Reinforcement Learning Based Algorithms For Average Cost Markov Decision Processes

Mohammed Shahid Abdulla Shalabh Bhatnagar

IISc-CSA-TR-2005-6

http://archive.csa.iisc.ernet.in/TR/2005/6/

Computer Science and Automation

Indian Institute of Science, India

May 2005

Reinforcement Learning Based Algorithms For Average Cost Markov Decision Processes*

Mohammed Shahid Abdulla and Shalabh Bhatnagar†

Department of Computer Science and Automation,

Indian Institute of Science,

Bangalore - 560 012, India.

E-mail:{shahid,shalabh}@csa.iisc.ernet.in

*This work was supported in part by Grant no. SR/S3/EE/43/2002-SERC-Engg from the Department of Science and Technology, Government of India.

†Corresponding Author, E-mail for correspondence: shalabh@csa.iisc.ernet.in

Abstract

This article proposes several two-timescale simulation-based actor-critic algorithms for solution of infinite horizon Markov Decision Processes with finite state-space under the average cost criterion. Two of the algorithms are for the compact (non-discrete) action setting while the rest are for finite-action spaces. On the slower timescale, all the algorithms perform a gradient search over corresponding policy spaces using two different Simultaneous Perturbation Stochastic Approximation (SPSA) gradient estimates. On the faster timescale, the differential cost function corresponding to a given stationary policy is updated and averaged for enhanced performance. A proof of convergence to a locally optimal policy is presented. Next, a memory efficient implementation using a feature-vector representation of the state-space and TD(0) learning along the faster timescale is discussed. The TD(0) algorithm does not follow an on-line sampling of states but is observed to do well on on our setting. Numerical experiments on rate based flow control on a bottleneck node using a continuous-time queueing model are presented using the proposed algorithms. We show performance comparisons of our algorithms with the two-timescale actor-critic algorithms of (Konda & Borkar, 1999). Our algorithms exhibit more than an order of magnitude better performance over those of (Konda & Borkar, 1999).

Keywords

Actor-critic algorithms, two timescale stochastic approximation, Markov decision processes, policy iteration, simultaneous perturbation stochastic approximation, normalized Hadamard matrices, reinforcement learning, TD-learning.

1 Introduction

While Dynamic Programming (DP) makes available a family of methods to solve Markov Decision Process (MDP) models, see for instance, (Puterman, 1994), it also assumes explicit knowledge of the system dynamics through the transition probabilities. Simulation-based schemes for solving MDPs have recently been used for systems when one has no knowledge of the system dynamics but can simulate the transitions of the system, see (Bertsekas & Tsitsiklis, 1996), (Bhatnagar & Kumar, 2004), (Borkar & Konda, 2004), (Konda & Borkar, 1999), (Konda & Tsitsiklis, 2003), (Tsitsiklis & Van Roy, 1997), (Tsitsiklis & Van Roy, 1999), and (Van Roy, 2001). Even with complete knowledge of the transition probabilities, DP methods suffer from 'the curse of dimensionality', where by the computational requirements to explicitly solve the Bellman equation become prohibitive when state spaces are large. This problem is mitigated by parameterizing either or both the value-function and the policy. The parameterized form of the policy is known as the 'actor' while the 'critic' is an analogous manifestation of the value-function.

Policy iteration is one of the classical methods for solving MDPs (cf. (Bertsekas, 1976, section 8.3)) that is performed using two loops - a policy update on the outer loop being performed once the inner loop that computes the stationary value function for a given policy estimate has converged. The outer loop's 'wait' for convergence of the inner loop is avoided when two timescale stochastic approximation is used. In such a setting, the policy evaluation recursions proceed on a faster timescale whereas the policy updates, using the above evaluation estimates, are performed on a slower timescale. A simulation based, two timescale, approach for policy iteration is considered by (Konda & Borkar, 1999) for MDPs with finite state and finite action spaces. In (Konda & Tsitsiklis, 2003), the value function is parameterized using a linear approximation architecture and two timescale algorithms are used where on the faster scale, a temporal difference (TD) type learning algorithm, see (Tsitsiklis & Van Roy, 1999), is used and on the slower scale, an approximate gradient search is performed.

For noisy measurement based parameter optimization, a gradient descent stochastic approximation algorithm known as simultaneous perturbation stochastic approximation (SPSA) was proposed in (Spall, 1992) that uses only two cost function measurements for any N-dimensional parameter vector. Further, (Spall, 1997) proposed a similar method that used only one measurement. The algorithm of (Spall, 1992) has been seen to perform well in a variety of settings by several authors. The algorithm of (Spall, 1997), however, does not perform as well (as (Spall, 1992)) because of the presence of additional bias terms in its gradient estimate. The algorithms of (Spall, 1992) and (Spall, 1997) perturb the parameter vector randomly in all components by using independent and identically distributed (i.i.d.), symmetric mean-zero random variables. In (Bhatnagar *et al.*, 2003), perturbations based on certain deterministic sequences for two-timescale SPSA algorithms were proposed and a simulation based optimization setting was considered. It was observed in (Bhatnagar *et al.*, 2003) that algorithms that use deterministic perturbations performed better than those that use randomized perturbations on the settings considered therein. In particular, the improvement in performance of one-simulation algorithms that use certain normalized Hadamard matrix based perturbations, over those that use randomized perturbations was found to be significant. The SPSA approach to gradient approximation has recently been used in (Bhatnagar & Kumar, 2004) to compute the optimal policy in infinite-horizon discounted-cost MDPs and in (Bhatnagar & Abdulla, 2005) for finding the optimal policy in the case of MDPs under the finite horizon total cost criterion.

Long run average-cost problems are studied in areas such as communication networks where steady-state system performance is of interest. This paper is concerned with computing an optimal policy for the long run average cost, extending the work of (Bhatnagar & Kumar, 2004) in several ways. In (Bhatnagar & Kumar, 2004), the problem considered was of the discounted cost criterion. Moreover, the action sets were considered to be compact (non-discrete). Here we consider not just compact action sets but also those that are discrete (finite). We consider both one and two simulation algorithms that use deterministic Hadamard matrix based perturbations whereas in (Bhatnagar & Kumar, 2004), only the two simulation approach using randomized perturbations was considered. We employ an actor-critic approach similar to (Konda & Borkar, 1999) and (Konda & Tsitsiklis, 2003) with the difference that the actor updates in our algorithm use suitable SPSA policy gradient estimates. Moreover, the critic updates have an extra averaging step for enhanced performance. Our algorithms show more than an order of magnitude better performance over the algorithms of (Konda & Borkar, 1999).

The critic used is an estimate of the differential cost function $h(\cdot)$. In the general case, we perform updation of the critic by simulating transitions out of every state in the MDP state-space S. The controls belong to the control space C, a possibly infinite set, and are specified by the present iterate of the (parameterized) policy π. Such an implementation of the critic is amenable to parallelization when multiple processors are available, each capable of simulating the system transitions. Though the critic updates are performed synchronously in (Konda & Borkar, 1999), an associated stability test to make the critic-updates asynchronous was suggested in (Borkar & Meyn, 2000), and that permits therefore the above-mentioned parallelization. The actor updates in the algorithms proposed in (Konda & Borkar, 1999) converge to the optimal policy using stochastic approximation with 'reinforcement-signals' of varied types. All these signals require actor-specific simulation to be performed that results in a computational effort of $O(|C|)$ in each update step of the actor. In contrast, we employ SPSA estimates of the gradient (as do (Bhatnagar *et al.*, 2003), (Bhatnagar & Kumar, 2004) and (Bhatnagar & Abdulla, 2005)) that do not cause any simulation load *specific* to the actor. Thus the processors responsible for updating the actor parameters here need not simulate any transitions of the system. This results in a significant reduction in the overall computational effort (as is also seen from our experiments) in the case of our algorithms over those of (Konda & Borkar, 1999).

Estimates of $h(i)$, $\forall i \in S$, are stored in a size-$|S|$ look-up table with an entry for each state $i \in S$. Such a look-up table becomes prohibitive in size for large state-spaces. We also propose variants that use a smaller, K-dimensional coefficient vector with $K << |S|$ for the critic update. This vector is then used to approximate $h(i)$ via a linear approximation architecture, using a feature-vector representation of state i, denoted $\phi(i)$. This is similar to the method of (Tsitsiklis & Van Roy, 1999), but with some key differences.

The rest of the article is organized as follows: §2 identifies the setting, notation and a generic form of the proposed algorithms. In §3, the Hadamard matrix based construction of deterministic perturbation sequences is presented. The algorithms are proposed in §4,§5 and §6 respectively, for which the convergence analysis is shown in §7. The linear approximation method (above) is described in §8 while §9 provides the numerical results in the setting of flow control in communication networks. We conclude and identify some future directions in §10.

2 Framework and General Algorithm

We consider an MDP $\{X_t, t = 0, 1, ...\}$ where decisions are made at instants $t = 0, 1, ...$, using an associated control-valued process $\{Z_t\}$. Suppose $S \equiv \{1, 2, ..., s\}$ is the (finite) state space and C is the control space. Suppose also that $U(i) \subseteq C$ is the set of all feasible controls in state i. Let $p(i, u, j), i, j \in S, u \in U(i)$ be the transition probabilities associated with this MDP. An admissible policy $\mu = \{\mu^0, \mu^1, \mu^2, ...\}$ with $\mu^t : S \mapsto C$ is one for which $\mu_i^t \in U(i)$, $\forall t \in \{0, 1, ...\}$, $\forall i \in S$. We call $\mu = \{\mu^0, \mu^1, \mu^2, ...\}$ a stationary policy when $\mu^t = \pi$, $\forall t \in \{0, 1, ...\}$, i.e., if the policy is time/stage invariant. For simplicity, we denote using $\pi = (\pi_i,\ i \in S)^T$, a stationary policy. Suppose $K(i_t, u_t, i_{t+1})$ denotes the one-step transition cost when the current state is $i_t \in S$, the action chosen is $u_t \in U(i_t)$ and the next state is $i_{t+1} \in S$. The aim here is to find an admissible policy π that minimizes the associated infinite horizon average cost λ_π, where

$$\lambda_\pi = \lim_{N \to \infty} \frac{1}{N} E \left\{ \sum_{t=0}^{N-1} K(i_t, \pi_{i_t}, i_{t+1}) \right\},$$

starting from any initial state i_0. We assume here that the resulting chain under every stationary policy is aperiodic, irreducible (and hence also positive recurrent). A weaker assumption requiring that the resulting chain under every stationary policy be unichain (cf. (Puterman, 1994, Theorem 8.4.3)), viz. a chain with a single class of recurrent states and a possibly empty set of transient states, would also suffice. For an MDP under stationary policy π, with steady-state probabilities $q_\pi(i)$ of the resulting chain, $\forall i \in S$ (obtained using $q_\pi(i) = \sum_{j=1}^{s} p(j, \pi_j, i) q_\pi(j)$) one obtains

$$\lambda_\pi = \sum_{i=1}^{s} q_\pi(i) \left(\sum_{j=1}^{s} p(i, \pi_i, j) K(i, \pi_i, j) \right).$$

The corresponding differential cost function is denoted $h_\pi(\cdot)$ and satisfies the Poisson Equation:

$$\lambda_\pi + h_\pi(i) = \sum_{j=1}^{s} p(i, \pi_i, j)(K(i, \pi_i, j) + h_\pi(j)),\ \forall i \in S, \tag{1}$$

when action π_i is used in state i. A unique solution for (1) can be obtained if we set $h_\pi(i_0) = \lambda_\pi$ for a prescribed $i_0 \in S$ that we call as a reference state. Suppose we define

$$G_\pi(i, h) \stackrel{\triangle}{=} \sum_{j=1}^{s} p(i, \pi_i, j)(K(i, \pi_i, j) + h(j)), \tag{2}$$

then $G_\pi(\cdot,\cdot)$ satisfies, for every stationary policy π, $G_\pi(i,h_\pi) = h_\pi(i) + h_\pi(i_0),\ \forall i \in S.$

We consider two types of stationary policies π viz., randomized and deterministic. For a system operating under a deterministic policy π, the action $\pi_i \in U(i)$ is applied when in state i. It is for such systems that equation (1) holds. In contrast, a randomized stationary policy (RSP) π is denoted (via abuse of notation) as the vector $\pi = (\pi_i,\ \forall i \in S)^T$ with each $\pi_i = (\pi_i^j, 0 \leq j \leq N_i)$, where π_i^j is the probability of picking action $u_i^j \in U(i)$. Here $U(i)$ is a finite set with (N_i+1) elements $\{u_i^0, u_i^1, ..., u_i^{N_i}\}$. In practice, an RSP is applied by generating, when in state i, a $U(i)-$valued random variable $\xi(i,\pi_i)$ distributed according to the probability mass function (p.m.f.) π_i. Note that to generate $\xi(i,\pi_i)$, only N_i of the N_i+1 available weights π_i^j are sufficient. Thus, the vector $\hat{\pi}_i = (\pi_i^1, \pi_i^2, ..., \pi_i^{N_i})^T$, $\forall i \in S$, $1 \leq j \leq N_i$, is sufficient to denote the 'control' π_i, since $\pi_i^0 = 1 - \sum_{j=1}^{N_i} \pi_i^j$. For the sake of uniformity in notation across all policies, we assume that $\xi(i,\pi_i)$ is also used when the deterministic policy π is applied, except that in such a case $\xi(i,\pi_i) = \pi_i$.

Since we will require to generate multiple instances of $\xi(i,\pi_i)$ during simulation, a subscript (m in $\xi_m(i,\pi_i)$) will identify the particular random variable in a stochastic process $\{\xi_n(\cdot,\cdot)\}, \forall n \geq 0$ of independently generated random variables ξ. Similarly, a superscript (r in $\xi_m^r(i,\pi_i)$) will identify a particular stochastic process. We describe simulated transitions of the system operating under a policy π, using the $S-$valued random variable $\eta(i,\xi(i,\pi_i))$ which is distributed according to the law $p(i,\xi(i,\pi_i),\cdot)$. Here $\eta(i,\xi(i,\pi_i))$ indicates the state to which the system transits due to application of action $\xi(i,\pi_i)$ when in state i. With the same connotation as for ξ, we will use subscripts and superscripts for the random variable η also.

For the reasons explained above, the analogous Poisson equation for an RSP π differs slightly from (1), and is as follows:

$$\lambda_\pi + h_\pi(i) = \sum_{j=1}^{s} \sum_{k=0}^{N_i} \pi_i^k p(i, u_i^k, j) \left(K(i, u_i^k, j) + h_\pi(j) \right), \ \forall i \in S. \tag{3}$$

As in (1), a unique solution for (3) can be obtained if we set $h_\pi(i_0) = \lambda_\pi$ for a prescribed $i_0 \in S$. Further, $G_\pi(i, h_\pi)$ can also be analogously defined.

We explain the mechanism used in the coupled stochastic recursions that we employ. On the faster timescale, our algorithms use either one or two simulations that are identified by the index r taking values in the sets $\{2\}$ or $\{1,2\}$, respectively. We use the generic r to qualify statements that hold for both cases above. At the end of a given L steps of each simulation, we update π_i once and denote by $\pi_i(n)$ the $n-$th update of π_i. Here $\pi_i(n)$ in general is a vector with N_i elements, viz. $\pi_i(n) = (\pi_i^1(n), ..., \pi_i^{N_i}(n))^T$. Thus, the policy $\pi(n) = (\pi_1(n), \pi_2(n), ..., \pi_s(n))^T$ is a column vector of size $\sum_{i=1}^{s} N_i$. The two simulations mentioned above correspond to the system operating under two perturbed policies, $\pi^1(n)$ and $\pi^2(n)$, respectively. The differential-cost estimates, $h_i^1(nL)$ and $h_i^2(nL)$, of $h_{\pi^1(n)}(i)$ and $h_{\pi^2(n)}(i)$ respectively, (see (5) below) corresponding to the above perturbed policies are computed in the course of updates $(n-1)L$ to nL of the two simulations. We denote the general policy-gradient estimate as $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)}) = (\tilde{\bigtriangledown}_i^j G_{\pi(n)}(i, h_{\pi(n)}), 1 \leq j \leq N_i)^T$. Here, $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)})$ stands for the N_i-element vector gradient estimate of $G_{\pi(n)}(i, h_{\pi(n)})$ w.r.t. $\pi_i(n)$. The policy gradient estimate depends on the 'critic' estimates $h_i^r(nL)$. In particular, we propose forms of $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)})$ that use either $h_i^2(nL)$ or both $h_i^1(nL)$ and $h_i^2(nL)$, respectively (see (9)-(12) below).

A projection operator $P_i(\cdot)$ is used to ensure that the updated actions as given by the algorithms remain feasible. We use perturbed policies $\pi^r(n)$, obtained from $\pi(n)$ (explained later, see (8)), for driving recursions (5) below.

We have the following general form for our algorithms: For all $i \in S$

$$\pi_i(n+1) = P_i\left(\pi_i(n) - c(n)\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)})\right). \tag{4}$$

Since the estimate $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, \cdot)$ above requires critic estimates $h_i^r(nL)$, $1 \leq i \leq s$ with $r \in \{2\}$ or $r \in \{1, 2\}$ (depending on whether one or two simulations are used), we perform the following L updates corresponding to $0 \leq m \leq L-1$ of these estimates: For all $i \in S$

$$\begin{aligned} h_i^r(nL+m+1) &= (1-b(n))h_i^r(nL+m) \\ &\quad + b(n)(K(i, \xi^r_{nL+m}(i, \pi_i^r(n)), \eta^r_{nL+m}(i, \xi^r_{nL+m}(i, \pi_i^r(n)))) \\ &\quad - h^r_{i_0}(nL+m) + h^r_{\eta^r_{nL+m}(i, \xi^r_{nL+m}(i, \pi_i^r(n)))}(nL+m)). \end{aligned} \tag{5}$$

The precise relationship between $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)})$ and $h_i^r(nL)$ will be made clear as the algorithms are introduced in §4-§6. It was observed in (Bhatnagar *et al.*, 2001b) that SPSA based algorithms in the setting of simulation optimization improved performance when an L-step additional averaging as above is used. The value of L can be chosen arbitrarily in practice. We use $L = 100$ in our experiments. Note that recursions in (5) are common to all algorithms. These are similar to the adaptive-critic updates in the average-cost, finite-action setting of (Konda & Borkar, 1999, (3.5)), except that the above $L-$step averaging is not present in the algorithms in (Konda & Borkar, 1999). All six proposed versions of algorithm (4)-(5) have differences that pertain to the dissimilarities in the structure of $U(i)$ or the gradient estimate $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)})$ of (4).

The requirements on the step-sizes in (4)-(5) are as follows:

$$b(n), c(n) > 0, \forall n \geq 0$$

$$\sum_n b(n) = \sum_n c(n) = \infty, \quad \sum_n b(n)^2, \sum_n c(n)^2 < \infty, \tag{6}$$

and

$$c(n) = o(b(n)), \tag{7}$$

respectively.

The extra averaging of the recursion (5) over L iterations, besides being beneficial for convergence (cf. (Bhatnagar & Kumar, 2004), (Bhatnagar *et al.*, 2003), (Bhatnagar *et al.*, 2001b)), has an added relevance to any asynchronous implementation involving multiple parallel processors each updating the differential cost estimate for a given state. Note that it is sufficient if we choose L such that $L > 2D$ where D is the upper bound on inter-processor communication delay (cf. (Konda & Borkar, 1999, Assumption A5)), since in the recursion in (4), at iteration n, the 'current' policy gradient estimate $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)})$ can still be used rather than the need to use any previous estimates $\tilde{\bigtriangledown}_i G_{\pi(m)}(i, h_{\pi(m)})$, $0 \leq m < n$.

3 Construction for Perturbation Sequences $\Delta_i(n)$

Perturbation vectors $\Delta_i(n),\ \forall i \in S, n \geq 0$, used in the perturbed policies $\pi_i^1(n)$ and $\pi_i^2(n)$ are obtained as under. The construction used is as in (Bhatnagar *et al.*, 2003, Section 3). Consider H_{d_i} as a normalized Hadamard matrix (a Hadamard matrix is said to be normalized if all the elements of its first row and column are 1s) of order d_i with $d_i \geq N_i + 1,\ \forall i \in S$. Let $h^i(1), h^i(2), ..., h^i(N_i)$ be any N_i columns other than the first column of H_{d_i}, and form a new $d_i \times N_i$ dimensional matrix $\tilde{H}_{d_i}$ which has the above as its columns. Let $\tilde{\Delta}(p),\ p = 1, ..., d_i$ be the d_i rows of $\tilde{H}_{d_i}$. Now set $\Delta_i(n) = \tilde{\Delta}(n \bmod d_i + 1)$, $\forall n \geq 0$. The perturbations are thus generated by cycling through the rows of $\tilde{H}_{d_i}$. Matrices H_{d_i}, with $d_i = 2^{k_i}$, are systematically constructed as follows:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

$$H_{2^{k_i}} = \begin{pmatrix} H_{2^{k_i-1}} & H_{2^{k_i-1}} \\ H_{2^{k_i-1}} & -H_{2^{k_i-1}} \end{pmatrix},$$

for $k_i > 1$, respectively.

One-simulation algorithms require $d_i = 2^{\lceil \log_2 (N_i+1) \rceil}$, $\forall i \in S$, whereas two-simulation algorithms require $d_i = 2^{\lceil \log_2 N_i \rceil}$, $\forall i \in S$ (see (Bhatnagar *et al.*, 2003)). In what follows, we consider two different settings for the action sets $U(i)$ viz., compact (non-discrete) action sets and discrete (finite) action sets, respectively, and devise algorithms for both settings.

4 Algorithms for Compact Action Sets

Suppose $U(i), i \in S$, are of the form $U(i) = \prod_{j=1}^{N_i} [a_i^j, b_i^j]$. We make the following assumption:

Assumption(A): For all $i, j \in S, a \in U(i)$, both $K(i,a,j)$ and $p(i,a,j)$ are continuously differentiable in a.

Let $P_i^j(y) = \min(b_i^j, \max(a_i^j, y)), y \in \mathcal{R}$, be the projection of y onto the interval $[a_i^j, b_i^j]$, $1 \leq j \leq N_i$. Further, for $y = (y_1, y_2 ..., y_{N_i})^T \in \mathcal{R}^{N_i}$, let $P_i(y) = (P_i^1(y_1), ..., P_i^{N_i}(y_{N_i}))^T$. Then $P_i(y)$ denotes the projection of $y \in \mathcal{R}^{N_i}$ to the set $U(i)$, $i \in S$. Using the method of §3, we obtain $\{\pm 1\}^{N_i}$-valued vectors $\Delta_i(n) \stackrel{\triangle}{=} (\Delta_i^1(n), ..., \Delta_i^{N_i}(n))^T$, $\forall i \in S, n \geq 0$ to construct the two perturbed policies, $\pi^1(n)$ and $\pi^2(n)$ respectively. Here, analogous to π, we have $\pi^r(n) = (\pi_1^r(n), \pi_2^r(n), ..., \pi_s^r(n))^T$, with $\pi_i^r(n)$ defined as follows:

$$\pi_i^1(n) \quad = \quad P_i(\pi_i(n) - \delta \Delta_i(n)),$$

$$\pi_i^2(n) \quad = \quad P_i(\pi_i(n) + \delta\Delta_i(n)), \tag{8}$$

respectively. Let the iterates $h_i^r(n),\ \forall i \in S, n \geq 0$ be defined as in (5) with

$$h_i^r(0) = h_i^r(1) = ... = h_i^r(L-1) = 0,\ \forall i \in S.$$

Denote $(\Delta_i(n))^{-1}$ as

$$(\Delta_i(n))^{-1} = \left(\frac{1}{\Delta_i^1(n)}, ..., \frac{1}{\Delta_i^N(n)}\right)^T, \forall i \in S.$$

We are now in a position to describe the first two algorithms. We adopt acronyms for all the algorithms in order to avoid expanded descriptions. In the following, 'ACA' stands for Algorithms for Compact Action sets (similar to the notation used by (Bhatnagar & Abdulla, 2005)). The numeral 1 or 2 at the end of the algorithm name identifies if the algorithm needs 1 or 2 simulations, respectively.

ACA-1

For all $i \in S$, substitute for the gradient estimate $\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)})$ in (4):

$$\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i_0}^2(nL))}{\delta} (\Delta_i(n))^{-1} \tag{9}$$

where for $m = 0, 1, ..., L-1$, we perform the recursion (5) for $r = 2$ to compute the critic estimates $h_i^2(nL)$ required above.

ACA-2

For all $i \in S$, we propose a finite-difference policy gradient estimate where

$$\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i_0}^2(nL)) - (h_i^1(nL) + h_{i_0}^1(nL))}{2\delta} (\Delta_i(n))^{-1} \tag{10}$$

where for $m = 0, 1, ..., L-1$, we perform recursion (5) for both $r = 1, 2$ in order to compute $h_i^2(nL)$ and $h_i^1(nL)$, respectively. As stated before, ACA-2 uses two simulations while ACA-1 uses only one. Next, we present two analogs of this algorithm for finite action sets.

5 Algorithms For Finite Action Sets using RSPs

Suppose $U(i), i \in S$, be finite action sets of the form $U(i) = \{u_i^0, u_i^1, ..., u_i^{N_i}\}$, where $u_i^j, 0 \leq j \leq N_i$ are feasible actions in state i. We assume:

Assumption (B): The cost functions $K(i, a, j),\ i, j \in S, a \in U(i)$, are bounded.

Let $S_i = \{(y^1, ..., y^{N_i})^T | y^j \geq 0, \forall j \in \{1, 2, ..., N_i\}, \sum_{j=1}^{N_i} y^j \leq 1\}$ denote the simplex of the probabilities of selecting actions in $U(i) \backslash \{u_i^0\} = \{u_i^1, ..., u_i^{N_i}\}$. We now define projection $P_i : \mathcal{R}^{N_i} \mapsto S_i$, such that for any $x \in \mathcal{R}^{N_i}$, $P_i(x)$ is the closest point in S_i to x. Let $\hat{\Delta}_i(n)$ be a vector of $\{\pm 1\}^{N_i}-$valued perturbations generated in the same manner as the $\Delta_i(n)$ used in ACA-r. We have seen earlier that $\hat{\pi} = (\hat{\pi}_1, \hat{\pi}_2, ..., \hat{\pi}_s)^T$ identifies an RSP π completely and that $\hat{\pi}_i \in S_i,\ \forall i \in S$.

Let $\hat{\pi}_i^1(n) = (P_i(\hat{\pi}_i(n) - \delta\hat{\Delta}_i(n)),\ \forall i \in S)^T$ and $\hat{\pi}_i^2(n) = (P_i(\hat{\pi}_i(n) + \delta\hat{\Delta}_i(n)),\ \forall i \in S)^T$, respectively, denote the perturbed policies. Let

$$(\hat{\Delta}_i(n))^{-1} = \left(\frac{1}{\hat{\Delta}_i^1(n)}, ..., \frac{1}{\hat{\Delta}_i^{N_i}(n)}\right)^T,$$

as before. In the following, 'RPAFA' stands for Randomized Policy Algorithms over Finite Action sets.

RPAFA-1

Replace $\pi_i(n)$ and $\pi_i(n+1)$ in (4) with $\hat{\pi}_i(n)$ and $\hat{\pi}_i(n+1)$, respectively. As was done in ACA-1 (cf. (9)), for all $i \in S$ substitute for $\tilde{\bigtriangledown}_i G_{\pi(n)}(i, h_{\pi(n)})$ of (4) with:

$$\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i_0}^2(nL))}{\delta}\left(\hat{\Delta}_i(n)\right)^{-1} \tag{11}$$

where for $m = 0, 1, ..., L-1$, we perform (5) with $r = 2$. Note that, as seen earlier in (5), the action taken in state i i.e. $\xi^2_{nL+m}(i, \pi_i^2(n))$ is now a $U(i)-$ valued random variable that selects actions according to the $(N_i + 1)-$element p.m.f. $\pi_i^2(n)$, unlike in ACA-2 where $\xi^2_{nL+m}(i, \pi_i^2(n))$ was the deterministic action $\pi_i^2(n)$ itself. This therefore results in a different form of the Poisson equation (cf. (3)) that is tracked by the recursion (5) for a given RSP as explained earlier.

RPAFA-2

Replace $\pi_i(n)$ and $\pi_i(n+1)$ in (4) with $\hat{\pi}_i(n)$ and $\hat{\pi}_i(n+1)$, respectively. Analogous to ACA-2 (cf. (10)), the policy gradient estimate is:

$$\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i_0}^2(nL)) - (h_i^1(nL) + h_{i_0}^1(nL))}{2\delta}(\hat{\Delta}_i(n))^{-1} \tag{12}$$

where for $m = 0, 1, ..., L-1$, recursion (5) is performed for both $r = 1, 2$. As with RPAFA-1, $\xi^r_{nL+m}(i, \pi_i^r(n))$ are also $U(i)-$valued random variables.

6 Algorithms for Finite Action sets using Deterministic Policies

In order to develop the remaining two algorithms for finite action sets, we make some further assumptions on the structure of $U(i)$, $\forall i \in S$ used in RPAFA-r above. Let $U(i)$ be the feasible action set such that $U(i) = \prod_{j=1}^{N_i} U^j(i)$, where $U^j(i)$ are finite sets $\{u_i^j(1), u_i^j(2), ..., u_i^j(N_i^j)\}$

and $|U^j(i)| = N_i^j$. An admissible control $\pi_i \in U(i),\ \forall i \in S$ is such that $\pi_i^j \in U^j(i),\ 1 \leq j \leq N_i$. The additional requirement is that the convex hull of $U(i)$ be the compact set $\hat{U}(i) = \prod_{j=1}^{N_i}[a_i^j, b_i^j]$. Without loss of generality, if we assume that $u_i^j(1) \leq u_i^j(2) \leq ... \leq u_i^j(N_i^j)$, then in the above $a_i^j = u_i^j(1)$ and $b_i^j = u_i^j(N_i^j),\ \forall i \in S$, respectively. Note that $\hat{U}(i)$ qualifies as the feasible action set for state i in algorithms ACA-r. Though the feasible action set $U(i)$ here is finite just as in RPAFA-r, the policy $\pi(n)$ used in the algorithm at each update n is deterministic. A similar idea is used by (Gerencser *et al.*, 1999) where a measurement based stochastic approximation scheme is applied for a static resource allocation optimization problem.

In the proposed algorithm, the policy iterates $\pi_i(n)$ evolve in the convex hull $\hat{U}(i)$ to project into which the projection P_i as used in ACA-r is employed. Thus, the perturbed deterministic policies at the $n-$th iteration are $\pi^1(n) = (P_i(\pi_i(n) - \delta\Delta_i(n)), \forall i \in S)^T$ and $\pi^2(n) = (P_i(\pi_i(n) + \delta\Delta_i(n)), \forall i \in S)^T$, where $P_i : \mathcal{R}^{N_i} \mapsto \hat{U}(i)$ is such that $P_i^j(x_i^j) = \min(\max(a_i^j, x_i^j), b_i^j),\ 1 \leq j \leq N_i$. Note that the policy $\pi^r(n)$ to be applied in (5) need not be admissible. It is made so by another projection $\bar{P}_i(\pi_i^r(n))$ i.e. $\xi_{nL+m}(i, \pi_i^r(n)) = \bar{P}_i(\pi_i^r(n))$. Here, $\bar{P}_i(\pi_i^r(n)) = \arg\min_{u \in U(i)} \|\pi_i^r(n) - u\|$ is the map into the feasible set $U(i)$ of the control $\pi_i^r(n) \in \hat{U}(i)$. If the minimum above is attained at two points, one of the two is arbitrarily selected. In contrast, ACA-r algorithms apply $\xi_{nL+m}(i, \pi_i^r(n)) = \pi_i^r(n)$ in recursion (5). The letters 'DPAFA' in the following stand for Deterministic Policy Algorithms over Finite Action sets.

DPAFA-1

Redefine $\eta_{nL+m}^r(i, \xi_{nL+m}^r(i, \pi_i^r(n))), \forall n, m > 0, r = 2$, in (5) to be independent families of i.i.d. random variables with distribution $p(i, \bar{P}_i(\pi_i^r(n)), \cdot)$ with $\xi_{nL+m}^r(i, \pi_i^r(n)) = \bar{P}_i(\pi_i^r(n))$

as described above. DPAFA-1 is now the same as ACA-1.

DPAFA-2

This algorithm is similar to ACA-2, using $\eta^r_{nL+m}(\cdot,\cdot)$ and $\xi^r_{nL+m}(\cdot,\cdot)$ as described above, for both $r = 1$ and 2.

7 Convergence Analysis

We use Assumption (A) first and prove convergence for ACA-r. The differential cost $h_\pi(i),\ \forall i \in S$, for an infinite-horizon MDP operating under a stationary deterministic policy π can be obtained as the unique solution of the following linear system of s equations (i.e., the Poisson equation, also identified in (1)):

$$G_\pi(i, h_\pi) = h_\pi(i_0) + h_\pi(i) = \sum_{j\in S} p(i,\pi_i,j)(K(i,\pi_i,j) + h_\pi(j)). \tag{13}$$

From Assumption (A), it is easy to see that $K(i, a, j)$ and $h_\pi(i)$ are uniformly bounded functions. Moreover, $h_\pi(i)$ is continuously differentiable in $\pi, \forall i \in S$.

Lemma 1 *The iterates $h^r_i(n)$, satisfy* $\sup_{n\geq 0} |h^r_i(n)| < \infty, \forall i \in S, r = 1, 2$.

Proof: Let $\tilde{b}(n) = b([\frac{n}{L}]),\ n \geq 0$, where $[\frac{n}{L}]$ denotes the integer part of $\frac{n}{L}$. Then the recursion in (5) can be written as

$$\begin{aligned} h^r_i(n+1) \quad &= \quad (1-\tilde{b}(n))h^r_i(n) + \tilde{b}(n)(K(i,\xi^r_n(i,\tilde{\pi}^r_i(n)), \eta^r_n(i,\xi^r_n(i,\tilde{\pi}^r_i(n)))) \\ &\quad - h^r_{i_0}(n) + h^r_{\eta^r_n(i,\xi^r_n(i,\tilde{\pi}^r_i(n)))}(n)) \end{aligned}$$

where $\tilde{\pi}^r_i(n) = \pi^r_i([\frac{n}{L}])$. The above is now analogous to recursion (3.5) of (Konda & Borkar, 1999). The proof now follows in a similar manner as (Konda & Borkar, 1999, Lemma 6.4).

$\Box$

Now define $\{s(n)\}$ according to $s(0) = 0, s(n) = \sum_{k=0}^{n-1} c(k), n \geq 1$. Let $\Delta_i(t) \stackrel{\triangle}{=} (\Delta_i^1(t), ..., \Delta_i^{N_i}(t))^T$ be defined according to $\Delta_i(t) = \Delta_i(n)$ for $t \in [s(n), s(n+1)], n \geq 0, i \in S$. Suppose that for any bounded, continuous and real-valued function $v(\cdot)$,

$$\hat{P}_i^j(v(y)) = \lim_{\eta \to 0} \left(\frac{P_i^j(y + \eta v(y)) - y}{\eta} \right)$$

for $j = 1, ..., N_i, i \in S$. Further for any $y = (y_1, ..., y_{N_i})^T$, let $\hat{P}_i(y) = (\hat{P}_i^1(y_1), ..., \hat{P}_i^{N_i}(y_{N_i}))^T$. For a given stationary deterministic policy $\pi = (\pi_i, i \in S)^T$, suppose

$$F_\pi(i, h) \stackrel{\triangle}{=} \sum_{j \in S} p(i, \pi_i, j)(K(i, \pi_i, j) + h_j) - h_{i_0},$$

where h is any vector $(h_j,\ j \in S)^T$. The Poisson equation (13) can now be written as

$$h_\pi(i) = F_\pi(i, h_\pi) = G_\pi(i, h_\pi) - h_\pi(i_0). \tag{14}$$

Consider $\{t(n)\}$ defined as $t(0) = 0, t(n) = \sum_{j=0}^{n-1} \tilde{b}(j), n \geq 1$ where $\tilde{b}(n) = b([\frac{n}{L}])$ and the continuous-time policies $\tilde{\pi}(t) = \pi(k)$ when $t \in [t(k), t(k+1))$. Further, consider the system of ODEs: for all $i \in S$,

$$\dot{\tilde{\pi}}_i^r(t) = 0, \tag{15}$$

$$\dot{h}_i^r(t) = F_{\tilde{\pi}^r(t)}(i, h^r(t)) - h_i^r(t). \tag{16}$$

Here $\tilde{\pi}_i^1(t) = P_i(\tilde{\pi}_i(t) - \delta\Delta_i(t))$ and $\tilde{\pi}_i^2(t) = P_i(\tilde{\pi}_i(t) + \delta\Delta_i(t))$, respectively for all $i \in S$. Note that (15) corresponds to a fixed policy $\tilde{\pi}^r$ that is independent of t. Thus, (16) can be written as (with a time invariant $\tilde{\pi}^r$)

$$\dot{h}_i^r(t) = F_{\tilde{\pi}^r}(i, h^r(t)) - h_i^r(t), \tag{17}$$

$\forall i \in S$. The ODE (17) is (cf. (Konda & Borkar, 1999, Lemma 6.3)) an asymptotically stable linear system with $h_{\tilde{\pi}^r}(i), \forall i \in S$ as its unique asymptotically stable equilibrium point. Let $\bigtriangledown_i h_\pi(i)$ denote the N_i-vector gradient of $h_\pi(i)$ w.r.t. π_i. Consider now the ODE:

$$\dot{\pi}_i(s) = \hat{P}_i(- \bigtriangledown_i h_{\pi(s)}(i) - \bigtriangledown_i h_{\pi(s)}(i_0)) = \hat{P}_i(-\bigtriangledown_i G_{\pi(s)}(i, h_{\pi(s)})) \tag{18}$$

for all $i \in S$. Let $M = \{\pi \mid \hat{P}_i(\bigtriangledown_i h_{\pi(s)}(i) + \bigtriangledown_i h_{\pi(s)}(i_0)) = 0,\ \forall i \in S\}$ be the set of all fixed points of (18). Also, given $\epsilon > 0$, let $M^\epsilon = \{\pi \mid \exists \bar{\pi} \in M\ s.t.\ \|\pi - \bar{\pi}\| < \epsilon\}$ be the set of all policies that are within a distance ϵ from M.

Applying standard martingale analysis on (5) using (6) and Gronwall's inequality upon the ODE (17), we can see along similar lines as (Bhatnagar & Kumar, 2004, Corollary 4.2),

Lemma 2 *For all* $i \in S$,$r = 1, 2$ $\lim_{n\to\infty} |h_i^r(n) - h_{\pi^r(n)}(i)| = 0$ *w.p. 1.* □

We now note that the slower time-scale recursion (4) can be rewritten as

$$\pi_i(n+1) = P_i(\pi_i(n) - \tilde{b}(n)\xi(n)) \tag{19}$$

where $\xi(n) = o(1)$ since $c(n) = o(\tilde{b}(n))$. From (19) and Lemma 2, algorithms ACA-r can be seen to asymptotically track the trajectories of the ODE (15)-(16) along the faster timescale $\{t(n)\}$. One now needs to show that on the slower timescale, $\pi(n)$ converges to M in the limit as $\delta \to 0$. We first consider the case of ACA-2. Define $U^o \equiv \prod_{i=1}^s U^o(i)$ as the interior of the space $U = \prod_{i=1}^s U(i)$ of all feasible policies.

Theorem 1 *Under Assumption (A), given* $\eta > 0$*, there exists* $\delta_0 > 0$ *such that for all* $\delta \in (0, \delta_0]$, $\{\pi(n)\}$ *of Algorithm ACA-2 converges to* M^η *with probability 1.*

Proof: We assume that $\forall i \in S$, $N_i = N$, the general case being a minor modification. Fix $1 \le k_0 \le N$. We may rewrite recursion (4) as follows:

$$\pi_i^{k_0}(m+1) = \pi_i^{k_0}(m) - c(m)\tilde{\bigtriangledown}_i^{k_0} G_{\pi(m)}(i, h_{\pi(m)}) + c(m) Z_i^{k_0}(m), \forall i \in S, k_0 \in \{1, 2, ..., N\}$$

where $Z_i^{k_0}(m)$ represent error terms due to projection $P_i^{k_0}$. Define $H_i^{k_0}(m) = -\tilde{\bigtriangledown}_i^{k_0} G_{\pi(m)}(i, h_{\pi(m)})$, then using (10), we have

$$H_i^{k_0}(m) = \frac{h_{\pi^1(m)}(i) + h_{\pi^1(m)}(i_0) - h_{\pi^2(m)}(i) - h_{\pi^2(m)}(i_0)}{2\delta\Delta_i^{k_0}(m)} + r_i^{k_0}(m),$$

where $r_i^{k_0}(m) = o(1)$ by Lemma 2. For a fixed $d \in \mathcal{Z}^+$, we may write the above iteratively as:

$$\pi_i^{k_0}(m+d) = \pi_i^{k_0}(m) + \sum_{l=m}^{m+d-1} c(l) H_i^{k_0}(l) + \sum_{l=m}^{m+d-1} c(l) Z_i^{k_0}(l), \ \forall i \in S. \tag{20}$$

Now using the fact that $h_{\pi(m)}$ is C^1 w.r.t $\pi(m)$, and performing a Taylor series expansion about $\pi(m)$, we have:

$$\begin{aligned} H_i^{k_0}(m) &= -\nabla_i^{k_0} h_{\pi(m)}(i) - \nabla_i^{k_0} h_{\pi(m)}(i_0) \\ &\quad - \sum_{q=1, q \neq k_0}^{N} \frac{\Delta_i^q(m)}{\Delta_i^{k_0}(m)} \nabla_i^q \left(h_{\pi(m)}(i) + h_{\pi(m)}(i_0) \right) + O(\delta). \end{aligned} \tag{21}$$

In the above, we assume that $\pi_i(m) \in U^o(i), \forall i \in S$ and $\delta > 0$ is small enough so that $\pi_i^r(m) \in U^o(i)$. For the case of $\pi_i^r(m) \in U(i) \backslash U^o(i)$, the above would continue to hold except for a constant multiplying the first RHS term in (21) (cf. Corollary 4.3 of (Bhatnagar & Kumar, 2004)). Now from (20) and (21), we have

$$\begin{aligned} \pi_i^{k_0}(m+d) &= \pi_i^{k_0}(m) - \sum_{l=m}^{m+d-1} c(l) \nabla_i^{k_0} \left(h_{\pi(l)}(i) + h_{\pi(l)}(i_0) \right) \\ &\quad - c(m) \sum_{l=m}^{m+d-1} \sum_{q=1, q \neq k_0}^{N} \frac{c(l)}{c(m)} \frac{\Delta_i^q(l)}{\Delta_i^{k_0}(l)} \nabla_i^q \left(h_{\pi(l)}(i) + h_{\pi(l)}(i_0) \right) \\ &\quad + \sum_{l=m}^{m+d-1} c(l) Z_i^{k_0}(l) + O(\delta). \end{aligned} \tag{22}$$

Here we take $d = 2^{\lceil \log_2 N \rceil}$. It can now be seen using Corollary 2.3 of (Bhatnagar *et al.*, 2003) that

$$\left| \sum_{l=m}^{m+d-1} \sum_{k=1, k \neq k_0}^{N} \frac{c(l)}{c(m)} \frac{\Delta_i^k(l)}{\Delta_i^{k_0}(l)} \nabla_i^k \left(h_{\pi(l)}(i) + h_{\pi(l)}(i_0) \right) \right| \stackrel{m \to \infty}{\to} 0. \tag{23}$$

Therefore, (22) is equivalent in an asymptotic sense to the recursion

$$\pi_i^{k_0}(n+1) = P_i^{k_0} \left(\pi_i^{k_0}(n) - c(n) \left(\nabla_i^{k_0} h_{\pi(n)}(i) + \nabla_i^{k_0} h_{\pi(n)}(i_0) \right) \right), \tag{24}$$

except for an additional error term which however is $O(\delta)$. It can now be seen, as in pp. 191-194 of (Kushner & Clark, 1978), that the above is a discretization of the projected ODE

(18). Finally, note that M is an asymptotically stable attractor set for the ODE (18) with $\sum_{i \in S} G_\pi(i, h_\pi)$ itself serving as the associated strict Liapunov function. The claim now follows. $\square$

With the same arguments, the above theorem also holds for RPAFA-2, the difference being that since randomized policies are now used, the Poisson equation tracked on the faster scale by a given randomized policy is (3) instead of (1) as above. The modifications required in the proof for the other algorithms are identified in the following:

ACA-1

Using the gradient estimate in (9), we may modify the proof in Theorem 1 with

$$H_i^{k_0}(n) = -\frac{(h_{\pi^2(n)}(i) + h_{\pi^2(n)}(i_0))}{\delta \Delta_i^{k_0}(n)} + r_i^{k_0}(n)$$

with $r_i^{k_0}(n) = o(1)$. A Taylor series expansion on $H_i^{k_0}(n)$ yields:

$$\begin{aligned} H_i^{k_0}(n) \quad = \quad & -\frac{(h_{\pi(n)}(i) + h_{\pi(n)}(i_0))}{\delta \Delta_i^{k_0}(n)} - \nabla_i^{k_0} h_{\pi(n)}(i) - \nabla_i^{k_0} h_{\pi(n)}(i_0) \\ & - \sum_{q=1, q \neq k_0}^{N} \frac{\Delta_i^q(n)}{\Delta_i^{k_0}(n)} \nabla_i^q \left(h_{\pi(n)}(i) + h_{\pi(n)}(i_0) \right) + O(\delta) \end{aligned} \tag{25}$$

which is similar to (21), except for the additional first term in the RHS above. Note that $d = 2^{\lceil \log_2 N+1 \rceil}$ here. A similar argument as the one using (23) holds. Further, using Corollary 2.6 of (Bhatnagar *et al.*, 2003), we obtain

$$\left| \sum_{l=m}^{m+d-1} \frac{c(l)}{c(m)} \frac{1}{\Delta_i^{k_0}(l)} \left(h_{\pi(l)}(i) + h_{\pi(l)}(i_0) \right) \right| \overset{m \to \infty}{\rightarrow} 0, \tag{26}$$

as well. This makes the algorithm equivalent to (24) with asymptotically diminishing error terms and the result follows. With the changes mentioned earlier, Theorem 1 can be seen to hold for RPAFA-1 as well.

DPAFA-1

Using the properties of $\bar{P}_i^{k_0}$, we see that $\pi_i^2(n) = (\pi_i(n) + \delta_i(n)\Delta_i(n))^T, \ \forall i \in S$, where the terms $\delta_i(n) = (\delta_i^j(n), \ 1 \leq j \leq N)^T$ are such that

$$0 \leq |\delta_i^j(n)| \leq \tilde{\delta} < \infty, \forall n \geq 0, i \in S, 1 \leq j \leq N,$$

for some $\tilde{\delta} > 0$ that is a function of the 'fineness' of the grid. We now perform Taylor series expansion about $\pi(n)$ as in (25) to obtain

$$\begin{aligned}\frac{h_{\pi^2(n)}(i) + h_{\pi^2(n)}(i_0)}{\delta\Delta_i^{k_0}(n)} &= \frac{h_{\pi(n)}(i) + h_{\pi(n)}(i_0)}{\delta\Delta_i^{k_0}(n)} + \frac{\delta_i^{k_0}(n)}{\delta}\left(\nabla_i^{k_0} h_{\pi(n)}(i) + \nabla_i^{k_0} h_{\pi(n)}(i_0)\right) \\ &+ \sum_{q=1, q\neq k_0}^{N} \frac{\delta_i^q(n)}{\delta}\frac{\Delta_i^q(n)}{\Delta_i^{k_0}(n)} \nabla_i^{k_0}\left(h_{\pi(n)}(i) + h_{\pi(n)}(i_0)\right) + O(\delta) \qquad (27)\end{aligned}$$

We use the limit in (26) for the first term in the RHS above. For $k \neq k_0$, note that

$$\left|\sum_{l=m}^{m+d-1} \frac{\delta_i^k(l)}{\delta}\frac{c(l)}{c(m)}\frac{\Delta_i^k(l)}{\Delta_i^{k_0}(l)} \nabla_i^k \left(h_{\pi(l)}(i) + h_{\pi(l)}(i_0)\right)\right| \stackrel{m\to\infty,\, \delta_i^k(l)\to\delta}{\longrightarrow} 0. \qquad (28)$$

In the above, the fact that $\frac{c(l)}{c(m)} \to 1$ as $m \to \infty, \ l \in \{m, m+1, ..., m+d-1\}$ is used. Note that as the partition is made 'finer', $\delta_i^k(l) \to \delta$, the above reduces to the case of (22). The algorithm will then converge to a local minimum in the limit as $\delta \to 0$. For a fixed $\delta > 0$, the algorithm can be seen to converge either to the local minimum or a point in its neighborhood. A similar analysis holds for DPAFA-2.

8 A Variant of the TD(0) Algorithm

We propose a variant of the two-timescale TD(0) algorithm of (Konda & Tsitsiklis, 2003). Here we approximate the differential cost $h(\cdot)$ using a linear approximation architecture (cf.

(Bertsekas & Tsitsiklis, 1996) and (Tsitsiklis & Van Roy, 1997)) as $\tilde{h}(i) = \phi(i)^T \nu$, where ν, $\phi(i) \in \mathcal{R}^K$ for some $K \geq 1$. Here, $\phi(i)$ is called the feature vector of state i, (see (Tsitsiklis & Van Roy, 1997), (Tsitsiklis & Van Roy, 1999) and (Konda & Tsitsiklis, 2003)). We have the following recursion in place of (5): $\forall i \in S$,

$$\begin{aligned}\nu^r_{n\tilde{L}+ms+i+1} &= \nu^r_{n\tilde{L}+ms+i} + b(n)\left(K(i, \pi^r_i(n), \eta^r_{nL+m}(i, \xi^r_{nL+m}(i, \pi^r_i(n))))\right.\\ &\quad \left. + \tilde{h}^r_{n\tilde{L}+ms+i}(\eta^r_{nL+m}(i, \pi^r_i(n))) - \tilde{h}^r_{n\tilde{L}+ms+i}(i_0) - \tilde{h}^r_{n\tilde{L}+ms+i}(i)\right)\phi(i) \qquad (29)\end{aligned}$$

for $1 \leq m \leq L$ and $\tilde{L} = Ls$. The $n-$th policy update is performed using $\tilde{h}^r_{n\tilde{L}}(i) = \phi(i)^T \nu^r_{n\tilde{L}}$, $\forall i \in S$ in place of $h^r_{nL}(i)$ in recursion (4) for all algorithms.

Note that in (29), the states $i = 1, 2, ..., s$ are thus sampled cyclically, one after another. Thus, s updations of the vector ν^r would be performed before the algorithm visits state i again, and sL updations would be performed between two consecutive updates of the policy $\pi(n)$ using (4). Such a method corresponds to the implementation of the analogous 'critic' recursion (5) in the previous six algorithms. However, the recursion in (29) requires less memory due to the use of a $K-$dimensional coefficient vector ν^r in place of a size s vector h^r in (5), where typically $K \ll s$.

Note that in visiting states cyclically (in a given lexicographic order), we are sampling the state space with a distribution that is not the same as the stationary distribution imposed by policy $\pi^r(n)$ in (29), as is asserted by (Bertsekas & Tsitsiklis, 1996, Section 6.3) and (Tsitsiklis & Van Roy, 1997, Section 9). Nevertheless, the method is good in practice as borne out by the simulation results.

We give an outline of the theoretical impediments in proving convergence for (29). Consider an $s \times K$ matrix Φ where ϕ^T_i, $\forall i \in S$ are the rows. Just as in (Tsitsiklis & Van Roy, 1997) and (Tsitsiklis & Van Roy, 1999), Φ is assumed to have full column rank, a requirement implying that none of the features are correlated. Using Lemma 4 of (Tsitsiklis & Van Roy, 1999) with appropriate modifications, we have to establish that the (vector) linear ODE

$$\dot{\nu}^r(t) = \frac{1}{s}\Phi^T A^r \Phi \nu^r(t) + \frac{1}{s}\Phi^T B^r \tag{30}$$

is asymptotically stable, where A^r and B^r are $s \times s$ and $s \times 1$ matrices, respectively, used in the 'exact' critic ODE (cf. (16))

$$\dot{h}^r(t) = A^r h^r(t) + B^r \tag{31}$$

Here, using (13), we see that $A^r = P_{\pi^r(n)} - I - I_0$ where $I_0(k, i_0) = 1$ for $1 \leq k \leq s$ and 0 elsewhere whereas $B_i^r = \sum_{j=1}^{s} p(i, \pi_i^r, j)K(i, \pi_i^r, j),\ \forall i \in S$. Using (Perko, 1998, Theorem 2, pp.56), A^r will have negative real parts in all its eigen values by virtue of the asymptotic stability of (31) (see Lemma 6.3 of (Konda & Borkar, 1999) for a proof of asymptotic stability of (31)). However, A^r need not be negative definite in general, which would have been a *sufficient* condition to render (30) asymptotically stable, since then $\Phi^T A^r \Phi$ would also be negative definite. As a result, (29) cannot be guaranteed to track an asymptotically stable ODE for all policies $\pi^r(n)$. However, as already mentioned, the above algorithm shows good numerical performance on our setting.

9 Numerical Experiments

9.1 Setting and Parameters

We consider a continuous time queuing model for flow control in communication networks as in (Bhatnagar *et al.*, 2001a). A single bottleneck node is fed with two arrival streams, one an uncontrolled Poisson stream and the other a controlled Poisson process. Service times are assumed i.i.d. with exponential distribution. We assume that the node has a buffer size

$B < \infty$. Given a constant $T > 0$, we assume that the continuous-time queue length process $\{q_t, t > 0\}$ at the node is observed every T instants and on the basis of this information the controlled source tunes the rate at which it sends packets so as to minimize a certain cost. Suppose q_n denotes the queue length observed at time $nT, n \geq 0$. The controlled source thus sends packets according to a Poisson process with rate $\lambda_c(q_n)$ during the time interval $[nT, (n+1)T)$. The rate is then changed to $\lambda_c(q_{n+1})$ at instant $(n+1)T$ upon observation of state q_{n+1} - we assume for simplicity that there is no feedback delay. The aim then is to find a stationary optimal rate allocation policy that minimizes the associated infinite horizon average cost. We compare performance of the algorithms in this setting.

Initially, we choose $B = 50$ and take the compact action set $U(i), \forall i \in S$ (where $\lambda_c(i) \in U(i)$) to be the interval $[0.05, 4.5]$ for the algorithms ACA-r. In the finite action setting, the algorithms RPAFA-r compute the optimal stationary randomized policy over the discrete action set, $U(i),\ \forall i \in S$, consisting of evenly spaced values within the action set for ACA-r. Thus, for each state i, $U(i)$ is the set $\{0.05, 1.1625, 2.275, 3.3875, 4.5\}$ and the probability of applying action $u_i^k, 0 \leq k \leq 4$ is obtained from the vector $(\pi_i^0, \pi_i^1, ..., \pi_i^4)$, where $\pi_i^k \geq 0, \forall k$ and $\sum_{k=0}^{4} \pi_i^k = 1$. For DPAFA-$r$, all deterministic policies π, when applied, belong to a similar discrete action set as RPAFA-r, i.e.,

$$\bar{P}(\pi_i) = \lambda_c(i) \in \{0.05, 1.1625, 2.275, 3.3875, 4.5\},\ \forall i \in S = \{0, 1, ..., B\}.$$

The uncontrolled traffic rate λ_u is chosen as 0.2 and the service rate μ for incoming packets is set to 2.0. For a system operating under a stationary policy π, we use the cost function

$$K(q_n, \pi_{q_n}, q_{n+1}) = |q_{n+1} - \frac{B}{2}|.$$

Note that while the source rate chosen, π_{q_n}, does not directly enter into the cost function above, it has an impact on the queue length q_{n+1} observed T seconds later, which in turn affects the cost. A cost function of this type is useful in cases where the goal is to simultaneously maximize throughput and minimize the delay in the system.

The value of δ needed in (4) is set to 0.1. We arbitrarily set the initial policy as $(\pi_i(0) = 2.275, \forall i \in S)^T$ for ACA-r and DPAFA-r and the initial stationary randomized policy to the uniform distribution $(\pi_i^k(0) = 0.2,\ 0 \leq k \leq 4)^T$ for each state $i \in S$ for RPAFA-r. The value of L in all algorithms is taken to be 100 and the reference state (in the algorithms) i_0 is 25.

In order to test for convergence of the ACA-r and DPAFA-r algorithms, we measured a quantity err_n at policy update n in the following manner:

$$err_n = \max_{i \in S, 1 \leq k \leq 50} |\pi_i(n) - \pi_i(n-k)|$$

where $\pi(n)$ is the policy obtained from the n-th update. Similarly, convergence for the RPAFA-r algorithms was measured using

$$err_n = \max_{i \in S, 1 \leq k \leq 50} \sqrt{\sum_{j=0}^{4} (\pi_i^j(n) - \pi_i^j(n-k))^2}.$$

The step-size sequences $\{c(n)\}$ and $\{b(n)\}$ needed in (4)-(5) were chosen as

$$c(0) = b(0) = 1, c(n) = \frac{1}{n}, b(n) = \frac{1}{n^{2/3}}, \forall n \geq 1.$$

9.2 Simulation Results

We stop algorithms ACA-r and DPAFA-r when $err_n \leq 0.1$. This is achieved within $n = 2.2 \times 10^3$ policy updates for both ACA-r and DPAFA-r. Note that for the given settings, maximum possible err_n is 4.45. In Figure 1, we plot the converged rates for $T = 5, 10$ and 15 respectively for ACA-2. We observe that for given T, the source rates are inversely related to the queue length values since the cost function imposes a high penalty for states away from $\frac{B}{2}$. Further, the difference between the highest and lowest rates decreases as T increases since for lower values of T, the controller has better control over the system dynamics than when T is large.

In Figure 2, we show the convergence plot for the source rates corresponding to states $0, 15, 25, 35$ and 50 for ACA-2 with $T = 5$. In Figure 3, we plot the source rates with highest probability of selection as given by the converged randomized policy computed by RPAFA-2 for $T = 5, 10$ and 15 respectively. The value of $err_n \leq 0.1$ is achieved within 1.3×10^4 policy updates for both RPAFA-1 and RPAFA-2. In Figure 4, we show the convergence plot for the probability of selection of source rates corresponding to state 25 for RPAFA-2 with $T = 5$. In Figure 5 we plot the converged rates for $T = 5, 10$ and 15 respectively for DPAFA-2. Similar behaviour was observed in the case of algorithms that use one simulation.

We also compute the optimal policy for the case where there are multiple sources feeding the bottleneck node. In a simulation where three controlled sources feed the node instead of one, Figure 6 depicts the optimal policy of one among the sources, computed using ACA-2. The action sets $U(i)$ for all three sources are taken as $[0.015, 1.8]$. Figure 7 plots the net arrival rate (from all three sources) into the bottleneck node. This rate, being the sum of contributions from each of the sources, takes values in $[0.045, 5.4]$. The individual source rates show here a trend that is similar to the single-source case, but the policy graph obtained is not as smooth. This is because the algorithm cannot detect contributions to the net source rate from individual sources. However, as expected, the plot for the net source rate (sum of the source rates from all three sources in an interval) in Figure 7, is similar to that of the single-source case (Figure 1).

The optimal differential-cost function $h^*(i)$, obtained using the converged policies, is shown in Figures 8, 9 and 10 using algorithms ACA-2, RPAFA-2 and DPAFA-2 respectively, for $T = 5, 10$ and 15 in each. The function is computed as follows: After convergence of the policy vector in (4), the inner recursion (5) is run with $b(n) = \frac{1}{n}$, for 2×10^4 iterations using source rates obtained from the converged policy π^* found earlier. The h^* curves for RPAFA-2 and DPAFA-2 have higher values than ACA-2 because of the significantly broader range of actions using which the latter algorithm operates.

The long run average costs computed using the converged policies of all three algorithm types, for three values of T, are indicated in Table 1. Note that the average costs for the discrete action setting are higher than those for the compact action case, again due to the wider range of actions in the compact action case. Further, among the algorithms with discrete action sets, DPAFA-r algorithms have lower average costs compared to RPAFA-r.

Next we compare the performance of algorithms DPAFA-r and RPAFA-r with the average-cost actor-critic policy iteration algorithms given by (Konda & Borkar, 1999). In Table 2, AC-4, AC-5, and AC-6 stand for the relevant algorithms 4, 5 and 6 respectively, of (Konda & Borkar, 1999). The critic recursion employed for all the (above) three algorithms is (32) which resembles (5) except that its estimates $h(n)$ approximate the differential-cost function for the policy estimate $\pi(n)$ rather than the perturbed policies $\pi^r(n)$ and do not perform the additional $L-$step averaging. Also, the $\pi(n)$ are randomized policies in AC-4, AC-5, and AC-6, respectively. The actor recursions vary for each algorithm, they are (33) for AC-4, (34) for AC-5 and (35) for AC-6, respectively. In these recursions, e_i^l is an N_i-dimensional unit vector with 1 in the $l-$th place and the $S-$valued random variable $\bar{\eta}_n(i, u_i^l)$ represents the state to which the system transits when action u_i^l is taken in state i. Note that algorithms AC-5 and AC-6 require apriori choices of certain parameters used in the policy updates. In particular, AC-5 (cf. (34)) needs a zero-mean noise required to push the estimates away from the boundary of the simplex - ψ_i^l - which in this case is chosen to be the uniform distribution $U(-0.5, 0.5)$. Simulation results for AC-5 under a somewhat different noise condition have been presented by (Borkar & Konda, 2004) for the discounted cost case. Similarly, AC-6 operates by sampling the discrete action set using a Boltzmann distribution scheme in which the inverse-temperature weights, $\pi_i^l(n)$, represent the policy. In this experiment, these weights are projected to within $[-10.0, 10.0]$ via the operator P.

$$\begin{aligned} h_i(n+1) &= (1-b(n))h_i(n) + b(n)(K(i,\xi_n(i,\pi_i(n)),\eta_n(i,\xi_n(i,\pi_i(n)))) \\ &\quad - h_{i_0}(n) + h_{\eta_n(i,\xi_n(i,\pi_i(n)))}(n)) \end{aligned} \tag{32}$$

$$\begin{aligned} \hat{\pi}_i(n+1) &= \hat{P}(\hat{\pi}_i(n) + c(n)\sum_{l=1}^{N_i}(K(i,u_i^0,\bar{\eta}_n(i,u_i^0)) - K(i,u_i^l,\bar{\eta}_n(i,u_i^l)) \\ &\quad + h_{\bar{\eta}_n(i,u_i^0)}(nL) - h_{\bar{\eta}_n(i,u_i^l)}(nL))e_i^l) \end{aligned} \tag{33}$$

$$\begin{aligned} \hat{\pi}_i(n+1) &= \hat{P}(\hat{\pi}_i(n) + c(n)\sum_{l=1}^{N_i}\{(h_{i_0}(nL) + h_i(nL) - K(i,u_i^l,\bar{\eta}_n(i,u_i^l)) \\ &\quad - h_{\bar{\eta}_n(i,u_i^l)}(nL))\hat{\pi}_i^l + \psi_i^l(n)\}e_i^l) \end{aligned} \tag{34}$$

$$\pi_i(n+1) = P(\sum_{l=1}^{N_i}\pi_i^l(n) + c(n)(h_{i_0}(nL) + h_i(nL) - K(i,u_i^l,\bar{\eta}_n(i,u_i^l)) - h_{\bar{\eta}_n(i,u_i^l)}(nL))e_i^l) \tag{35}$$

We compute certain steady-state performance metrics for comparison. We denote $P([\frac{B}{2}-1, \frac{B}{2}+1])$ as the stationary probability of the queue being in the states $\{\frac{B}{2}-1, \frac{B}{2}, \frac{B}{2}+1\}$. Also, σ^2 indicates the variance of the queue size about the mean queue length $E[q_n]$, given that the system is operating under the converged policy. The quantity err_n is also tabulated. Note that the maximum possible err_n in the discrete action setting is $\sqrt{|1| + |-1|} = 1.414$. The varying number of policy updates for each algorithm is justified as follows: we run AC-r, $r = 4, 5, 6$ for $L \times 10^4 = 10^6$ updates, so as to match RPAFA-1 in the number of function evaluations. For the same reason, RPAFA-2 (resp. DPAFA-2) is run for half the number of policy updates as RPAFA-1 (resp. DPAFA-1), resulting in RPAFA-1 and RPAFA-2 (resp.

DPAFA-1 and DPAFA-2) taking roughly the same time to perform the prescribed number of policy updates.

It is observed that both DPAFA-r and RPAFA-r display lower average cost (λ^*) than the three AC-r algorithms, although the value of err_n is lower (in fact, $\leq 10^{-2}$) for the latter algorithms. As borne out in the earlier results (cf. Table 1), DPAFA-r performs better than RPAFA-r. The computation times shown are measured on a Pentium III computer. Though comparable in the number of function evaluations and simulation of the critic recursion (32), the three AC-r algorithms take more than an order of magnitude time than all our algorithms. This difference is contributed to by the additional simulation effort needed in the policy update step of the AC-r algorithms, i.e., in simulating the $\bar{\eta}_n(\cdot,\cdot)$ terms in each one of (33), (34) and (35), respectively. Note that the σ^2 values obtained using AC-4 and AC-5 are comparable to the values obtained using our algorithms. However, this is mainly because the mean queue lengths in our algorithms are higher as these are closer to their target means than those of the AC-r algorithms. Inspite of the above, AC-6 has a significantly higher σ^2 (as compared to our algorithms). Moreover, our algorithms show significantly better performance in terms of all other performance metrics described.

Next, we increase the size of the state space and use the $TD(0)$ variant of ACA-2. Table 3 indicates, for the $B = 500$ case, the average costs obtained after 500 updates of the policy vector, using a polynomial function approximation of order $K-1$ to the differential-cost i.e., $\phi_k(i) = \left(\frac{i-i_0}{i_0}\right)^k$, $0 \leq k \leq K-1$. The times required for each algorithm are also indicated. The computation times are seen to be roughly linear in K but in general higher than 'exact' ACA-2 since the function-approximation procedure is computationally intensive. In particular, every step of the critic's computation (cf. (29)) involves a size-K vector addition and dot-product on top of the simulation effort required in simulating transitions out of each state. The advantage of the approximation method, however, lies in the diminished storage requirement. Since the policy vector is also of size B, differential-cost function approximation operates using roughly one-third of the memory space required for the 'exact' algorithms.

Using $K = 4$ and $B = 2000$, 5000 and $10,000$, respectively, the average cost obtained using polynomial approximation of differential-cost in ACA-2 is indicated in Table 4. This is compared with the cost obtained using 'exact' ACA-2. The average cost in these cases is found to be higher, since the policy obtained uses only approximations of the h function for gradient-finding. The number of iterations and time required for convergence of all the algorithms, when implemented using the approximation method with $K = 4$ and $B = 50$, is shown in Table 5.

10 Conclusions

In this paper we developed two timescale gradient search based actor-critic algorithms for solving infinite horizon MDPs with finite state space under the average cost criterion. The action spaces considered were both compact and discrete action sets, respectively. All our algorithms update the actor recursion on the slower timescale using appropriate SPSA based policy gradient estimates. On the faster timescale, the differential cost function values for perturbed policies were estimated with an additional L-step averaging. The algorithms were theoretically shown to converge to a locally optimal policy. A memory efficient implementation of the critic recursion using off-line TD(0) learning was also discussed. We showed numerical experiments using a continuous time queueing model for rate based flow control and compared the performance of the algorithms for the discrete action case with those of (Konda & Borkar, 1999). We observed that our algorithms show almost an order of magnitude better performance than the algorithms of (Konda & Borkar, 1999).

Using simulation-based algorithms with similar coupled stochastic approximation as (4)-(5) it would be possible to implement the value iteration algorithm of DP (cf. (Bertsekas, 1976, section 8.2)) as well. The fast convergence of the proposed algorithms in this work could be theoretically established using rate-of-convergence studies of SPSA-based gradient estimates vis-a-vis actor-critic algorithms that use other policy gradient methods as of (Konda & Borkar, 1999). In this context, further improvement in performance of the proposed algorithms can be expected by using efficient simulation-based higher order SPSA algorithms that estimate the Hessian in addition to the gradient along the lines of (Bhatnagar, 2005). Also, in (29) a strict lexicographic sampling is not necessary. Moreover, the order of states need not be stored by the (distributed) critic, that would result in a promising implementation more in line with the *bounded rationality* assumption of agents performing policy updates in an asynchronous manner.

References

Bertsekas, D. 1976. *Dynamic Programming and Stochastic Control*. New York: Academic Press.

Bertsekas, D.P., & Tsitsiklis, J.N. 1996. *Neuro–Dynamic Programming*. Belmont MA: Athena Scientific.

Bhatnagar, S. 2005. Adaptive multivariate three–timescale stochastic approximation algorithms for simulation based optimization. *ACM Transactions on Modeling and Computer Simulation*, **15**(1), 74–107.

Bhatnagar, S., & Abdulla, M.S. 2005. Reinforcement Learning Based Algorithms for Finite Horizon Markov Decision Processes. *Submitted.*

Bhatnagar, S., & Kumar, S. 2004. A Simultaneous Perturbation Stochastic Approximation–Based Actor–Critic Algorithm for Markov Decision Processes. *IEEE Transactions on Automatic Control*, **49**(4), 592–598.

Bhatnagar, S., Fu, M.C., Marcus, S.I., & Fard, P.J. 2001a. Optimal structured feedback policies for ABR flow control using two–timescale SPSA. *IEEE/ACM Transactions on Networking*, **9**(4), 479–491.

Bhatnagar, S., Fu, M.C., Marcus, S.I., & Bhatnagar, S. 2001b. Two timescale algorithms for simulation optimization of hidden Markov models. *IIE Transactions (Pritsker special issue on simulation)*, **3**, 245–258.

Bhatnagar, S., Fu, M.C., Marcus, S.I., & Wang, I-J. 2003. Two–timescale simultaneous perturbation stochastic approximation using deterministic perturbation sequences. *ACM Transactions on Modeling and Computer Simulation*, **13**(4), 180–209.

Borkar, V.S., & Konda, V.R. 2004. Actor–critic algorithm as multi–time scale stochastic approximation. *Sadhana*, **22**, 525–543.

Borkar, V.S., & Meyn, S.P. 2000. The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, **38**(2), 447–469.

Gerencser, L., Hill, S.D., & Vago, Z. 1999. Optimization over discrete sets via SPSA. *Pages 1791–1794 of: Proceedings of the 38th IEEE Conference on Decision and Control–CDC99, Phoenix, Arizona, USA*. Piscataway, NJ: IEEE Press.

Konda, V.R., & Borkar, V.S. 1999. Actor–Critic Type Learning Algorithms for Markov Decision Processes. *SIAM Journal on Control and Optimization*, **38**(1), 94–123.

Konda, V.R., & Tsitsiklis, J.N. 2003. Actor–Critic Algorithms. *SIAM Journal on Control and Optimization*, **42**(4), 1143–1166.

Kushner, H.J., & Clark, D.S. 1978. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. New York: Springer Verlag.

Perko, L. 1998. *Differential Equations and Dynamical Systems, 2nd ed., Texts in Applied Mathematics, Vol.7*. New York: Springer Verlag.

Puterman, M.L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley.

Spall, J.C. 1992. Multivariate Stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, **37**(1), 332–341.

Spall, J.C. 1997. A One–Measurement Form of Simultaneous Perturbation Stochastic Approximation. *Automatica*, **33**(1), 109–112.

Tsitsiklis, J.N., & Van Roy, B. 1997. An Analysis of Temporal–Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, **42**(5), 674–690.

Tsitsiklis, J.N., & Van Roy, B. 1999. Average Cost Temporal–Difference Learning. *Automatica*, **35**(11), 1799–1808.

Van Roy, B. 2001. Handbook of Markov Decision Processes: Methods and Applications. *In:* Feinberg, E., & Shwartz, A. (eds), *Neuro–Dynamic Programming: Overview and Recent Trends*. Dordrecht: Kluwer International.

11 Tables

Algorithm	λ^*(T=5s)	λ^*(T=10s)	λ^*(T=15s)
ACA-2	3.98	5.08	6.18
ACA-1	4.0	5.09	6.17
DPAFA-2	4.58	5.95	7.38
DPAFA-1	4.88	5.96	7.38
RPAFA-2	5.68	6.29	9.48
RPAFA-1	5.62	7.17	9.03

Table 1: Average Cost λ^* as computed by the proposed algorithms

Metric/Algorithm	DPAFA-2	DPAFA-1	RPAFA-2	RPAFA-1	AC4	AC5	AC6
# policy updates	5×10^3	10^4	5×10^3	10^4	10^6	10^6	10^6
λ^*	4.87	4.88	5.69	5.57	11.29	9.90	11.72
$P([\frac{B}{2}-1, \frac{B}{2}+1])$	0.22	0.22	.18	.18	.04	.07	.06
$E[q_n]$	23.2	23.2	22.1	23.0	13.7	15.0	14.6
σ^2	41.1	41.42	46.55	42.45	40.54	40.50	73.02
Time in seconds	376	402	648	653	15932	12093	16111
err	.001	0.025	.01	.15	$\leq 10^{-2}$	$\leq 10^{-2}$	$\leq 10^{-2}$

Table 2: Comparison of DPAFA-r and RPAFA-r with Actor-Critic Policy Iteration

Metric/Polynomial order	Exact	3	4	5	6	7
J^*	5.68	16.0	14.7	13.8	18.4	18.4
Time in seconds	176	175	190	204	211	216

Table 3: J^* using Differential-Cost Approximation for $B = 500$

B	λ^*(Look-up Table)	$\lambda^*(K = 4)$
2000	6.3	8.7
5000	7.5	29.1
10000	8.3	60.1

Table 4: λ^* using h-approximation for large B and $K = 4$

Algorithm	# Iterations	Time in seconds	λ^*
RPAFA-1	14500	586	10.7
RPAFA-2	1620	135	10.6
DPAFA-1	9550	406	9.7
DPAFA-2	750	70	6.9
ACA-1	9300	384	8.14
ACA-2	700	79	5.4

Table 5: Comparison of Convergence using h-approximation for $B = 50$ and $K = 4$

12 Figures

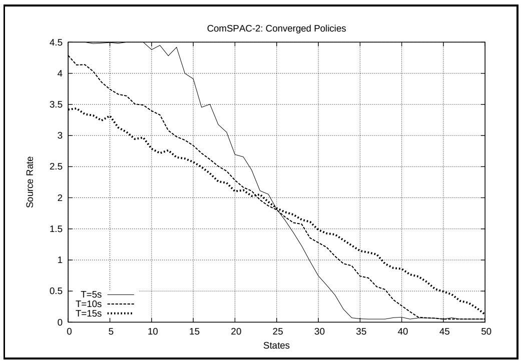

Figure 1: Converged Policy computed by ACA-2

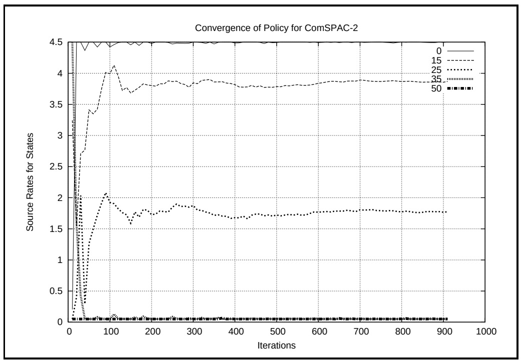

Figure 2: Convergence Behaviour for ACA-2

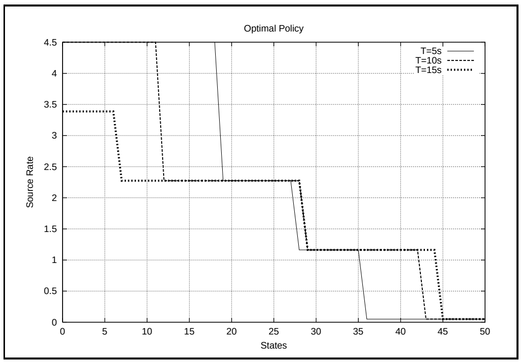

Figure 3: Randomized Stationary Policy computed using RPAFA-2

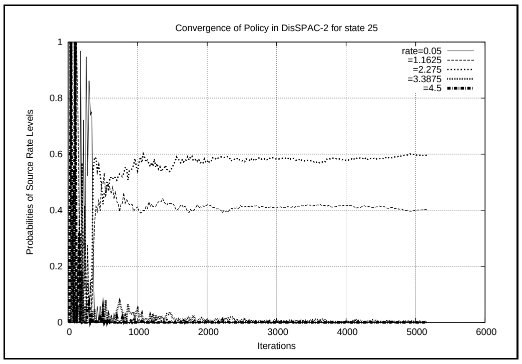

Figure 4: Convergence Behaviour for RPAFA-2

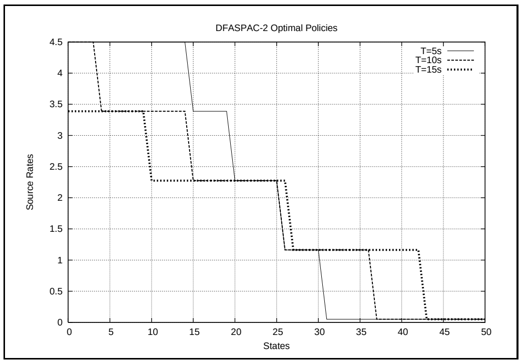

Figure 5: Policy computed using DPAFA-2

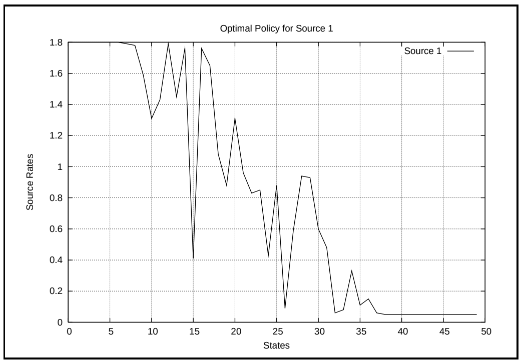

Figure 6: Optimal Policy for Source 1

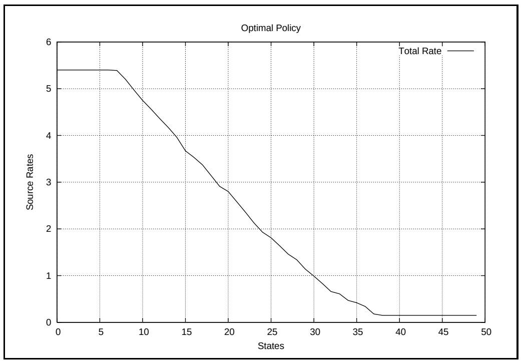

Figure 7: Optimal Policy - Sum of Source Rates

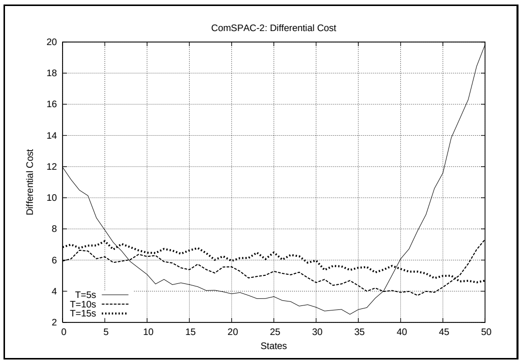

Figure 8: Differential-Cost Function computed using ACA-2

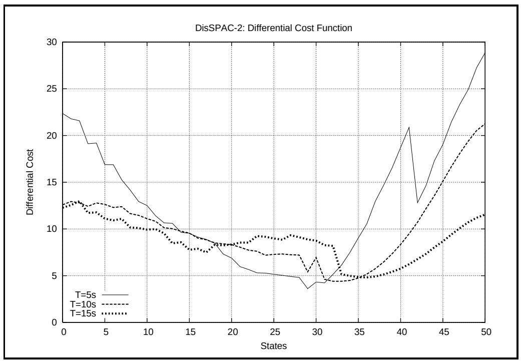

Figure 9: Differential Cost Function computed using RPAFA-2

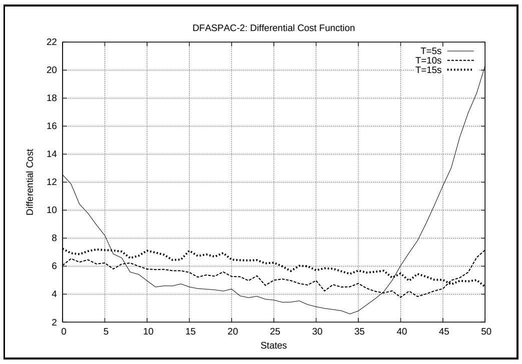

Figure 10: Differential Cost Function computed using DPAFA-2