ij} denotes the cost of obtaining equipment from the trailer. The problem is modeled as an infinite horizon MDP by defining the states to be the tuple (w,r), where w and r are the location of the workforce and the trailer, respectively.

We used an instance with 4 facilities, and thus 16 states, and the following costs and probabilities:

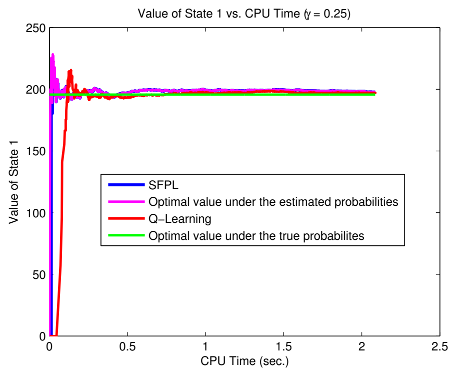
$$R = \begin{pmatrix} 0 & 100 & 200 & 300 \\ 100 & 0 & 100 & 200 \\ 200 & 100 & 0 & 100 \\ 300 & 200 & 100 & 0 \end{pmatrix}, \ U = \begin{pmatrix} 0 & 999 & 999 & 999 \\ 200 & 0 & 100 & 100 \\ 200 & 100 & 0 & 100 \\ 200 & 100 & 100 & 0 \end{pmatrix}, \ P = \begin{pmatrix} 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.6 & 0.4 \\ 0.1 & 0.4 & 0.1 & 0.4 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}.$$

It is important to note that the probability distribution of the workforce location (*i.e.*, the random disturbance) given above is only used to simulate the system and to calculate the optimal solution of the problem. The algorithms we tested do not have access to P; instead they estimate it by observing the system.

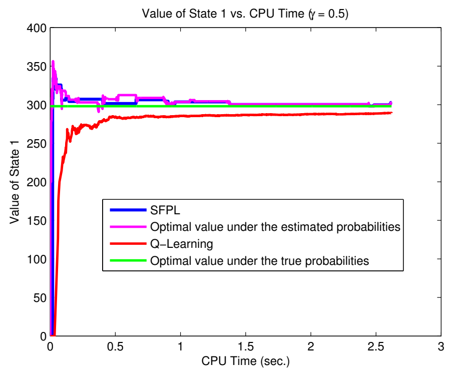
We compared SFPL to a simple version of Q-learning with the ε-greedy learning policy. Implementation of Q-learning requires a choice of sampling parameter ε and a step size α used in value update. Convergence properties the algorithm strongly depend on the choice of these parameters. George and Powell (2006) list various step size parameters and discuss the effect of various choices. The results presented below were obtained for the following "default" choices of parameters: $\varepsilon = 0.1$ and $\alpha = 1/I(s,a)$, where $I(s,a)$ is the number of times the value of state-action pair (s,a) has been updated. While we acknowledge that parameter tuning can improve performance of Q-learning, such tuning was beyond the scope of our experiments. Since SFPL requires no parameter tuning, this illustrates its ease of implementation.

To compare the performance of SFPL and Q-learning on the dynamic location problem, the two algorithms were run for the same amount of CPU time. Figure 1 compares the values computed by SFPL and Q-learning for state $w = 1, r = 1$ (which we label simply as State 1) as a function of CPU time elapsed for problem instances with various discount rates ($\gamma = 0.25, 0.50, 0.75, 0.9$). The optimal value of State 1 in *MDP* (the system with the true probability distribution $p(\omega(s))$) and in MDP_t (the estimated system at time t with the probability distributions $p_t(\omega(s))$) are also plotted for comparison. (Optimal solutions of *MDP* and MDP_t for all t were computed off-line.) This comparison provides valuable insights into the performance of SFPL. For example, as shown in Figure 1, values returned by SFPL are very close to the optimal solution of the estimated system MDP_t after a short time. This implies that SFPL quickly starts choosing the true optimal actions based on the current knowledge of the system. As can be seen from the figures, Q-learning and SFPL are comparable for lower discount rates; however, performance of Q-learning deteriorates for higher discount rates, while performance of SFPL is unaffected.

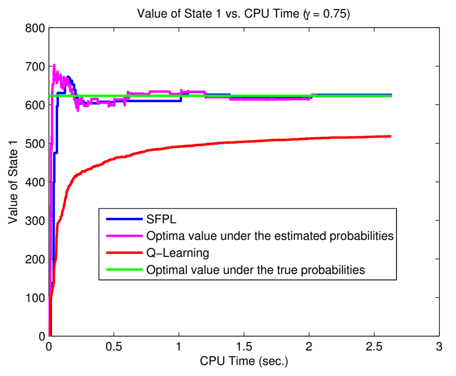
Figure 2 shows absolute value differences between the values returned by SFPL and Q-learning at termination (given the same amount of CPU time) and the optimal values $J^\star(s)$ of all states for $\gamma = 0.5$ and $\gamma = 0.75$. As before, performance of the two algorithms is comparable for lower discount rates; however, SFPL performs significantly better for higher discount rates.



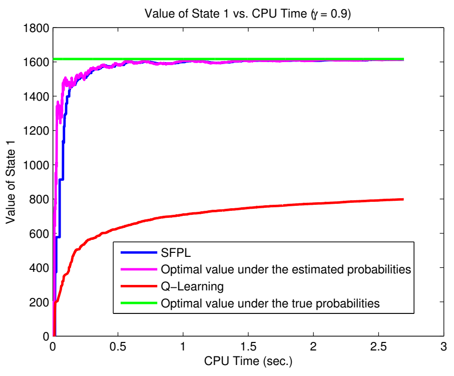
(a) $\gamma = 0.25$



(b) $\gamma = 0.50$

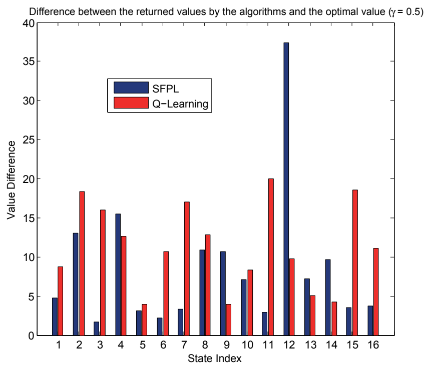


(c) $\gamma = 0.75$

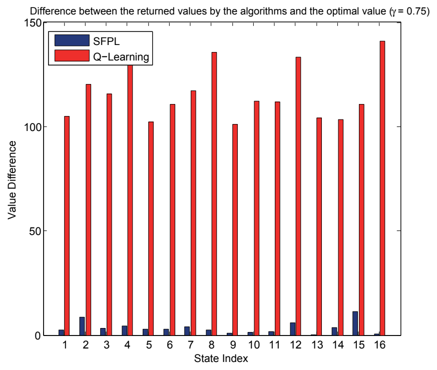


(d) $\gamma = 0.9$

Figure 1: Progress of value estimates for State 1 of the dynamic location problem in SFPL and Q-learning, and its optimal values in MDP and MDP_t.



(a) $\gamma = 0.5$



(b) $\gamma = 0.75$

Figure 2: Differences between the the optimal values of each state of the dynamic location problem and the values returned by SFPL and Q-learning.

5.2 Windy Gridworld

The windy gridworld problem involves traveling in a grid each column of which is subject to winds of various strengths. We use the grid given in Sutton and Barto (1998), also shown in Figure 3a. The number below each column represents the mean strength of the wind blowing towards North (upward) affecting only that column. The strength of the wind is random and possibly changes every time a state in that column is visited. We assume that in each column the wind strength can be one unit above, one unit below, or at its mean value equally likely at any time; *i.e.*, if the mean strength of the wind is 0 , then with 1/3 probability the wind blows South with strength 1, with 1/3 probability the wind blows North strength 1, and with 1/3 probability there is no wind in effect. (Once again, the probability distribution of this random disturbance is only used to simulate the system; it is not known to the algorithms compared in this section.) In each state the feasible actions are to move in one of the eight possible directions, namely E, SE, S, SW, etc. Once an action is chosen, the next state is determined by the current state (*i.e.*, the current location), the action in the current state (*i.e.*, the movement direction) and the realization of the wind strength at this time. If the resulting state is outside the boundaries of the grid, then it is mapped back to the closest state within the boundaries. The goal is to minimize the expected travel time from point S to point F in the grid. Thus, the cost of each decision is taken to be 1, and the objective is to minimize total expected cost.

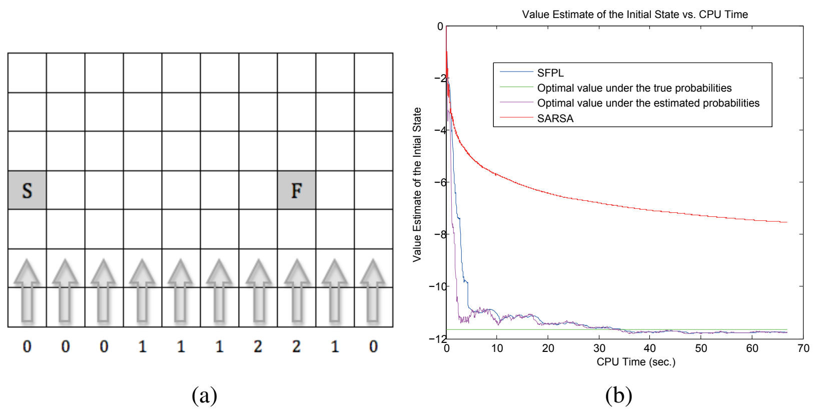
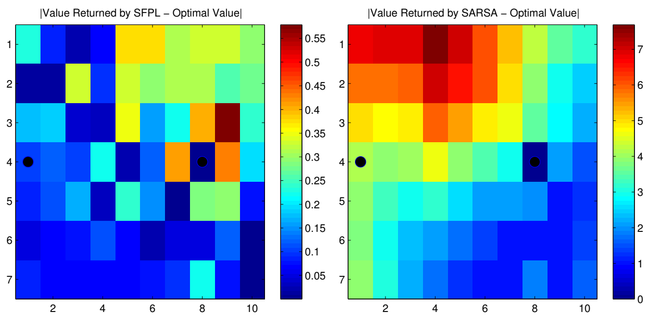


Figure 3: (a) Windy Gridworld Problem; (b) The value estimates of the initial state S returned by SFPL, SARSA and by running value iteration on the intermediate estimated system MDP_t. The optimal value under the true probabilities is also included for comparison.

To apply SFPL to this problem with an absorbing state, the system was restarted at the initial state (point S) once the destination (point F) was reached. One run, from S to F, is called an *episode* (see, *e.g.*, (Szepesvári 2010)). Note that this problem does not satisfy the assumptions of Theorem 1: some states may not be visited infinitely often under some policies. Moreover, $\gamma = 1$. As we will see, the numerical results obtained support the intuitive argument in Section 4 that the values of the states in the recurrent class of the induced markov chain under the optimal policy converge to the optimal values.

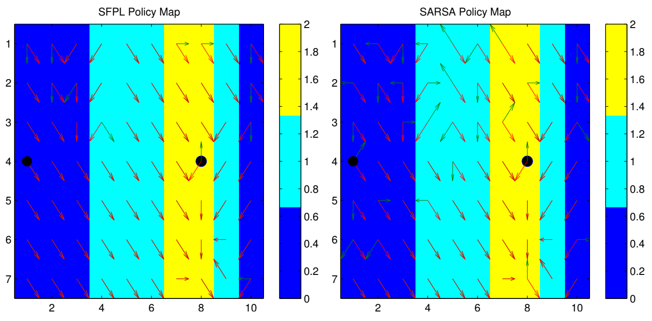
We compared SFPL with the SARSA algorithm (Sutton and Barto 1998). We used the implementation code for SARSA(0) developed by John Weatherwax available online at (Weatherwax 2005). For the SARSA algorithm we set the ε-greedy sampling parameter $\varepsilon = 0.1$ and the step size $\alpha = \frac{1}{I(s,a)}$, where $I(s,a)$ is the number of times the value of state-action pair (s,a) has been updated.

In SFPL, the values of all states are initialized at value 0. We ran SFPL and SARSA for one minute of CPU time. During this time, SFPL completed around 2,000 episodes, whereas SARSA completed around 15,000 episodes. At the end of each episode of SFPL, we also calculated (off line) the optimal solution of MDP_t by running the classic synchronous value iteration algorithm with the most up-to-date estimate of the probability distribution of the wind strength; these values are provided for comparison. Figure 3b shows value estimates of the initial state S returned by SFPL and SARSA at the beginning of each episode, as a function of CPU time elapsed. The optimal value of S in MDP and MDP_t are also plotted for comparison. Note that SFPL finds values closer to the optimum faster than SARSA. Figure 4 shows the absolute value difference between the final estimated values returned by SFPL and SARSA and true optimal values of the states. The blue color (indicating high approximation accuracy) is dominant on the states that are to the South of the states S and F, identified by dots, in both figures. This is because both algorithms sample the optimal action, which is to move in the SE direction, more often than other available actions, and therefore the value estimates of the states to the South of the grid get closer to the optimal values. These figures also show that error of the value estimates obtained by SFPL is significantly smaller than that of SARSA on all states (note that the color-wash scale had to be made different in the two figures; *e.g.*, a dark red square represents an error which is an order of magnitude smaller for SFPL than for SARSA). Another way to compare these algorithms is to check if they successfully identify the optimal actions. Policies returned by SFPL and SARSA are presented in Figure 5 (colors represent the nominal value of the wind in each column). The red arrows in each state represent the optimal action, and the black arrows denote the action returned by the algorithms; only the red arrow is shown if the two coincide. According to the figure, SFPL found the correct actions for more states than SARSA.



(a) (b)

Figure 4: Absolute value difference of the final estimated state values returned by (a) SFPL and (b) SARSA and the optimal values.



(a) (b)

Figure 5: Policy returned by SFPL (a) and SARSA (b) (black arcs) vs. the optimal policy (red arcs).

In each iteration of SFPL, the best action in the current state is obtained by calculating and comparing the expected value of each feasible action, cf. equation (2). Thus the computational workload per iteration of SFPL is higher than for SARSA and Q-learning. Our presentation of algorithm progress as a function of

CPU time allows for a fair comparison. Our results suggest that the magnitude of improvement obtained in each iteration of SFPL is higher.

Another important property that affects an algorithm's performance is its memory requirement. Q-learning and SARSA maintain an estimate of the value of each state-action pair. Therefore, these algorithms need to keep a data structure of size $O(|S| \times \max_s |A_s|)$. In SFPL, the history of best replies and the history of random disturbances are recorded. However, this data is only used to construct empirical distributions. Therefore, the memory requirement of SFPL is $O(|S| \times \max_s(\max\{|A_s|, |\Omega(s)|\}))$. Depending on the problem structure (specifically, $\max_s |\Omega(s)|$), the memory requirements of SFPL may be larger than those of Q-learning and SARSA.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed SFPL — a learning algorithm designed, using SFP concepts, to solve homogeneous discrete MDPs with unknown transition probabilities. The algorithm can be seen as an off-policy reinforcement learning algorithm. However, the particular learning policy used — in which implemented actions are sampled from the history of best replies computed for each state — provides an unique balance between exploration and exploitation within the reinforcement learning framework. Moreover, the algorithm is free of any parameters that need to be selected and tuned, unlike most other reinforcement learning methods, leading to simpler implementation.

Initial experiments with SFPL suggest that using the history of best replies to sample actions helps to discover the states in the "recurrent class" of the induced markov chain under the optimal policy and therefore find better value estimates for these states. Also, for a discounted MDP, SFPL performed robustly for a range discount rates, unlike Q-learning and SARSA(0).

In the future, we intend to test SFPL on larger-scale problems, and compare it to more modern reinforcement learning algorithms. On the theoretical side, we would like to find the weakest possible forms of the assumption in Theorem 1 that make provable convergence possible. The possibility of a rigorous analysis of the algorithm's convergence rate is also an open question.

ACKNOWLEDGMENTS

The authors acknowledge support from NSF grants CMMI-0422752 and CCF-0830092.

REFERENCES

Bertsekas, D., and J. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Brown, G. 1951. "Iterative solution of games by fictitious play". *Activity analysis of production and allocation* 13:374–376.

Chang, H., M. Fu, J. Hu, and S. Marcus. 2005. "An adaptive sampling algorithm for solving Markov decision processes". *Operations Research* 53:126–139.

Chang, H., M. Fu, J. Hu, and S. Marcus. 2007. *Simulation-based Algorithms for Markov Decision Processes*. Springer Berlin.

Epelman, M. A., A. Ghate, and R. L. Smith. 2011. "Sampled fictitious play for approximate dynamic programming". *Computers & Operations Research* 28:1705–1718.

George, A., and W. Powell. 2006. "Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming". *Machine Learning* 65:167–198.

Kearns, M., and S. Singh. 1999. "Finite-sample convergence rates for Q-learning and indirect algorithms". *Advances in Neural Information Processing Systems* 11:996–1002.

Kearns, M., and S. Singh. 2002. "Near-optimal reinforcement learning in polynomial time". *Machine Learning* 49:209–232.

Lambert III, T., M. Epelman, and R. Smith. 2005. "A fictitious play approach to large-scale optimization". *Operations Research* 53:477–489.

Monderer, D., and L. Shapley. 1996. "Fictitious play property for games with identical interests". *Journal of Economic Theory* 68:258–265.

Powell, W. 2007. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Wiley-Interscience.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. New York, NY, USA.

Robinson, J. 1951. "An iterative method of solving a game". *Annals of Mathematics* 54:296–301.

Rosenthal, R., J. White, and D. Young. 1978. "Stochastic Dynamic Location Analysis". *Management Science* 24:645–653.

Şimşek, Ö., and A. G. Barto. 2006. "An intrinsic reward mechanism for efficient exploration". In *Proceedings of International Conference on Machine Learning*, edited by W. Cohen and A. Moore, 841–848. The International Machine Learning Society - ICML.

Singh, S., T. Jaakkola, M. Littman, and C. Szepesvari. 2000. "Convergence results for single-step on-policy reinforcement-learning algorithms". *Machine Learning* 39:287–308.

Sisikoglu, E. 2009. *Distributed Algorithms Based on Fictitious Play for Near Optimal Sequential Decision Making*. Ph.D. thesis, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI. Available via http://hdl.handle.net/2027.42/64602.

Sutton, R., and A. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press Cambridge, MA.

Szepesvári, C. 2010, January. *Algorithms for Reinforcement Learning*, Volume 4. Synthesis Lectures on Artificial Intelligence and Machine Learning.

Weatherwax, J. 2005. "SARSA implementation is MATLAB". Accessed June 4, 2011. http://waxworksmath.com/Authors/N_Z/Sutton/sutton.html.

AUTHOR BIOGRAPHIES

ESRA SISIKOGLU is an Assistant Professor in the Department of Industrial and Manufacturing Systems Engineering at the University of Missouri. She received her B.S., M.S. and Ph.D. degrees in Industrial and Operations Engineering from the University of Michigan. She is a member of INFORMS. Her research interests include modeling and developing algorithmic and computational methods for large-scale decision making problems. Her email address is sisikoglue@missouri.edu.

MARINA A. EPELMAN is an Associate Professor in the department of Industrial and Operations Engineering at the University of Michigan. She received a B.A. from Cornell University and a Ph.D. in Operations Research from Massachusetts Institute of Technology. She is an Associate Editor of *Operations Research* and a member of MOS, INFORMS and SIAM. She currently serves as a secretary of the INFORMS Optimization Society and SIAM Activity Group in Optimization. Her e-mail is mepelman@umich.edu.

ROBERT L. SMITH is the Altarum/ERIM Russell D. O'Neal Professor of Engineering at the University of Michigan and is a Fellow of INFORMS. He is Director of the Dynamic Systems Optimization Laboratory at the University of Michigan which engages in research on the modeling and analysis of dynamical systems over time. He is an Associate Editor of *Operations Research* and is the author of over a hundred peer reviewed publications. He spent the past two years as Program Director for Operations Research at the NSF while on leave from the University of Michigan. His email address is rlsmith@umich.edu.