Simulation-Based Optimization Algorithms for Finite-Horizon Markov Decision Processes

Shalabh Bhatnagar
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012, India
shalabh@csa.iisc.ernet.in

Mohammed Shahid Abdulla
General Motors
India Science Lab
Bangalore

We develop four simulation-based algorithms for finite-horizon Markov decision processes. Two of these algorithms are developed for finite state and compact action spaces while the other two are for finite state and finite action spaces. Of the former two, one algorithm uses a linear parameterization for the policy, resulting in reduced memory complexity. Convergence analysis is briefly sketched and illustrative numerical experiments with the four algorithms are shown for a problem of flow control in communication networks.

Keywords: Finite-horizon Markov decision processes, simulation-based algorithms, two-timescale stochastic approximation, function approximation, actor-critic algorithms, normalized Hadamard matrices

1. Introduction

Markov decision processes (MDPs) are a general framework for solving stochastic control problems [1, 2]. Value iteration and policy iteration are two of the classical approaches for solving the Bellman equation for optimality. While value iteration proceeds by recursively iterating over value function estimates starting from a given such estimate, policy iteration functions by iterating over policies and involves updates in two nested loops. The inner loop estimates the value function for a given policy update while the outer loop updates the policy. The former estimates are obtained as solutions to linear systems of equations that are most often solved using value iteration type recursions (particularly when the numbers of states and actions are large).

In the above classical approaches, however, one requires complete knowledge of the system model via transition probabilities. This information is usually unavailable in the case of most real-life systems. Moreover, the computational requirements for solving the Bellman equation typically become prohibitive in the presence of large state spaces (the curse of dimensionality). Motivated by these considerations, research on simulation-based methods that largely go under the rubric of reinforcement learning [3] or neuro-dynamic programming [4, 5] has gathered momentum in recent times.

The main idea behind these schemes is to simulate transitions instead of directly computing transition probabilities that may be hard to obtain and, in scenarios where the numbers of states and actions are large, use parametric representations of the cost-to-go function and/or policies. Most reinforcement learning schemes are based on value iteration or policy iteration. For instance, temporal difference learning [4, 6] and Q-learning [4, 7] algorithms are based on the value iteration method. Similarly, actor-critic algorithms that involve both cost-to-go and policy parameterizations [8–11] are based on the policy iteration method.

The inner loop of the policy iteration algorithm for any given policy update may typically take a long time to converge, particularly when the numbers of states and actions are large. An actor-critic algorithm based on two-timescale stochastic approximation was proposed [9]. The algorithm proceeds using two coupled recursions driven

SIMULATION, Vol. 84, Issue 12, December 2008 577–600

DOI: 10.1177/0037549708098120

by different step-size schedules or timescales. The policy evaluation step of policy iteration is performed on the faster timescale while the policy improvement step is carried out along the slower one. Thus, one need not wait for convergence of the inner-loop procedure before an outer-loop update as with regular policy iteration. Instead, both recursions are executed in tandem, one after the other, and the optimal policy-value function pair is obtained upon convergence of the algorithm.

The technique of two-timescale stochastic approximation is further used [10], where parameterizations of both value function and policy are considered. A temporal difference type update is performed along the faster timescale while gradient search is performed on the slower one. Most reinforcement learning based algorithms in the literature have been developed for the case of finite state and finite action sets. An actor-critic, two-timescale algorithm has been proposed for finite state and compact (non-discrete) action sets [11]. The algorithm performs updates in the space of stationary deterministic and not randomized policies as with algorithms for finite action sets. The faster timescale recursion is similar to the corresponding recursion in [9] except that an additional averaging over L epochs (for some fixed $L > 1$) is proposed for enhanced convergence behavior [11]. Also, on the slower timescale, a gradient search using simultaneous perturbation stochastic approximation (SPSA) gradient estimates [12] is performed and convergence to a locally optimal policy is shown.

SPSA has been used in a wide range of settings in applications involving parameter optimization, and is found to be computationally efficient. Using SPSA, only two measurements of the loss function are required at each update epoch for parameters of any dimension, by simultaneously perturbing randomly all parameter components. A one-measurement form of SPSA has also been proposed [13]. This, however, is not seen to show good performance because of certain 'additional' bias terms in the gradient estimate therein that contribute significantly to the (gradient) estimation error.

It was observed [14] in a setting of simulation-based parameter optimization that the use of certain deterministic perturbation sequences improves performance significantly in the case of one-measurement SPSA. The idea is to identify a 'base' matrix of perturbation vectors whose columns satisfy the desired properties associated with the perturbation sequences. This matrix is in fact obtained from a normalized Hadamard matrix and typically has a small dimension. The perturbation vectors for the gradient estimates are then obtained by cycling through the rows of this base matrix.

Reinforcement learning algorithms have generally been developed and studied in the infinite-horizon MDP framework [3–5] under the discounted cost or the long-run average cost criteria. For instance, TD learning, Q-learning and actor-critic algorithms have been developed in the infinite-horizon framework i.e. when the number of stages over which decisions are made is infinite. However, in most real-life scenarios, finite-horizon decision problems (i.e. those with a finite number of stages) assume serious significance. For instance, in the design of a manufacturing fab, one requires planning over a finite decision horizon [15, 16]. In communication or vehicular networks, flow and congestion control problems should realistically be studied only as finite-horizon decision-making problems, since the amount of time required in clearing congestion and restoring normal traffic flows in the network is of prime interest.

Unlike infinite-horizon problems, the optimal policies in finite-horizon problems may in general depend on time and thus need not be stationary. In scenarios where model information (i.e. transition probabilities of the system) is not known, the dynamic programming algorithm cannot be directly used to obtain optimal policies. Moreover, the amount of computational effort increases significantly when the number of stages and the numbers of states and admissible actions in each state become large. Reinforcement learning could be applied in such scenarios.

Most of the work on developing computationally efficient algorithms for finite-horizon problems, however, assumes that model information is known. For instance, the problem of solving a finite-horizon MDP under partial observations is formulated as a nonlinear programming problem [17] and a gradient search based solution methodology is developed. For a similar problem, a solution procedure based on genetic algorithms and mixed integer programming is presented [18]. A hierarchical structure using state aggregation is proposed for solving finite-horizon problems [19].

We develop two-timescale simulation-based actor-critic algorithms for finding optimal policies in finite-horizon MDPs (F-H MDPs). We assume that information on transition probabilities (or model) of the system is not known, however, transitions can be simulated. We consider two different settings: finite state and compact (non-discrete) action sets and also finite state and finite action sets. Three variants of the Q-learning algorithm for the finite-horizon problem are developed [20] assuming lack of model information. However, the F-H MDP problem is embedded as an infinite-horizon MDP either by adding an absorbing state at the terminal stage (or the end of horizon) or a modified MDP is obtained by restarting the process by selecting one of the states at the initial (first) stage of the MDP according to the uniform distribution, once the terminal stage is hit.

Our approach is fundamentally different from that of [20]. In particular, we do not embed the F-H MDP into an infinite horizon. The solution procedure that one obtains using the approach in [20] is at best only approximate, whereas our solution procedure does not involve such approximations.

The second algorithm that we propose produces an optimum subject to the constraints of linear parameterization of the policy. The duration T of the decision horizon

remains fixed in our case. With the exception of the second algorithm that we propose, the aim at each stage i, $i = 0, 1, \ldots, T-1$ is to find the optimal decision rule by taking into account the single-stage costs and the cost-to-go from the subsequent stage. Assuming that the action in each (state, stage) pair is a scalar and that S is the state space over all stages i, our algorithms update all components of the $|S| \times T$-size policy vector π, or a proxy thereof, at each update step. We show that our algorithms converge in the limit to the optimal T-stage finite-horizon policy.

The faster timescale recursions in our algorithms are designed to solve the system of T linear equations for a given policy, as given by the dynamic programming algorithm, while the slower timescale recursions perform gradient search in the space of policies. We will have occasion to use two variants of SPSA: the classical two-simulation form [12] and the one-simulation version that uses Hadamard matrix based perturbations [14].

Both the proposed algorithms for MDPs with compact action sets perform a search in the space of deterministic policies. Of these, the second algorithm addresses the key concern that the size of *both* the policy and cost-to-go table require a storage of size at least $|S| \times T$. This is an overhead compared to infinite-horizon MDPs where only size $|S|$ tables are needed, indicating that the memory requirements in F-H MDPs assume severity if T is large. In turn, the second algorithm has a computational complexity proportional to the decision horizon T. A lack of stationary distributions in F-H MDPs also manifests itself in the absence of interpretable bounds on the cost-to-go approximation – a distinguishing feature of the previous cost-to-go approximation work [6, 21].

Among applications of F-H MDPs with an emphasis on cost-to-go approximation are problems in Online Mechanism Design [22, 23]. In a spirit similar to [6] and related work, the second algorithm uses feature vectors $\phi_r(i), \forall i \in S$ and $0 \leq r \leq T-1$ where $\phi_r(i) \in \mathcal{R}^K$, $K \ll |S|$. Accordingly, this algorithm searches in a subset of the space of deterministic policies, namely policies whose rth stage actions lie in the span of the feature vector matrix $\Phi_r \stackrel{\triangle}{=} (\phi_r(i), 1 \leq i \leq |S|)^T$.

Further, one of the algorithms for MDPs with finite action sets performs a search in the space of randomized policies while the other does so in the space of deterministic policies. We provide a convergence analysis for the algorithms of compact action sets and briefly identify the changes in analysis required for the other two algorithms.

1.1 Application of Finite-Horizon MDPs

Problems of control in communication networks, such as those of routing, resource allocation, admission control, flow and congestion control, are most naturally studied in the framework of MDPs. Further, many of these problems such as flow and congestion control can be realistically studied in a finite-horizon MDP framework since any user typically holds the network for only a finite time duration. It is imperative, from the user's perspective, to develop control strategies for the duration of time that the user is active in the network. High priority applications also place a premium on the time needed to control congestion, while at the same time requiring enough bandwidth. Such issues cannot be realistically captured while modeling using an infinite-horizon time-stationary framework.

As an example, in frame-relay networks, Generic Traffic Shaping (GTS) modules use closed-loop control, with control packets indicating the rate of ingress to the source. If we consider this control-packet data as the action chosen by the router, the queue dynamics can be modeled as an MDP. There even exists one such proprietary scheme for ATM networks, termed Cisco ForeSight (CFS) [24]; our choice of the queue sampling period $T = 5$ s is drawn from CFS.

We mention two specific cases where finite-horizon traffic shaping by GTS modules is of interest.

- The token-bucket protocol implemented by a source permits a burst stream. However, this is a finite-time phenomenon, as the tokens will be replaced at the next (source) epoch. However, there is a possibility that such a burst will swamp the router queue. A router's GTS module could control this behavior of a source, by knowing typical token-bucket replenish times.

- Although on a slower time-scale than the above burst-stream scenario, the time localized behavior of guaranteed-service CBR (constant bit-rate) traffic in ATM networks also suggests finite-horizon control. Here, ABR (available bit rate) traffic can make best use of the network's low-traffic periods (e.g. traffic outside of business hours, say 18:00–09:00) via a suitable finite-horizon GTS module at the router.

For our numerical experiments, we consider the problem of flow and congestion control in communication networks. We consider a continuous time queueing model involving a single bottleneck node that is fed with two arrival streams: an uncontrolled (fixed rate) Poisson stream and a controlled Poisson process with a time-varying rate that is a function of the state of the system. Such a model has also been studied in the simulation optimization framework [25] and the infinite-horizon discounted-cost MDP framework [11].

A continuous-time queueing model of flow control is important in the case of communication networks such as the internet and ATM networks where packet arrivals and departures to and from routers and/or sources can take place at any real-time instant i.e. in continuous time. In the case of the transmission control protocol (TCP) based flow control in the internet, the sources using the additive increase multiplicative decrease (AIMD) algorithm

for flow control update their window sizes once after each round-trip time (RTT). Likewise in the case of ABR flow control in ATM networks, the routers or switches periodically transmit rate feedback information to the sources, which the latter uses to adjust the rates at which they transmit packets.

These network settings are thus natural candidates for studying flow control using MDPs. Here the duration of a stage (time interval between successive decision epochs) may correspond to an RTT in the case of the internet and to the time duration between successive rate feedbacks in the case of ATM networks. Finite-horizon dynamic programming tasks also form natural subproblems in certain kinds of MDPs, e.g. §2 of [16] illustrates the same by using models from semiconductor fabrication and communication networks where the upper level MDP has an infinite horizon and each transition spawns a finite-horizon MDP at a lower level.

The rest of the paper is organized as follows. Section 2 provides the framework and algorithms. In Section 3, convergence analysis for the algorithms is sketched. Section 4 describes the numerical experiments. Finally, Section 5 provides the concluding remarks.

2. Framework and Algorithms

Consider an MDP $\{X_r, r = 0, 1, \ldots, T\}$ with decision horizon $T < \infty$. Let $\{Z_r, r = 0, 1, \ldots, T-1\}$ be the associated control valued process. Decisions are made at instants $r = 0, 1, \ldots, T-1$, and the process terminates at instant T. Let state space at stage r be $S_r, r = 0, 1, \ldots, T$ and let the control space at stage r be $C_r, r = 0, 1, \ldots, T-1$. Note that S_T is the set of terminating states of this process. Thus we consider a general scenario wherein the sets of states and actions may vary from one stage to another. Let $U_r(i_r) \subset C_r, r = 0, 1, \ldots, T-1$, be the set of all feasible controls in state i_r, in stage r. Let $p_r(i, a, j), i \in S_r, a \in U_r(i), j \in S_{r+1}, r = 0, 1, \ldots, T-1$ denote the transition probabilities associated with this MDP. The transition dynamics of the process $\{X_r\}$ depends on $\{Z_r\}$ and is governed according to

$$P(X_{r+1} = i_{r+1} | X_r = i_r, Z_r = a_r, X_{r-1} = i_{r-1}, Z_{r-1} = a_{r-1}, \ldots, X_0 = i_0, Z_0 = a_0) = p_r(i_r, a_r, i_{r+1}),$$

where $r = 0, 1, \ldots, T-1$, for all $i_0, i_1, \ldots, i_T, a_0, a_1, \ldots, a_{T-1}$, in appropriate sets.

We define an admissible policy π as a set of T functions $\pi = \{\mu_0, \mu_1, \ldots, \mu_{T-1}\}$ with $\mu_r : S_r \longmapsto C_r$ such that $\mu_r(i) \in U_r(i), \forall i \in S_r, r = 0, 1, \ldots, T-1$. Thus at (a given) instant r with the system in state i, the controller under policy π selects the action $\mu_r(i)$.

Let $g_r(i, a, j)$ denote the single-stage cost at instant r when state is $i \in S_r$, the action chosen is $a \in U_r(i)$ and the subsequent next state is $j \in S_{r+1}$ for $r = 0, 1, \ldots, T-1$. Also, let $g_T(k)$ denote the terminal cost at instant T when the terminating state is $k \in S_T$. The aim here is to find an admissible policy $\pi = \{\mu_0, \mu_1, \ldots, \mu_{T-1}\}$ that minimizes for all $i \in S_0$,

$$V_0^i(\pi) = E\left\{ g_T(X_T) + \sum_{r=0}^{T-1} g_r(X_r, \mu_r(X_r), X_{r+1}) | X_0 = i \right\}. \quad (1)$$

The expectation above is over the joint distribution of $X_1, X_2, \ldots, X_T$. The dynamic programming algorithm for this problem is now given as follows [2]. For all $i \in S_T$,

$$V_T^i(\pi) = g_T(i) \quad (2)$$

and for all $i \in S_r, r = 0, 1, \ldots, T-1$,

$$V_r^i(\pi) = \min_{a \in U_r(i)} \left\{ \sum_{j \in S_{r+1}} p_r(i, a, j)(g_r(i, a, j) + V_{r+1}^j(\pi)) \right\}. \quad (3)$$

2.1 Overview and Motivation

Note that by using dynamic programming, the original problem of minimizing, over all admissible policies π, the T-stage cost-to-go $V_0^i(\pi), \forall i \in S_0$ (Equation (1)) is broken down into T coupled minimization problems given by Equations (2) and (3). Each problem is defined over the corresponding feasible set of actions for each state. Here,'coupled' means that the solution to the rth problem depends on that of the $(r+1)$st problem, for $0 \leq r < T$.

In general, any solution procedure based on directly solving the dynamic programming equations (above) would require complete knowledge of the transition probabilities $p_r(i, a, j)$. In this paper, we are interested in scenarios where the $p_r(i, a, j)$ are not known however, where transitions can be simulated. In other words, given that the state of the system at stage r ($r \in \{0, 1, \ldots, T-1\}$) is i and action a is picked, we assume that the next state j can be obtained through simulation, even although $p_r(i, a, j)$ are not numerically known quantities. The same approach as we use in our algorithms can also be used when observations are obtained from an actual system instead of a simulated one. Simulated observations are only required if those from the actual system are not available.

We combine the theories of two-timescale stochastic approximation and SPSA to obtain simulation-based

actor-critic algorithms that solve all the T coupled minimization problems (Equations (2) and (3)) under the (above) lack of model information constraint. For the case of infinite-horizon problems, two-timescale stochastic approximation algorithms are used [9–11]. As explained in Section 1, these algorithms are the simulation-based analogs of the policy iteration algorithm.

The algorithms of [9] and [11] follow the general approach of obtaining solutions to the Poisson equation for a given policy update, under lack of model information, along the faster timescale (the policy evaluation step); the policy itself is updated along the slower scale. The algorithm of [10] performs a temporal difference (TD) learning step along the faster scale by assuming a linear parameterized form for the value function and does a policy update along the slower scale. While the algorithms of [9] and [10] are for finite state and finite action spaces (and hence perform updates in the space of randomized policies), the algorithm presented in [11] is for the case of finite state and compact action spaces (and performs updates in the space of deterministic policies). Moreover, in [11], SPSA based gradient estimates are used in policy updates for enhanced performance unlike [9] and [10]. Even although gradient search on the slower timescale is recommended in [10], no specific form of the gradient estimates is proposed there.

Before we proceed further, we first motivate the use of SPSA based gradient estimates and the need for two timescales in our algorithms. Suppose we are interested in finding the minimum of a function $F(\theta)$ when F is not analytically available. However, noisy observations $f(\theta, \xi_n), n \geq 0$ of $F(\theta)$ are available with $\xi_n, n \geq 0$ being i.i.d. random variables satisfying $F(\theta) = E\left[f(\theta, \xi_n)\right]$.

The expectation above is taken with respect to the common distribution of ξ_n, $n \geq 0$. Let $\theta = (\theta_1, \ldots, \theta_N)^T$. Also, let $\Delta_i(n)$, $i = 1, \ldots, N$, $n \geq 0$ be i.i.d., Bernoulli distributed, ± 1-valued random variables with $P(\Delta_i(n) = +1) = P(\Delta_i(n) = -1) = 1/2$ and let $\Delta(n) = (\Delta_1(n), \ldots, \Delta_N(n))^T$, $n \geq 0$. Let $\theta(n)$ denote the nth update of θ. For a given scalar sequence $\{\delta_n\}$ with $\delta_n \downarrow 0$ as $n \to \infty$, form two parameter vectors $\theta(n) + \delta_n \Delta(n)$ and $\theta(n) - \delta_n \Delta(n)$, respectively. Let $\{\xi_n^+\}$ and $\{\xi_n^-\}$ be sequences of i.i.d. random variables, independent of each other and having a common distribution which is the same as that of $\{\xi_n\}$ above. Then the two-measurement SPSA gradient estimate $\tilde{\nabla}_i F(\theta(n))$ of $\nabla_i F(\theta(n))$, $i = 1, \ldots, N$ has the form [12]:

$$\tilde{\nabla}_i F(\theta(n)) = \left(\frac{f(\theta(n)+\delta_n \Delta(n), \xi_n^+) - f(\theta(n)-\delta_n \Delta(n), \xi_n^-)}{2\delta_n \Delta_i(n)} \right). \quad (4)$$

It is this form of SPSA that will be used in the parameterized actor algorithm of Section 2.3 below. Further, the one-measurement SPSA gradient estimate [13] has the form of Equation (5). Note that only one measurement (corresponding to $\theta(n) + \delta_n \Delta(n)$) is required here:

$$\tilde{\nabla}_i F(\theta(n)) = \frac{f(\theta(n) + \delta_n \Delta(n), \xi_n^+)}{\delta_n \Delta_i(n)}, \quad (5)$$

where $i = 1, \ldots, N$. The analysis of SPSA works under much more general conditions on the distribution of $\Delta_i(n)$ [12, 13]. Note that unlike SPSA estimates, Kiefer-Wolfowitz gradient estimates require $2N$ (respectively $(N + 1)$) measurements when symmetric (respectively one-sided) differences are used. Two-measurement SPSA estimates are in general found to show good performance. However, one-measurement SPSA does not perform well because of the presence of certain additional bias terms that contribute significantly to the overall bias in those estimates.

Two deterministic constructions for the perturbations $\Delta(n)$, $n \geq 1$, were proposed [14]. One of these was based on certain normalized Hadamard matrices and was found to considerably improve performance in the case of one-simulation SPSA (over randomized sequences as above). As argued [14], this is due to the fact that the cardinality of the space of deterministic perturbations with Hadamard matrices is less by an (exponential) order of magnitude compared to that of randomized perturbations. Hence, a dominant bias term in the expression for gradient estimates gets canceled much more often in the former case, thereby resulting in much superior performance. We now describe this construction as we use the same in three of the proposed algorithms.

2.1.1 Construction for Deterministic Perturbations

Let H be 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 P with $P \geq N + 1$. Let $h(1), \ldots, h(N)$ be any N columns other than the first column of H, and form a new $(P \times N)$–dimensional matrix $\hat{H}$ which has $h(1), \ldots, h(N)$ as its columns. Let $\hat{H}(p)$, $p = 1, \ldots, P$ denote the P rows of $\hat{H}$. Now set $\Delta(n) = \hat{H}(n \bmod P + 1)$, $\forall n \geq 0$. The perturbations are thus generated by cycling through the rows of the matrix $\hat{H}$. Here P is chosen as $P = 2^{\lceil \log_2 (N+1) \rceil}$. It is shown [14] that under the above choice of P, the bias in gradient estimates asymptotically vanishes. Finally, matrices $H \equiv H_{P \times P}$ of dimension $P \times P$, for $P = 2^k$, are systematically constructed as follows:

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

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

for $k > 1$. Using the above matrices, the space of perturbation vectors has only $P = 2^{\lceil \log_2 (N+1) \rceil}$ elements as opposed to 2^N elements that would result when randomized perturbations are used.

Note that the Bellman equation for the infinite-horizon problem involves $|S|$ minimizations, where $|S|$ is the cardinality of S. On the other hand, the Bellman equation (Equations (2) and (3)) in the finite-horizon setting involves T coupled minimization problems with the rth such problem involving $|S_r|$ minimizations. We view the T-stage finite-horizon policy as a vector of decision rules at each of the T stages. Our algorithms update the entire T-stage policy at each update epoch and converge to the optimal such policy as the number of updates increase. As stated previously, we develop our algorithms for the case when transition probabilities $p_r(i, a, j)$ are not known, hence Equations (2) and (3) cannot be directly solved to obtain the optimal policy.

We develop algorithms for the finite-horizon setting with finite state spaces and under two different settings for the action sets: compact and finite action sets, respectively. In the following, for simplicity and ease of exposition, we assume that the state and action spaces are fixed and do not vary with stage r. Thus S and C shall denote the state and control spaces, respectively. Further, $U(i)$ shall denote the set of feasible actions in state $i \in S$ and is also stage invariant. We now present our algorithm for the setting of compact action sets.

2.2 Finite-Horizon Algorithm for Compact Action Sets

Consider a finite-horizon MDP $\{X_r, r = 0, 1, \ldots, T\}$ with $\{Z_r, r = 0, 1, \ldots, T-1\}$ as its associated control-valued process. Let action sets $U(i), i \in S$ be of the form $U(i) = \prod_{j=1}^{N} \left[a_j^i, b_j^i\right]$. In general, N may depend on state i or stage r (we consider it fixed, however). We make the following assumption:

Assumption(A) For all $i, j \in S, a \in U(i)$, both $g_r(i, a, j)$ and $p_r(i, a, j)$ are continuously differentiable with respect to a. Also, $|g_T(i)| < \infty, \forall i \in S$.

We identify an admissible policy π with the vector $\pi \stackrel{\triangle}{=} (\mu_r(i), i \in S, r = 0, 1, \ldots, T-1)^T$, with its components ordered lexicographically. Further, each action $\mu_r(i) \in U(i)$ is an N-vector denoted by $\mu_r(i) = (\mu_r(i, 1), \ldots, \mu_r(i, N))^T$. We denote by $\Gamma_j^i(y) = \min(b_j^i, \max(a_j^i, y))$, $y \in \mathcal{R}$, the projection of y onto the interval $\left[a_j^i, b_j^i\right], j = 1, \ldots, N$. Also, for any $z \in \mathcal{R}^N$, say $z = (z_1, \ldots, z_N)^T$, let $\Gamma^i(z) = (\Gamma_1^i(z_1), \ldots, \Gamma_N^i(z_N))^T$. Then $\Gamma^i(z)$ denotes the projection of z onto the set $U(i)$, $i \in S$. Similarly, let $\Gamma = (\Gamma^i, \forall i \in S)^T$. We shall use $\Gamma^i(\cdot)$ (respectively $\Gamma(\cdot)$) as the projection operators in our algorithms, to project each action update (respectively policy update) to the feasible set of actions (respectively policies). To accommodate the non-stationarity of the policy, we add a subscript r denoting the stage and define policy updates π_n as $(\mu_{n,r}(i), \forall i \in S, n \geq 0, 0 \leq r \leq T-1)^T$. In particular, each $\mu_{n,r}(i)$ is an N-dimensional vector whose elements are identified as $(\mu_{n,r}(i, j), 1 \leq j \leq N)^T$. Similarly, let $\Delta_{n,r}(i) \stackrel{\triangle}{=} (\Delta_{n,r}(i, j), 1 \leq j \leq N)^T$ be the $\{\pm 1\}^N$-valued perturbation vectors in the SPSA estimates that are obtained using appropriate normalized Hadamard matrices as in Section 2.1.1.

For each $n \geq 0$, let $\{X_r(n), r = 0, 1, \ldots, T\}$, denote a simulated trajectory of the process that is governed with policy $\bar{\pi}_n = (\Gamma^i(\mu_{n,r}(i) + \delta\Delta_{n,r}(i)), i \in S, r = 0, 1, \ldots, T-1)^T$. Here, $\delta > 0$ (say $\delta = 0.1$) is a chosen constant. We iterate over n, the policies π_n in our algorithms. Let $\{b(n)\}$ and $\{c(n)\}$ be two step-size schedules that satisfy

$$\sum_n b(n) = \sum_n c(n) = \infty,$$

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

and

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

respectively. Thus $\{c(n)\}$ goes to zero faster than $\{b(n)\}$ does and corresponds to the slower timescale. This is because beyond some integer N_0 (i.e. for $n \geq N_0$), the sizes of increments in recursions that use $\{c(n)\}$ are uniformly the smallest, and hence result in slow but graceful convergence. Likewise, $\{b(n)\}$ is the faster scale.

Let $\{\eta_{n,r}(i, a)\}, i \in S, a \in U(i), r = 0, 1, \ldots, T-1$ be independent families of i.i.d. random variables such that $\eta_{n,r}(i, a)$, $n \geq 0$ have the distribution $p_r(i, a, \cdot)$, $r = 0, 1, \ldots, T-1$. These random variables are used to generate the simulated 'next' states in our algorithm (see also the remark below). Suppose at instant r ($r = 0, 1, \ldots T-1$), the system is in state $i \in S$ and action $a \in U(i)$ is chosen, then $\eta_{n,r}(i, a)$ would denote the nth sample of the state at instant $(r+1)$ to which the system makes a transition. For economy of notation, we shall denote the random variable $\eta_{nL+m,r}(i, \bar{\mu}_{n,r}(i))$ by $\eta_{nL+m,r}$.

Algorithm for Compact Action Sets (ACA)

- *Step 0 (Initialize): Fix* $\mu_{0,r}(i), \forall i \in S, r = 0, 1, \ldots, T-1$. *Fix integers* L *and a (large)* M *as per desired convergence properties. Fix a (small) constant* $\delta > 0$. *Set* $n := 0$ *and* $m := 0$. *Generate a* $P \times P$ *normalized Hadamard matrix* H, *where* $P = 2^{\lceil \log_2(N+1) \rceil}$. *Let* $\hat{H}$ *be a* $P \times N$ *matrix formed from* H *by choosing any* N *of its columns other than the first and let* $\hat{H}(p)$, $p = 1, \ldots, P$ *denote the* P *rows of* $\hat{H}$. *Now set* $\Delta_{0,r}(i) := \hat{H}(1), \forall i \in S, r = 0, 1, \ldots, T-1$. *Set* $\bar{\pi}_0 = (\Gamma^i(\mu_{0,r}(i) + \delta\Delta_{0,r}(i)), i \in S, r = 0, 1, \ldots, T-1)^T$. *Set* $V_{q,r}(i) = 0$, $\forall 0 \leq r \leq T-1, 0 \leq q \leq L-1, i \in S$.

- *Step 1: For each $i \in S$ and $r = 0, 1, \ldots, T-1$, simulate the corresponding next state $\eta_{nL+m,r}$ according to distribution $p_r(i, \bar{\mu}_{n,r}(i), \cdot)$. (Here $\bar{\mu}_{n,r}(i)$ denotes the ith component of $\bar{\pi}_n$.) Set $V_{nL+m,T}(i) = g_T(i), \forall i \in S$. Next, for all $r = T-1, T-2, \ldots, 0$ and $i \in S$, update*

$$V_{nL+m+1,r}(i) = (1 - b(n))V_{nL+m,r}(i) + b(n)\left[g_r(i, \bar{\mu}_{n,r}(i), \eta_{nL+m,r}) + V_{nL+m,r+1}(\eta_{nL+m,r})\right]. \quad (8)$$

If $m = L-1$, set $nL := (n+1)L$, $m := 0$ and go to Step 2;
else, set $m := m+1$ and repeat Step 1.

- *Step 2: For each $i \in S, r = 0, 1, \ldots, T-1$, update*

$$\mu_{n+1,r}(i) = \Gamma^i\left(\mu_{n,r}(i) - c(n)\frac{V_{nL,r}(i)}{\delta}\left(\Delta_{n,r}(i)\right)^{-1}\right) \quad (9)$$

Set $n := n+1$. If $n = M$, go to Step 3;
else, for all $i \in S$, $0 \le r \le T-1$, set $\Delta_{n,r}(i) := \hat{H}(n \bmod P + 1)$ as the new Hadamard matrix generated perturbation. Set $\bar{\pi}_n = (\Gamma^i(\mu_{n,r}(i) + \delta\Delta_{n,r}(i)), i \in S, r = 0, 1, \ldots, T-1)^T$. Go to Step 1.

- *Step 3 (termination): Terminate algorithm and output $\pi_M = (\mu_{M,r}(i), i \in S, r = 0, 1, \ldots, T-1)^T$ as the final policy.*

Remark: Note that even although the simulated 'next' states $\eta_{n,r}(i, a)$ in our algorithm are generated using the distribution $p_r(i, a, \cdot)$, one normally does not require an explicit computation of $p_r(i, a, \cdot)$ in order to simulate these random variables. As an example, consider a problem of congestion control over a network of P $M/G/1$ queues, for some $P \ge 1$. Suppose M Poisson (traffic) sources send packets to various nodes in this network. The rates at which these sources send packets is based on the state of the system. The above decisions (on rates of the sources) are made every T instants of time.

Suppose $q_i(n)$ and $r_i(n)$, $i = 1, \ldots, P$ respectively denote the queue length at instant nT and the residual service time of the customer in service at time nT. Then $\{(q_1(n), r_1(n), \ldots, q_P(n), r_P(n))\}$ is Markov for any given admissible policy. An explicit computation of transition probabilities in such scenarios can be seen to be extremely difficult. However, the 'next' state $\eta_{n,r}(i, a)$ can be obtained by simply simulating the above network and running it for T time instants with initial state i (e.g. $i = (q_1, r_1, \ldots, q_P, r_P)$) and action a (e.g. corresponding to the vector of source rates). The simulated 'next' state can then be used directly in our algorithm. If, on the other hand, one has access to real data (instead of that obtained through simulation), the same could be used to perform updates in our algorithm.

Recursions (9), termed the 'critic', proceed on the faster timescale $b(n)$ (hence converge faster). In contrast, Equation (9), known as the 'actor', proceeds on the slower timescale $c(n)$. The critic and actor correspond to the policy evaluation and improvement steps of the policy iteration method, respectively. The additional averaging (over L iterations) in the faster timescale recursion (Equation 9) over the two-timescale averaging for given policy update π_n is seen to improve performance.

2.3 Parameterized Algorithm for Compact Action Sets (P-ACA)

The algorithm ACA proposed a simulation-based solution that stored the cost-to-go estimates in a look-up table. The ensuing advantage was that simulation of an entire trajectory of states was not necessary, the repeated simulation of a single transition from each state being sufficient. As in the Introduction, the $|S| \times K$ matrix Φ_r corresponds to $(\phi_r(i), \forall i \in S)^T$ and the feature vector $\phi_r(i)$ for each stage-state pair r, i is equivalent to $(\phi_{r,k}(i),\ 1 \le k \le K)^T$.

These features ϕ_r (which can be constructed based on engineering insight into the system or some metrics to grade the states) become available every time the state is encountered in the process of simulation and do not require storage. We give an example of featurization of the state space S in Section 4. Observe that the feature vectors of states change with the stage $0 \le r \le T-1$, demonstrating an added complication in the F-H MDP when compared to similar state-space featurization in infinite-horizon MDPs.

We approximate the gradient-estimate of the cost-to-go as $\tilde{\bigtriangledown} V_i^r(\pi_n) = (\tilde{\bigtriangledown}_i V_i^r(\pi_n), 1 \le i \le |S|)^T$ using a linear architecture, where $\tilde{\bigtriangledown}_i V_i^r(\pi_n)$ indicates an estimate of the partial gradient of $V_i^r(\pi_n)$ with respect to argument $\mu_{n,r}(i)$. For a parsimonious representation, this estimate is projected onto the span of Φ_r using the Euclidean distance minimizing projection, which is the linear transformation $\Pi_{\Phi_r} = \Phi_r(\Phi_r^T\Phi_r)^{-1}\Phi_r^T$. Thus $\tilde{\bigtriangledown} V_i^r(\pi_n)$ is approximated by $\Pi_{\Phi_r}\tilde{\bigtriangledown} V_i^r(\pi_n)$.

Certain key differences of P-ACA compared with ACA and algorithms currently in literature need to be outlined before the algorithm is proposed, including disadvantages of the proposed scheme.

1. The linear parameterization of the cost-to-go [6] rested on the fact that $V_r(\pi_n) \in \mathcal{R}^{|S|}$ (the stage index r is irrelevant in the treatment of [6]). The compact action set assumption, where $U(i)$ is a subset of $\mathcal{R}$, $\forall i \in S$ and the projection operator Γ help us in a similar parameterization of the 'actor' $\mu_{n,r}$.

2. In the proposed algorithm, the 'critic' used is an estimate of the *gradient* of the cost-to-go at policy iterate π_n. A parsimonious representation of this gradient vector is achieved using projection with operator Π_{Φ_r}. This, as Theorem 2 later shows, is central to showing convergence w.p. 1 to a constrained optimum.

3. Observe that the 'critic' has the same linear parameterization as the policy π_n, i.e. both belong to the span of Φ_r. However, an estimate of the policy-gradient has no obvious interpretation for the operator of the system who selects the featurization. Thus one can featurize from the policy point of view and avoid linking two parameterizations as required in the schemes of [26] or [10]. Our scheme is therefore an actor-critic implementation with a parameterized actor.

4. A notable handicap is that no Dynamic Programming results are made use of. For instance, the Poisson equation (Equation (3)) is avoided. The reason is that any use of projected iterates $\Pi_{\Phi_{r+1}} V^j_{r+1}(\pi)$ in lieu of $V^j_{r+1}(\pi)$ (the RHS of Equation (3)) presents problems, unlike in ACA (Equation 9) previously.

5. The 'Monte-Carlo' nature of the policy evaluation recursion (requiring a simulated trajectory until horizon T is reached) is partly due to this inability to use the Poisson equation. Such simulation is not in itself rare, since use of a finite length simulated trajectory until a designated state i^* is hit is proposed in the algorithm [27]. Note also that the setting of F-H MDPs has no randomness associated with trajectory lengths, which are $(T-r)$ for the rth stage, $0 \leq r < T$. In comparison to ACA, however, a length $(T-r)$ trajectory is generated instead of a single transition, resulting in $O(T^2)$ computation time for every policy update. This is in contrast to $O(T)$ for the ACA algorithm.

6. In a policy gradient algorithm, cost-to-go estimates $\tilde{V}_r(\pi_n)$ are not mandatory; an estimate of the policy gradient $\tilde{\bigtriangledown} V_r(\pi_n)$ suffices. This does not hold for the value iteration approach to finding the optimal policy, however, which we do not consider here.

7. In model-free algorithms, due to the availability of a computer simulation of the system, one has the liberty to simulate a $(T-r)$ length trajectory from any (state, stage) pair $(X_r = i \in S, r)$ even although it is the total cost-to-go that is to be minimized. We use this to good effect in the proposed algorithm, and this assumption is also implicit in the ACA algorithm.

8. Due to the compact action setting, we do not need a likelihood ratio term in the estimation, unlike the term $L_u(i,\theta)$ [27]. Similarly, neither is the gradient of reward term $\bigtriangledown g_r(X_r, \mu_r(X_r), X_{r+1})$ needed. This concession is of utility in systems where one-step transition costs $g_r(X_r, \mu_r(X_r), X_{r+1})$ have the destination state X_{r+1} as an argument. The cost function considered in Section 4 is an example of the above.

We next motivate the proposed algorithm. Similar to the Dynamic Programming algorithm, we update the current policy by moving backwards in horizon with index r from $T-1$ to 0 (although, crucially, we do not use the dynamic programming *principle*). The parameterization is such that action taken in (state, stage) pair (i,r) given policy $\pi_n = (\mu_{n,0}, \mu_{n,1}, \ldots, \mu_{n,T})^T$ is $\Gamma^i(\langle \phi_r(i), \mu_{n,r}\rangle)$ where $\mu_{n,r} \in \mathcal{R}^K$, $K \ll |S|$. However, as in ACA, this action is not used explicitly when simulating a $(T-r)$ length trajectory from the pair (i,r). Instead, the two-simulation SPSA method is applied and perturbation $\delta_n \Delta_{nL+m,r}$ is used to produce the actions $\mu^+_{nL+m,r} = \Gamma^i(\langle \phi_r(i), \mu_{n,r}\rangle + \delta_n \Delta_{nL+m,r})$ and similarly $\mu^-_{nL+m,r} = \Gamma^i(\langle \phi_r(i), \mu_{n,r}\rangle - \delta_n \Delta_{nL+m,r})$. The state i is chosen uniformly from the state-space S and measures system performance at these two policies. While δ_n is a perturbation parameter that diminishes to zero in order that the bias with respect to the true policy gradient at π_n vanishes as $n \to \infty$ [12], $\Delta_{nL+m,r}$ is such that $\Delta_{nL+m,r} = \pm 1$ with probability 0.5 each.

The evaluation of $\Pi_{\Phi_r} \tilde{\bigtriangledown} V_r(\pi_n)$ is undertaken in a manner resembling the LSTD(0) algorithm [28] except that our algorithm is incremental in nature while LSTD(0) is not. Note that $\Pi_{\Phi_r} = \Phi_r(\Phi_r^T \Phi_r)^{-1}\Phi_r^T$ and hence we estimate $\Phi_r^T \tilde{\bigtriangledown} V_r(\pi_n)$ first by making $L \gg 0$ starts from states i chosen uniformly out of S and simulating trajectory $\{X^+_r, X^+_{r+1}, \ldots, X^+_T | X^+_r = i\}$ using actions $\mu^+_{n,r}(i)$ in stage r and state i and actions corresponding to policies $\pi_{n+1,r+m}$, $m \geq 1$ for stages $r+m$ up to $T-1$. The actions for these latter $T-r-1$ stages $l \leq \tilde{r} \leq T-1$ are seen as having converged to the respective constrained optimum $\mu^*_{\tilde{r}}$ due to the relation between the step-sizes $c_{k,\tilde{r}}$ that update these iterates.

We call the accumulated cost in such a trajectory G^+ and observe that since the starting state i is chosen uniformly from S, $|S|\phi_r(i)\tilde{\bigtriangledown}_i V^i_r(\pi_n)$ is a vector whose mean is $\Phi_r^T \tilde{\bigtriangledown} V_r(\pi_n)$. The $K \times K$ matrix $(\Phi_r^T \Phi_r)^{-1}$ is assumed to be pre-stored but can also be estimated using an averaging of the $K \times K$ matrix iterates $\phi_r(i)\phi_r^T(i)$. Analogously, the simulation to compute G^- is performed.

As seen in similar examples in the two-timescale literature [11, 29], the averaging of the 'critic' iterates $|S|\phi_r(i)\tilde{\bigtriangledown}_i V_r(\pi_n)$ occurs on a faster timescale a_n than the scale $c_{n,r}$ used to update the 'actor' $\mu_{n,r}$. This relationship results in the actor recursion viewing the critic recursion as having converged to $\Phi_r^T \tilde{\bigtriangledown} V_r(\pi_n)$ at each update n. The number of policy updates M for which the algorithm below is run is typically decided using some convergence

criteria, making $\bar{\mu}_M$ the policy parameter output of the algorithm.

The $T+2$ step-sizes $c_{n,r}$, b_n and δ_n are only required to fulfill these conditions:

$$b_n, c_{n,r} > 0, \quad \forall n \geq 0, \quad 0 \leq r \leq T-1,$$

$$\sum_n b_n = \sum_n c_{n,r} = \infty, \quad \sum_n b_n^2, \sum_k c_{n,r}^2 < \infty,$$

$$c_{n,r} = o(b_n),$$

and

$$c_{n,r} = o(c_{n,r+1}),$$

respectively. Similarly, step size δ_n used in perturbations is such that $\sum_n \frac{c_{n,r}}{\delta_n} = \infty$, $\sum_n \left(\frac{c_{n,r}}{\delta_n}\right)^2 < \infty$, and $\frac{c_{n,r}}{\delta_n} = o(b_n)$.

The 'actor' recursion that we employ has the form:

$$\mu_{n+1,r} := (\Phi_r^T \Phi_r)^{-1} \Phi_r^T \Gamma_r \times \left(\Phi_r \mu_{n,r} - \frac{c_{n,r}}{2\delta_n}(V_{n,r}^+ - V_{n,r}^-) \cdot \Delta_{n,r}^{-1}\right), \quad (10)$$

where $\cdot$ represents component-wise multiplication, in this case with the vector $\Delta_{n,r}^{-1}$. The proposed algorithm is described as follows:

- *for* $n = 0, 1, \ldots, M$ *do:* {
- *for* $r = T-1, \ldots, 0$ *do:* {
- *Critic: for* $m = 0, 1, \ldots, L-1$ *do:* {
  - *choose start state* i *uniformly from* S. *Initialize* $X_r^+ := i$, $X_r^- = i$.
  - *generate perturbation* $\Delta_{nL+m,r}$
  - $\mu_{nL+m,r}^+ := \Gamma^i(\langle \mu_{n,r}, \phi_r(i) \rangle + \delta_n \Delta_{nL+m,r})$
  - $G^+ := 0$, $t := r$.
  - *while* $t < T$ *do:* {
    if $(t = r)$
    $X_{r+1}^+ := \eta_{nL+m,r}^+(i, \mu_{nL+m,r}^+)$;
    $G^+ := G^+ + g_r(i, \mu_{nL+m,r}^+, X_{r+1}^+)$;
    else
    $\mu_{nL+m,t}^+ := \Gamma^{X_t^+}(\langle \mu_{n,t}, \phi_t(X_t^+) \rangle)$
    $X_{t+1}^+ := \eta_{nL+m,t}^+(X_t^+, \mu_{nL+m,t}^+)$;
    $G^+ := G^+ + g_t(X_t^+, \mu_{nL+m,t}^+, X_{t+1}^+)$;
    endif
    $t := t+1$;
    }
  - $\mu_{nL+m,r}^- := \Gamma^i(\langle \mu_{n,r}, \phi_r(i) \rangle - \delta_n \Delta_{nL+m,r})$
  - $G^- := 0$, $t := r$.
  - *while* $t < T$ *do:* {
    if $(t = r)$
    $X_{r+1}^- := \eta_{nL+m,r}^-(i, \mu_{nL+m,r}^-)$;
    $G^- := G^- + g_r(i, \mu_{nL+m,r}^-, X_{r+1}^-)$;
    else
    $\mu_{nL+m,t}^- := \Gamma^{X_t^-}(\langle \mu_{n,t}, \phi_t(X_t^-) \rangle)$
    $X_{t+1}^- := \eta_{nL+m,t}^-(X_t^-, \mu_{nL+m,t}^-)$;
    $G^- := G^- + g_r(X_t^-, \mu_{nL+m,t}^-, X_{t+1}^-)$;
    endif
    $t := t+1$;
    }
  - $s_{nL+m+1,r} := s_{nL+m,r} + b_n \left(|S| \left(\frac{G^+ - G^-}{2\delta_n \Delta_{nL+m,r}}\right) \phi_r(i) - s_{nL+m,r}\right)$
    }
- *Actor:*

$$\mu_{n+1,r} := (\Phi_r^T \Phi_r)^{-1} \Phi_r^T \Gamma \left(\Phi_r \mu_{n,r} - c_{n,r} \Phi_r (\Phi_r^T \Phi_r)^{-1} s_{(n+1)L,r}\right), \quad (11)$$

}
}

Observe that unlike ACA, the policy $\pi_n = (\mu_{n,r}, 0 \leq r \leq T-1)$ is not indexed with the state i chosen as the start-state for the length $T-r$ trajectory. Similarly, the perturbation $\Delta_{nL+m,r}$ generated is not stored in any array. It is possible that a state j may be chosen twice, at indices m_1 and m_2, from S and have varied perturbations $\Delta_{nL+m_1,r}$ and $\Delta_{nL+m_2,r}$. This phenomenon does not affect the convergence of the algorithm. Also unlike ACA, we require two simulations represented by the random variables $\eta_{nL+m,r}^+(\cdot,\cdot)$ and $\eta_{nL+m,r}^-(\cdot,\cdot)$. The rationale behind using the two-simulation SPSA in P-ACA above as opposed to one-simulation SPSA in ACA has to do with the size of the look-up table $V_{nL+m,r}(\cdot)$ in Step 1 of ACA. This table would be double the size if the two-simulation SPSA was used. In contrast, such a problem does not arise in P-ACA and we can choose to use the smaller variance two-simulation SPSA method. Next, we present two analogs of this algorithm for the case when action sets are finite.

2.4 Finite-Horizon Algorithms for Finite Action Sets

The set $U(i)$ of feasible actions in state i is now assumed finite. For simplicity in notation and ease of exposition, we assume that each set $U(i)$ has exactly $(q+1)$ elements $u(i,0), \ldots, u(i,q)$ that, however, may depend on state i. We make the following assumption on costs.

Assumption (B) The cost functions $g_r(i,a,j), i, j \in S$, $a \in U(i)$, are bounded for all $r = 0, 1, \ldots, T-1$. Further, $g_T(i)$ is bounded for all $i \in S$.

2.4.1 Deterministic Policy Update Algorithm for Finite Action Sets (DPAFA)

Assume all elements of $U(i)$ lie in $\mathcal{R}^N$ and are placed such that the closed convex hull of these points is a compact set of the form $\prod_{j=1}^{N} [a_j^i, b_j^i]$, as before.

The algorithm is then the same as the above algorithm (ACA) for compact action sets except for the following difference. Note that the action $\bar{\mu}_{n,r}(i)$ prescribed in any state i using this algorithm need not be admissible. Hence we project this 'perturbed' action $\bar{\mu}_{n,r}(i)$ as prescribed by ACA onto the discrete set $\bar{U}(i)$ (see above) to obtain admissible perturbed actions which are then used when the system is in state i. If two or more actions in $\bar{U}(i)$ are equidistant to $\bar{\mu}_{n,r}(i)$, we arbitrarily choose one of them.

2.4.2 Randomized Policy Update Algorithm for Finite Action Sets (RPAFA)

Here we work with randomized policies instead of deterministic ones as in the previous two cases. Let $\pi_r(i,a)$ be the probability of selecting action a in state i at instant r and let $\hat{\pi}_r(i)$ be the vector $(\pi_r(i,a), i \in S, a \in \bar{U}(i)\backslash\{u(i,0)\})^T$, $r = 0, 1, \ldots, T-1$. Thus given $\hat{\pi}_r(i)$ as above, the probability $\pi_r(i, u(i,0))$ of selecting action $u(i,0)$ in state i at instant r is automatically specified as $\pi_r(i,u(i,0)) = 1 - \sum_{j=1}^{q} \pi_r(i,u(i,j))$. Hence, we identify a randomized policy with $\hat{\pi} = (\hat{\pi}_r(i), i \in S, r = 0, 1, \ldots, T-1)^T$. Let $\bar{S} = \{(y_1, \ldots, y_q) | y_j \geq 0, \forall j = 1, \ldots, q, \sum_{j=1}^{q} y_j \leq 1\}$ denote the simplex in which $\hat{\pi}_r(i)$, $i \in S$, $r = 0, 1, \ldots, T-1$ take values. Suppose $\bar{P} : \mathcal{R}^q \mapsto \bar{S}$ denotes the projection map that projects $\hat{\pi}_r(i)$ to the simplex $\bar{S}$ after each update of the algorithm below.

Algorithm RPAFA

- *Step 0 (Initialize): Fix $\pi_{0,r}(i,a)$, $\forall i \in S$, $a \in \bar{U}(i)$, $r = 0, 1, \ldots, T-1$ as the initial probabilities for selecting actions in state i. Set $\hat{\pi}_{0,r}(i) = (\pi_{0,r}(i,a), i \in S, a \in \bar{U}(i)\backslash\{u(i,0)\})^T$, $r = 0, 1, \ldots, T-1$. Fix integers L and (large) M arbitrarily. Fix a (small) constant $\delta > 0$. Set $n := 0$ and $m := 0$. Generate a $P \times P$ normalized Hadamard matrix H, where $P = 2^{\lceil \log_2(q+1) \rceil}$. Let $\hat{H}$ be a $P \times q$ matrix formed from H by choosing any q of its columns other than the first and let $\hat{H}(p)$, $p = 1, \ldots, P$ denote the P rows of $\hat{H}$. Now set $\hat{\Delta}_{0,r}(i) := \hat{H}(1)$, $\forall i \in S$, $r = 0, 1, \ldots, T-1$. Set $\bar{\pi}_0 = (\bar{P}(\hat{\pi}_{0,r}(i) + \delta\hat{\Delta}_{0,r}(i)), i \in S, r = 0, 1, \ldots, T-1)^T$ as the initial value of the perturbed randomized policy. For all $i \in S$ and $r = 0, 1, \ldots, T-1$, simulate the action to use, $\phi_{0,r}(i) \in \bar{U}(i)$, according to policy $\bar{\pi}_0$. Set $V_{q,r}(i) = 0$, $\forall 0 \leq r \leq T-1$, $0 \leq q \leq L-1$, $i \in S$ as the initial estimates of the cost-to-go function.*

- *Step 1: For each $i \in S$ and $r = 0, 1, \ldots, T-1$, simulate the next state $\eta_{nL+m,r}$ according to distribution $p_r(i, \phi_{nL+m,r}(i), \cdot)$. Set $V_{nL+m,T}(i) = g_T(i)$, $\forall i \in S$. Next, for all $r = T-1, T-2, \ldots, 0$, $i \in S$, update*

$$V_{nL+m+1,r}(i) = (1 - b(n)) V_{nL+m,r}(i) + b(n) \left[g_r(i, \phi_{nL+m,r}(i), \eta_{nL+m,r}) + V_{nL+m,r+1}(\eta_{nL+m,r}) \right]. \quad (12)$$

If $m = L-1$, set $nL := (n+1)L$, $m := 0$ and go to Step 2;
else, set $m := m+1$ and repeat Step 1.

- *Step 2: For $i \in S$, $r = 0, 1, \ldots, T-1$, $n \geq 0$, we have*

$$\hat{\pi}_{n+1,r}(i) = \bar{P}\left(\hat{\pi}_{n,r}(i) - c(n) \frac{V_{nL,r}(i)}{\delta} (\hat{\Delta}_{n,r}(i))^{-1} \right) \quad (13)$$

Set $n := n+1$. If $n = M$, go to Step 3;
else, for all $i \in S$, $0 \leq r \leq T-1$, set $\hat{\Delta}_{n,r}(i) := \hat{H}(n \bmod P + 1)$ as the new Hadamard matrix generated perturbation. Set $\bar{\pi}_n = (\bar{P}(\hat{\pi}_{n,r}(i) + \delta\hat{\Delta}_{n,r}(i)), i \in S, r = 0, 1, \ldots, T-1)^T$ as the new perturbed randomized policy. Next, for all $r = 0, 1, \ldots, T-1$, simulate the action to use above, $\phi_{nL+m,r}(i) \in \bar{U}(i)$, in each state $i \in S$, according to policy $\bar{\pi}_n$. Go to Step 1.

- *Step 3 (termination): Terminate algorithm and output $\bar{\pi}_M$ as the final policy.*

3. Convergence Analysis

We first present briefly the convergence analysis of our algorithm for the compact action setting and then point out the changes in analysis required for the algorithms in the finite action case. Our convergence results are asymptotic in nature. In particular, our main convergence results show that our algorithms converge asymptotically to one of the optimal policies. It would be desirable to obtain rate of convergence results using our algorithms. In [30], certain rate of convergence results for two-timescale stochastic approximation algorithms are presented in the case when the objective is a linear function. It appears difficult, however, to obtain rate of convergence results for the case when the objective is in fact a general gradient function (as we have). Nevertheless, we observe good numerical performance using our algorithms on a setting involving flow and congestion control in communication networks. The same needs to be tried on more difficult real-life scenarios.

3.1 Convergence Analysis for Compact Action Setting

The analysis proceeds using the standard ordinary differential equation (ODE) approach for stochastic approximation algorithms [31, 32]. Conditions (6) and (7) on the step-sizes are crucially used here. Suitable martingale difference sequences representing the noise and error terms are formed in the recursion updates and are seen to converge as a consequence of the second condition in Equation (6). The first condition in Equation (6) is required to perform a suitable time-scaling of the iterates. Continuously interpolated trajectories of the iterates are then shown to track the trajectories of a corresponding ODE along the same timescale. As a consequence of Equation (7), the two-timescale algorithm in fact tracks the trajectories of a singular ODE. We show the analysis under Assumption (A). Let $\tilde{n} \stackrel{\triangle}{=} \left[\frac{n}{L}\right]$ denote the integer part of $\frac{n}{L}$. Let $\tilde{\pi}_{n,r} = \pi_{\tilde{n},r}$, with elements $\tilde{\mu}_{n,r}(i) = \mu_{\tilde{n},r}(i)$ for $r = 0, 1, \ldots, T-1$, and $\tilde{b}(n) = b(\tilde{n})$.

Lemma 1 *Under Assumption (A), the iterates $V_{n,r}(i)$, $r = 0, 1, \ldots, T-1$, $n \geq 0$, of the algorithm remain uniformly bounded with probability one.*

Proof: Note that by Assumption (A), since $U(i), i \in S$ are compact sets and S is finite,

$$\sup_{\{i,j \in S,\ a \in U(i),\ 0 \leq r \leq T-1\}} |g_r(i,a,j)| \stackrel{\triangle}{=} \tilde{K} < \infty.$$

Similarly, $\sup_{i \in S} |g_T(i)| \stackrel{\triangle}{=} K_T < \infty$. Now note that $\sup_{i \in S, n \geq 0} |V_{n,T}(i)| = K_T < \infty$. Note that by Equation (6), $\tilde{b}(n) \to 0$ as $n \to \infty$. Also, $\tilde{b}(n) > 0, \forall n \geq 0$. Thus $\exists n_0$ such that $\forall n \geq n_0, \tilde{b}(n) \in (0, 1]$. From Equation (9) and the above for $n \geq n_0$, we have

$$\begin{aligned} \left|V_{n+1,r}(i)\right| \leq\ & (1 - \tilde{b}(n)) \left|V_{n,r}(i)\right| \\ & + \tilde{b}(n) \left[\left|g_r(i, \tilde{\mu}_{n,r}(i), \eta_{n,r})\right| \right. \\ & \left. + \left|V_{n,r+1}(\eta_{n,r})\right|\right] \end{aligned} \tag{14}$$

for $r = 0, 1, \ldots, T-1$.

Now for $r = T-1$, for $n \geq n_0$, the RHS of Equation (14) can be seen to be a convex combination of $|V_{n,T-1}(i)|$ and a quantity uniformly bounded with probability one by $\tilde{K} + K_T$. Since $|V_{0,T-1}(i)| < \infty, \forall i \in S$, it follows that $\sup_{i \in S, n \geq 0} |V_{n,T-1}(i)| < \infty$. The above argument can now be repeated successively for $t = T-2, T-3, \ldots, 0$. Hence the iterates (9) remain uniformly bounded with probability one. □

For an action $a \in U(i)$ consider the operator:

$$F_r(i, a, W) \stackrel{\triangle}{=} \sum_{j \in S} p_r(i, a, j)(g_r(i, a, j) + W(j)),$$

where W is the vector $(W(k), k \in S)^T$.

Let $\mathcal{F}_l = \sigma(\tilde{\mu}_{p,r}(i), V_{p,r}(i), p \leq l; \eta_{p,r}, p < l, i \in S, r = 0, 1, \ldots, T-1), l \geq 1$, denote a sequence of associated sigma fields. Consider now sequences $\{M_{l,r}(i), l \geq 1\}, i \in S, r = 0, 1, \ldots, T-1$ defined according to

$$\begin{aligned} M_{l,r}(i) =\ & \sum_{k=0}^{l-1} \tilde{b}(k) \left[g_r(i, \tilde{\mu}_{k,r}(i), \eta_{k,r}) \right. \\ & \left. + V_{k,r+1}(\eta_{k,r}) - F_r(i, \tilde{\mu}_r(i), V_{r+1}(k))\right]. \end{aligned}$$

Lemma 2 *The sequences $\{M_{l,r}(i), l \geq 1\}, i \in S, r = 0, 1, \ldots, T-1$, converge with probability one.*

Proof: Note that $M_{l,r}(i)$ are $\mathcal{F}_l$-measurable for all $l \geq 1$. Further, it is easy to see that

$$E\left[M_{l+1,r}(i) | \mathcal{F}_l\right] = M_{l,r}(i),$$

with probability one. It is now easy to see from Equation (6), Assumption (A) and Lemma 1 that $\{M_{l,r}(i), \mathcal{F}_l\}$ are martingale sequences with almost surely convergent quadratic variation processes. It then follows from Proposition VII.2.3 (c) of [33] that $\{M_{l,r}(i), l \geq 1\}$ are almost surely convergent sequences. □

Consider the system of ODEs: for all $i \in S$, $r = 0, 1, \ldots, T-1$,

$$\dot{\bar{\mu}}_r^i(t) = 0, \tag{15}$$

$$\dot{V}_r^i(t) = F_r(i, \bar{\mu}_r^i(t), V_{r+1}(t)) - V_r^i(t), \tag{16}$$

with $V_T^i(t) = g_T(i), \forall i \in S, t \geq 0$. Note that Equation (15) corresponds to a fixed policy $\bar{\pi}$ that is independent of t. Hence, in place of Equation (16), consider the ODE

$$\dot{V}_r^i(t) = F_r(i, \bar{\mu}_r(i), V_{r+1}(t)) - V_r^i(t), \tag{17}$$

$\forall i \in S, r = 0, 1, \ldots, T-1$, with $V_T^i(t) = g_T(i), \forall i \in S, t \geq 0$, as before. The ODE Equation (17) is an asymptotically stable linear system with $V_r^i(\bar{\pi})$ as its unique asymptotically stable equilibrium point where $V_r(\bar{\pi}) \stackrel{\triangle}{=} (V_r^i(\bar{\pi}), i \in S)^T$ satisfies the Poisson equation

$$V_r^i(\bar{\pi}) = F_r(i, \bar{\mu}_r(i), V_{r+1}(\bar{\pi})). \tag{18}$$

Now suppose that for any bounded, continuous and real-valued function $v(\cdot)$,

$$\hat{\Gamma}_j^i(v(y)) = \lim_{\eta \to 0} \left(\frac{\Gamma_j^i(y + \eta v(y)) - y}{\eta}\right)$$

for $j = 1, \ldots, N$, $i \in S$. Further, for any $y = (y_1, \ldots, y_N)^T$, let

$$\hat{\Gamma}^i(y) = (\hat{\Gamma}^i_1(y_1), \ldots, \hat{\Gamma}^i_N(y_N))^T.$$

The operators $\hat{\Gamma}^i(\cdot)$ are required to force the evolution of the corresponding ODE (below) within the feasible set of actions $U(i)$; see pp. 191 of [31] for a discussion on similar operators. Let $\bigtriangledown^i_r W(\pi)$ denote the N-vector gradient of any function $W(\pi)$ with respect to $\mu_r(i)$. Consider the ODE

$$\dot{\mu}^i_r(t) = \hat{\Gamma}^i(-\bigtriangledown^i_r V^i_r(\pi(t))) \tag{19}$$

for $i \in S, r = 0, 1, \ldots, T-1$. Let $K = \{\pi \mid \hat{\Gamma}^i(\bigtriangledown^i_r V^i_r(\pi)) = 0, \forall i \in S, r = 0, 1, \ldots, T-1\}$ be the set of all fixed points of Equation (19). Also, given $\epsilon > 0$, let $K^\epsilon = \{\pi \mid \exists \pi_0 \in K$ such that $\|\pi - \pi_0\| < \epsilon\}$ be the set of all policies that are within a distance ϵ from K. Let $\{t(n)\}$ be defined as $t(0) = 0$, $t(n) = \sum_{j=0}^{n-1} \tilde{b}(j)$, $n \geq 1$. In what follows, we obtain a continuous time process from the iterates of the algorithm using $\{t(n)\}$. The resulting process is shown below to asymptotically track the trajectories of the associated ODE. Now consider functions $\bar{\omega}_r(t) = (\bar{\omega}^i_r(t), i \in S)^T$, $r = 0, 1, \ldots, T-1$, defined by $\bar{\omega}^i_r(t(n)) = V_{n,r}(i)$ with the maps $t \mapsto V^i_r(t)$ being continuous linear interpolations on $[t(n), t(n+1)]$, $n \geq 0$. Given a constant $T > 0$, define $T_0 = 0$ and $T_n = min\{t(m) | t(m) \geq T_{n-1} + T\}$, $n \geq 1$. Let I_n denote the interval $I_n = [T_n, T_{n+1}]$. Suppose $m_n > 0$ is such that $T_n = t(m_n)$. Consider also functions $\bar{\omega}^i_{n,r}(t)$, $t \in I_n$, $n \geq 0$, defined as $\bar{\omega}^i_{n,r}(T_n) = \bar{\omega}^i_{n,r}(t(m_n))$ and

$$\dot{\bar{\omega}}^i_{n,r}(t) = F_r(i, \tilde{\mu}^i_r(t), \bar{\omega}_{n,r+1}(t)) - \bar{\omega}^i_{n,r}(t) \tag{20}$$

with $\bar{\omega}^i_{n,r}(T_n) = \bar{\omega}^i_{n,r}(t(m_n)) = V^i_r(m_n)$. Also, $\bar{\omega}_{n,r}(t) \stackrel{\triangle}{=} (\bar{\omega}^i_{n,r}(t), i \in S)^T$. Note that

$$\begin{aligned}
&\bar{\omega}^i_r(t(n+1)) \\
&= \bar{\omega}^i_r(t(n)) + \int_{t(n)}^{t(n+1)} F_r(i, \tilde{\mu}^i_r(t), \bar{\omega}_r(t))dt \\
&+ \int_{t(n)}^{t(n+1)} (F_r(i, \tilde{\mu}^i_r(t(n)), \bar{\omega}_r(t(n))) \\
&- F_r(i, \tilde{\mu}^i_r(t), \bar{\omega}_r(t)))dt \\
&+ (M_{n+1,r}(i) - M_{n,r}(i)).
\end{aligned} \tag{21}$$

By Lemma 1, the third term on the RHS of the above equation (within the integral) can be seen to be of order $O(\tilde{b}(n)^2)$. Also, the last term on the RHS above is $O(1)$ because of Lemma 2. Further,

$$\bar{\omega}^i_{n,r}(t) = \bar{\omega}^i_{n,r}(T_n) + \int_{T_n}^{t} F_r(i, \tilde{\mu}^i_r(s), \bar{\omega}_{n,r}(s))ds. \tag{22}$$

From Equations (21) and (22), an application of Gronwall's inequality gives

$$\lim_{n\to\infty} \sup_{t \in I_n} |\bar{\omega}^i_{n,r}(t) - \bar{\omega}^i_r(t)| = 0 \tag{23}$$

with probability one. Now note that the first iteration Equation (9) of the algorithm can be rewritten as

$$\mu_{n+1,r}(i) = \Gamma^i(\mu_{n,r}(i) - \tilde{b}(n)\xi(n)) \tag{24}$$

where $\xi(n) = O(1)$ since $c(n) = O(\tilde{b}(n))$. From Equations (23) and (24), the algorithm ACA can be seen to asymptotically track the trajectories of the ODE Equations (15) and (16) along the faster timescale $\{t(n)\}$. From Theorem 1 of [34], one obtains the following Lemma.

Lemma 3 *For all $i \in S, r = 0, 1, \ldots, T-1$,*

$$\lim_{n\to\infty} |V_{n,r}(i) - F_r(i, \bar{\mu}_{n,r}(i), V_{r+1}(\bar{\pi}(n)))| = 0$$

with probability one.

Finally, we have the following theorem.

Theorem 1 *Given $\epsilon > 0, \exists \delta_0 > 0$ such that $\forall \delta \in (0, \delta_0]$, the algorithm ACA converges to K^ϵ in the limit as $M \to \infty$ with probability one.*

Proof: Suppose $\pi(n) \stackrel{\triangle}{=} (\mu_{n,r}(i), i \in S, r = 0, 1, \ldots, T-1)^T \in \prod_{r=0}^{T-1} \prod_{i \in S} U^\circ_r(i)$, where $U^\circ_r(i)$ denotes the interior of $U_r(i)$. Choose $\delta > 0$ small enough so that $\bar{\pi}(n) = (\mu_{n,r}(i) + \delta\Delta_{n,r}(i), i \in S, r = 0, 1, \ldots, T-1)^T$, i.e. the perturbed policy lies within the feasible region. From Lemma 3 and the Poisson equation for finite-horizon MDPs Equation (18), the recursion Equation (9) of the algorithm can be seen to be asymptotically analogous to the recursion

$$\begin{aligned}
&\mu_{n+1,r}(i) \\
&= \Gamma^i\left(\mu_{n,r}(i) - c(n)\frac{V^i_r(\bar{\pi}_n)}{\delta}(\Delta_{n,r}(i))^{-1}\right).
\end{aligned} \tag{25}$$

Now performing a Taylor series expansion of $V^i_r(\bar{\pi}_n)$ around π_n, one can show (using Theorem 2.5 and Corollary 2.6 of [14]) that Equation (25) is asymptotically equivalent to

$$\mu_{n+1,r}(i) = \Gamma^i\left(\mu_{n,r}(i) - c(n)\bigtriangledown^i_r V^i_r(\pi_n)\right). \tag{26}$$

Constructing $t(n)$ using $c(n)$ (instead of $\tilde{b}(n)$ as done to motivate Lemma 3), and letting $\delta \to 0$ and $M \to \infty$, Equation (26) can be seen to be a discretization of the ODE Equation (19). For π_n on the boundary of

$\prod_{r=0}^{T-1}\prod_{i\in S}U_r(i)$, either $\bar{\pi}_n = (\mu_{n,r}(i) + \delta\Delta_{n,r}(i), i \in S, r = 0, 1, \ldots, T-1)^T$ for sufficiently small δ as before (in which case the above continues to hold) or else there exists an $r_0 \in \{0, 1, \ldots, T-1\}$ and a state $i \in S$ and $j \in \{1, 2, \ldots, N\}$ such that $\Gamma_j^i(\mu_{n,r_0}(i, j) + \delta\Delta_{n,r_0}(i, j)) = \mu_{n,r_0}(i, j)$. The algorithm can be seen to converge to a fixed point even if it lies on the boundary of the feasible region. Note however that there could be spurious fixed points on the boundary to which the algorithm may converge as with any projection based scheme (pp. 79 of [32]). However, in our algorithm, since $\Delta_{n,r_0}(i, j)$ changes sign within a (small) finite number of steps, the unprojected perturbed policy update will lie within the feasible region and the algorithm will restart (giving a new policy) after hitting the boundary of the region, until convergence is achieved. Now observe that $\bigtriangledown_r^i V_r^i(\pi)\cdot\hat{\Gamma}^i(-\bigtriangledown_r^i V_r^i(\pi)) < 0$ outside the set K. It is easy to see that K serves as an asymptotically stable attractor set for Equation (19) with $\sum_{r=0}^{T-1}\sum_{i\in S}V_r^i(\pi)$ as the associated strict Liapunov function. The claim follows. □

3.2 Convergence Analysis for P-ACA

The result below shows that, for a given stage r, the iterates in Equation (11) converge to a constrained optimum $\mu_r^* \in span(\Phi_r)$, discounting the possibility of spurious minima on the boundary of the feasible action set U. We adapt the algorithm in Section 5.1.5 of [31] which treats constrained function minimization using the well known Keifer–Wolfowitz algorithm.

Theorem 2 *$\mu_{n,r}$ converges w.p. 1 to the set of constrained optima μ_r^* i.e. $\{\mu_r^* : \Pi_{\Phi_r}\bigtriangledown_r V_r(\pi^*) = 0,\ \mu_r \in U;\ 0 \le r \le T-1\}$, where $\pi^* = (\mu_0^*, \mu_1^*, \ldots, \mu_{T-1}^*)^T$.*

Proof: We reproduce the algorithm in Section 5.1.5 of [31] for clarity and explain the terms:

$$\begin{aligned} X_{n+1} &:= X_n - a_n\left[\pi(X_n)Df(X_n, c_n)\right. \\ &- \left.\pi(X_n)\beta_n - \pi(X_n)\xi_n\right] \\ &- ka_n\Phi^T(X_n)\phi(X_n), \end{aligned} \tag{27}$$

where $X_n \in \mathcal{R}^r$ is the iterate and $Df(X_n, c_n)$ is a finite difference approximation to $\bigtriangledown_x f(X_n)$, the gradient at X_n of objective function $f : \mathcal{R}^r \mapsto \mathcal{R}$. The bias in such an estimate is represented by β_n whilst ξ_n is noise with certain conditions. Assumptions A5.1.1–A5.1.6 of [31] cover these.

The constraint set is denoted as $\{x : \phi(x) = \underline{0} \in \mathcal{R}^s, x \in \mathcal{R}^r\}$ where $\phi(x) \equiv (\phi_1(x), \phi_2(x), \ldots, \phi_s(x))^T$. Note that the ϕ above are not to be confused with feature vectors ϕ_r that we use. Further, $\Phi(X_n)$ is a matrix such that

$$\Phi^T(X_n) = \left(\bigtriangledown_x\phi_1(X_n), \bigtriangledown_x\phi_2(X_n), \ldots, \bigtriangledown_x\phi_s(X_n)\right)$$

(making $\Phi^T(X_n)$ an $r \times s$ matrix). The matrix $\pi(X_n)$ in Equation (27) is obtained from projection $(I - \pi(X_n))$ to the span of the columns of $\Phi^T(X_n)$, while k is an arbitrary but fixed positive number.

We shall use the SPSA gradient estimate $\tilde{\bigtriangledown}V_{n,r} \stackrel{\triangle}{=} \frac{1}{2\delta_n}(V_{n,l}^+ - V_{n,l}^-)\cdot\Delta_{n,r}^{-1}$ in place of $Df(X_n, c_n)$ in Equation (27). This is justified since replacing c_n with δ_n and using Lemma 1 of [12] results in A5.1.11 of [31] being satisfied. We now rewrite Equation (27) replacing iterates X_n with $\mu_{n,r}$ as follows

$$\begin{aligned} \mu_{n+1,r} &:= \mu_{n,r} - a_n\left[\pi(\mu_{n,r})\tilde{\bigtriangledown}_r V_{n,r}\right] \\ &- ka_n\Phi^T(\mu_{n,r})\phi(\mu_{n,r}). \end{aligned} \tag{28}$$

The subscript r will distinguish our notation from the Φ and ϕ of Equation (27) as we perform the following substitutions. The constraint set here is $\{\mu_r : (I - \Pi_{\Phi_r})\mu_r = 0\}$, indicating that $\mu_r \in span(\Phi_r)$. Similarly $\Phi^T(\mu_r)$ can be seen to be $(I - \Pi_{\Phi_r})^T$, $\forall\mu_r$. Observe that if $\mu_{n,r} \in span(\Phi_r)$ then $\phi(\mu_{n,r}) = 0$, canceling the last term in the RHS of Equation (28). Also note that $\pi(\mu_{n,r})$ is such that $(I - \pi(\mu_{n,r})) : \mathcal{R}^{|S|} \longrightarrow \mathcal{R}^{|S|}$ is a projection to the span of the columns of Φ^T, the symmetry of $I - \Pi_{\Phi_r}$ meaning that $\pi(\mu_{n,r}) = \Pi_{\Phi_r}$, $\forall\mu_{n,r}$. Thus the recursion is now simplified to:

$$\mu_{n+1,r} := \mu_{n,r} - a_n\left[\Pi_{\Phi_r}\tilde{\bigtriangledown}_r V_{n,r}\right].$$

Observe that pre-multiplying the 'actor' with Φ_r in the proposed algorithm (11) yields the above equation. To conclude, the ODE evolution is also constrained to within U, an issue treated in Theorem 4.1 of [11]. □

3.3 Convergence Analysis for Finite Action Setting

We use Assumption (B) here.

3.3.1 DPAFA

The key idea in this algorithm is to perform gradient search in the convex hull $U(i)$ of the set of feasible actions $\bar{U}(i), i \in S$. Suppose one replaces $g_r(i, u, j)$ and $p_r(i, u, j)$ by $\bar{g}_r(i, u, j)$ and $\bar{p}_r(i, u, j)$, respectively, for $i, j \in S, u \in U(i)$, such that $\bar{g}_r(i, u, j) = g_r(i, u, j)$ and $\bar{p}_r(i, u, j) = p_r(i, u, j), u \in \bar{U}(i)$ and such that $\bar{g}_r(i, u, j)$ and $\bar{p}_r(i, u, j)$ are continuously differentiable functions of $u \in U(i)$. Consider now a new system with transition dynamics governed according to $\bar{p}_r(i, u, j)$ and one-stage costs $\bar{g}_r(i, u, j)$ for which the ACA algorithm is used. Then one would obtain an optimal policy in the limit as $\delta \to 0$ and $M \to \infty$ for the new system. One can

then see that for the original system, the algorithm would converge to either the optimal policy or to a point in its immediate neighborhood. A similar idea as this algorithm has been used [35] for measurement-based optimization in a resource allocation problem, with the exception that the algorithm is restarted after projecting the iterates to the discrete set.

3.3.2 RPAFA

The analysis for this case proceeds in an entirely analogous manner as for the ACA algorithm except that we work with randomized policies $\hat{\pi}_r(i,\cdot)$ (specified earlier) instead of deterministic policies, so as to perform a gradient search in a continuous space. For a given policy $\hat{\pi}$ specified by the slower timescale recursion Equation (13), the faster timescale recursion Equation (13) can be seen to track the solution to the corresponding Poisson equation

$$V_r^i(\bar{\pi}_n) = \sum_{a \in U_r(i)} \bar{\pi}_{n,r}(i,a) F_r(i,a,V_{r+1}(\bar{\pi}_n)),$$

$$r = 0, 1, \ldots, T-1,$$

with $V_T^i(\bar{\pi}_n) = g_T(i)$, $\forall i \in S$. The preceding analysis (for ACA) with suitable modifications carries through for this case as well and the conclusions of Theorem 1 continue to hold.

4. Simulation Results

We consider a continuous time queuing model of flow control. Such a model is important in the case of say the internet or ATM networks where the characteristics of the system change in real (i.e. continuous) time. For instance, in the case of the internet, packet arrivals (departures) to (from) routers and/or sources can take place at any real time instant. The numerical setting that we consider is somewhat similar to others [11, 25]. This problem has been modeled in the infinite-horizon discounted-cost MDP framework [11], as well as been considered in the simulation optimization setting [25]. We study this problem in the finite-horizon MDP framework here.

A single bottleneck node has a finite buffer of size B. Packets are fed into the node by both an uncontrolled Poisson arrival stream with rate $\lambda_u = 0.2$, and a controlled Poisson process with rate $\lambda_c(t)$ at instant $t > 0$. Service times at the node are i.i.d., exponentially distributed with rate 2.0. We assume that the queue length process $\{x_t, t > 0\}$ at the node is observed every $\tilde{T}$ instants, for some $\tilde{T} > 0$, up to the instant $\tilde{T}T$. Here T stands for the terminating stage of the finite-horizon process. Suppose i_r denotes the queue length observed at instant $r\tilde{T}$, $0 \leq r \leq T$. This information is fed back to the controlled source which then starts sending packets with a rate $\lambda_c(r)$ in the interval $[r\tilde{T}, (r+1)\tilde{T})$. We assume no feedback delays here.

The one-step transition cost under a given admissible policy $\pi = \{\mu_0, \mu_1, \ldots, \mu_{T-1}\}$ is computed as follows. For $r = 0, 1, \ldots, T-2$,

$$g_r(i_r, \mu_r(i_r), i_{r+1}) = \left| i_{r+1} - \left(0.8B - \frac{r}{T} \times 0.6B\right) \right|, \qquad (29)$$

while $g_{T-1}(i_{T-1}, \mu_{T-1}(i_{T-1}), i_T) = i_T$. Also, $g_T(i) = 0, \forall i \in S$. One-stage cost functions as above penalize states away from the target states $\hat{T}_{r+1} \stackrel{\triangle}{=} \left(0.8B - \frac{r}{T} \times 0.6B\right)$, $r = 0, 1, \ldots, T-2$ and $\hat{T}_T \stackrel{\triangle}{=} 0$. Since we use $B = 50$ and $T = 10$, the target states (in the above single-stage costs) chosen are $40, 37, 34, \ldots, 16, 0$ for stages $1, 2, \ldots, 10$, respectively. The goal is therefore to maximize throughput in the early stages (r small), while as r increases, the goal steadily shifts towards minimizing the queue length and hence the delay as one approaches the termination stage T. Further, since $\hat{T}_T = 0$, the target at termination is the state 0.

We now briefly describe the implementation of our ACA algorithm. The implementations for the other algorithms are along similar lines. Associated with each time epoch r, $r = 0, 1, \ldots, T-1$ are quantities $V_{nL+m,r}(i)$, $n \geq 0$, $m = 0, 1, \ldots, L-1$, $i \in S$ and $\mu_{n,r}(i)$, $n \geq 0$, $i \in S$ that correspond to the value and policy updates respectively. Here n and m are update indices within the algorithm while r correspond to real-time instants. Note that our implementation is done off-line even although one can also propose online variants of the same. From each state i_r, the next state $\eta_{nL+m,r}$ in the $(nL+m)$th update of quantities $V_{nL+m,r}(i)$ is obtained via simulation in the following manner.

Set the state at instant r to be i_r in the simulation and let the system run. Next, record the state after T time instants. The new state corresponds to $\eta_{nL+m,r}$. The action in state i_r chosen is $\mu_{n,r}(i_r)$ as per the update Equation (9). Thus the single-stage cost in Equation (29) is obtained with $i_{r+1} = \eta_{nL+m,r}$. Note also that the cost-to-go estimates $V_{nL+m,t}(i), \forall t = 0, 1, \ldots, T-1, \forall i \in S$ are already recorded in the previous updates of Equation (9). Thus, the value $V_{nL+m,r+1}(\eta_{nL+m,r})$ is also obtained and the algorithm update Equation (9) is performed. As explained before, Equation (9) is performed for L time instants after which Equation (9) is performed. Note that at each simulation instant n, we use the policy $\bar{\pi}_n = (\Gamma^i(\mu_{n,r}(i) + \delta\Delta_{n,r}(i)),\ i \in S, r = 0, 1, \ldots, T-1)^T$, where perturbations $\Delta_{n,r}(i)$ are obtained using the Hadamard matrix based construction in Section 2.1.1.

For the case of compact action sets, we let the action set in each state be the set of rates (in the interval) [0.05, 4.5] from which the controlled source selects the rate for transmitting packets over the next $\tilde{T}$ instants.

Table 1. Mean one-stage costs $E(|i_r - \hat{T}_r|)$

	$r = 2$	$r = 4$	$r = 6$	$r = 8$	$r = 10$
Target $\hat{T}_r$	37	31	25	19	0
DP	10.48±5.49	4.81±3.23	5.18±3.61	4.98±3.28	5.75±3.59
SAMW	13.95±6.04	5.16±3.50	5.09±3.31	5.41±3.02	4.56±2.38
ACA	10.88±5.61	4.18±2.73	4.21±2.83	4.12±2.70	6.13±3.27
RPAFA	17.94±6.33	7.72±4.43	6.67±3.87	5.89±3.44	8.78±5.28
DPAFA	10.52±5.49	4.79±3.14	5.02±3.31	4.89±3.14	5.51±3.32

Note that the single-stage cost is bounded and does not depend explicitly on actions. Further, the (time homogeneous) transition probabilities for this problem are shown to be continuously differentiable functions of the source rate [25]. Thus Assumption (A) is clearly satisfied here. For the finite action setting, we discretize the (above) interval [0.05, 4.5] so as to obtain five equally spaced actions in each state. Assumption (B) is also valid here. For purposes of comparison, we also implemented the DP algorithm Equations (1) and (2) for the finite action setting. Note, however, that for applying DP, one requires information on transition probabilities.

As stated before, after starting from a state i_r where action $\mu_r(i_r)$ is applied, state i_{r+1} is observed after $\tilde{T}$ instants. For applying DP, in order to compute the transition probability matrix $P_{\tilde{T}}$ for the embedded Markov chain under policy π, we use the approximation method suggested in [36, section 6.8]. Assuming that each state has the same k admissible controls, k number of $P_{\tilde{T}}$ matrices of size $B \times B$ each are required. This task becomes prohibitive as the state space increases in size or the discretization is made finer. Further, the amount of computation also depends upon the convergence criterion specified. Moreover, such probabilities can only be computed for systems whose dynamics are well known, our setting being that of a well-studied $M/M/1/B$ queue.

We also compare against the simulation-based algorithm termed SAMW (Simulated Annealing with Multiplicative Weights) proposed [37], which is not an actor-critic algorithm. We reproduce the algorithm here:

$$\phi^{i+1}(\pi) := \phi^i(\pi)\frac{\beta_i^{V_i^\pi}}{Z^i}, \quad \forall \pi \in \Pi,$$

where $\phi^i(\pi)$ is the probability of choosing a policy $\pi \in \Pi$ to control the system in the ith iteration. By simulating a trajectory using policy π, we obtain V_i^π as a sample estimate of the corresponding value function. While indexed with i, the term $\beta_i > 0$ is in fact constant and is decided after choosing T (the total number of iterations). Note that the terminating iteration T is, in general, unknown. Further, Z^i is the *zustadsumme*, the normalizing term $\sum_{\forall \pi \in \Pi} \beta_i^{V_i^\pi}$.

Although the authors also prove convergence for a computationally light online sampling variant, both algorithms suffer from the curse of dimensionality. To see this, note that the 'meta-policy' iterate ϕ has one entry per feasible *policy*, making it exponential in the number of stages r and states x. The methods DPAFA and RPAFA (or even the ideal comparison, DP) require much smaller data structures. In the algorithms that we propose, it is decided by convergence criteria such as the err_n described below. To produce the better convergence properties, we chose $\beta_i = 1.025$ in the SAMW experiments shown here. We infer from Table 1 that SAMW does outperform RPAFA, but is bettered by DPAFA.

The policy obtained using finite-horizon DP with $P_{\tilde{T}}$ computed as above is shown in Figure 1. Note how the policy graphs shift to the left as the target state moves to the left for increasing r. Thus, the throughput of the system decreases from one stage to another as the target queue length is brought closer to zero. Also shown in Figure 2 is the finite-horizon cost for each state in a system operating under the (finite-horizon) policy of Figure 1.

The algorithms ACA and DPAFA for the compact and finite action settings, respectively, are terminated with a convergence criterion $err_n \leq 0.1$, where err_n is given by

$$err_n = \max_{i \in S, k \in \{1,2,\ldots,50\}} \left\{ \sqrt{\sum_{r=0}^{T-1} |\mu_{n,r}(i) - \mu_{n-k,r}(i)|} \right\}.$$

Here $\mu_{n,r}(i)$ specifies the feedback source rate when i is the rth state to be observed, $0 \leq r \leq T-1$, at policy update n. The convergence criterion chosen for the algorithm RPAFA is similar to $err_n \leq 0.01$, where err_n is now defined as

$$err_n = \max_{i \in S, k \in \{1,2,\ldots,50\}} \times \left\{ \sqrt{\sum_{r=0}^{T-1} \sum_{a \in U_r(i)} |\pi_{n,r}(i,a) - \pi_{n-k,r}(i,a)|} \right\}.$$

Here, $\pi_{n,r}(i,a)$ denotes the probability of choosing rate a in the rth observed state i for the policy obtained at the nth update. We choose err_n to be much lower here because RPAFA performs a search in the space of probabilities, components of which are all ≤ 1. The policies obtained in all three cases are shown in Figures 3–5 while

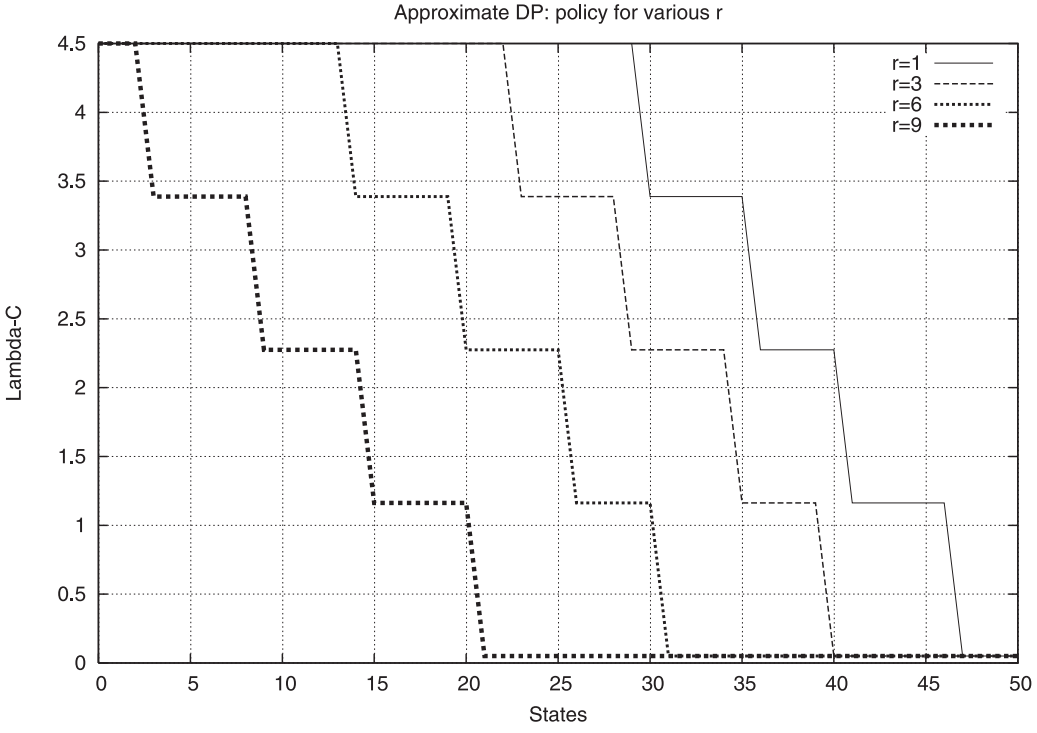

Figure 1. Optimal policy computed using DP

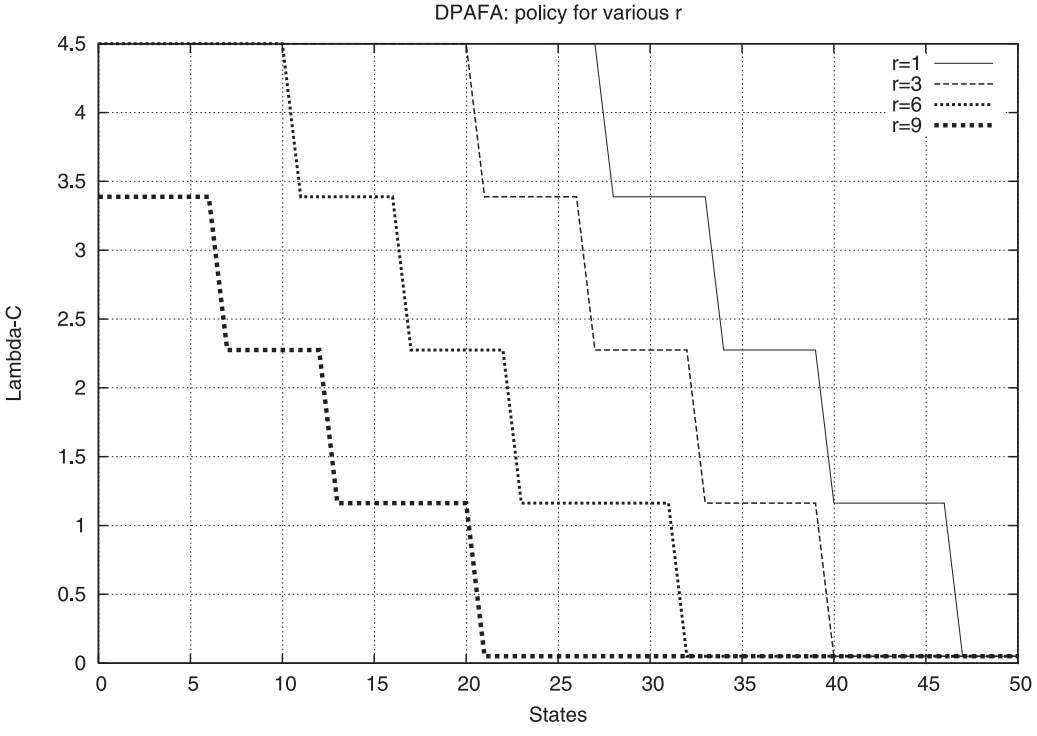

Figure 2. Finite-Horizon cost with DP

Table 2. Observed $E(i_r)$ for the proposed algorithms

	$r = 2$	$r = 4$	$r = 6$	$r = 8$	$r = 10$
Target $\hat{T}_r$	37	31	25	19	0
DP	27.14±6.28	29.36±4.85	22.43±5.10	16.51±4.71	5.73±3.56
SAMW	27.21±6.29	29.19±4.94	22.42±5.13	16.35±4.69	5.60±4.70
ACA	26.87±6.22	29.73±4.13	23.32±4.18	17.29±4.05	6.19±3.34
RPAFA	19.22±6.51	24.26±5.16	19.37±4.79	15.11±4.89	8.86±5.28
DPAFA	27.09±6.16	28.36±4.52	22.03±4.59	15.93±4.34	5.52±3.34

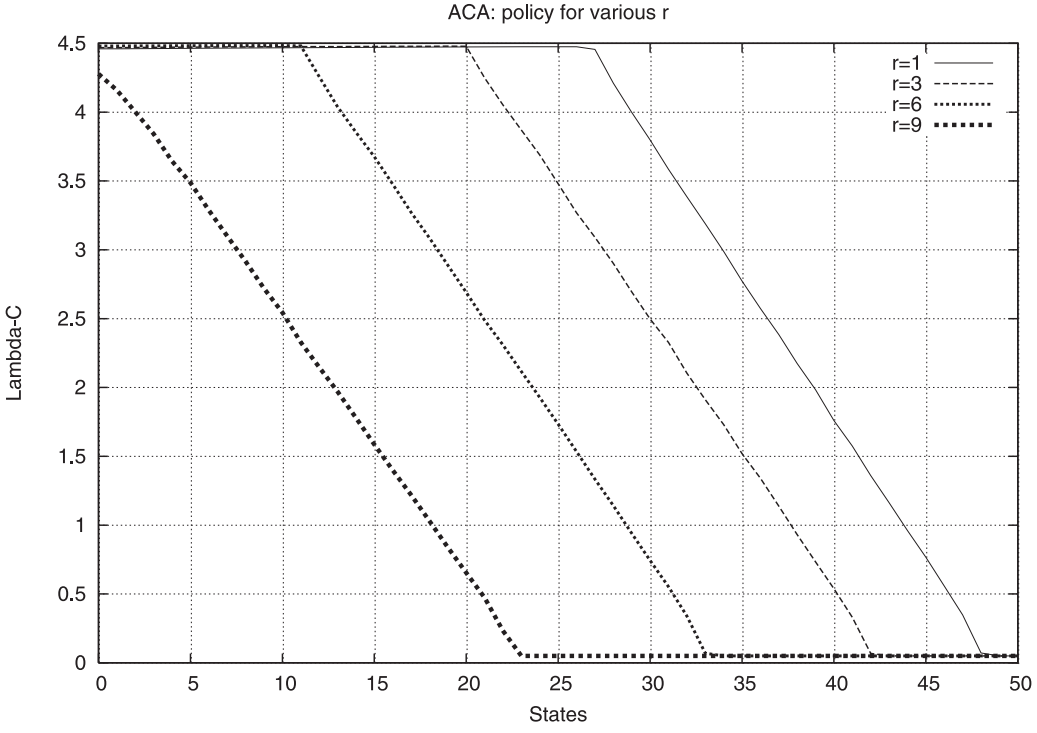

Figure 3. Converged policy using Algorithm ACA for compact action setting

the corresponding finite-horizon costs are plotted in Figures 6–8, respectively. The plots shown in Figures 2 and 6–8 are obtained from 2×10^5 independent sample trajectories $\{i_0, i_1, \ldots, i_T\}$, each starting from state $i_0 = 0$ with a different initial seed. The solid lines in these figures correspond to the finite-horizon costs obtained by averaging over the above samples. Also, the plots in non-solid lines correspond to the mean $\pm$ standard deviation obtained from these samples. In Figure 4, for each state the source rate indicated is the rate that has the maximum probability of selection. The costs in Figures 2 and 7 (also many states in Figure 8) are higher than those for the compact action case (cf. Figure 6) due to the significantly broader range of actions available in the latter setting.

To evaluate P-ACA, we use $B = 1000$ and $T = 5$, and designate the target states $\hat{T}_l$ as evenly spaced states within the queue i.e. $\hat{T}_l = \left(\frac{T-l}{T+1}\right) B$ that shift gradually to 0 as the stage l increases from 0 to $T/2$. Recall that we set $T = 10$ in our experiments. Also, in particular, we select $\hat{T}_1 = 666$, $\hat{T}_2 = 500$, $\hat{T}_3 = 333$, $\hat{T}_4 = 166$ and $\hat{T}_5 = 0$ as the target states. The one-step transition cost under a given policy π is computed as $g_r(X_r, \mu_r(X_r), X_{r+1}) = \left|X_{r+1} - \hat{T}_{r+1}\right|$. Also, as before, $g_T(i) = 0, \forall i \in S$. We use an order 3 polynomial approximation, implying storage of $K = 4$ coefficients. The feature vector $\phi_r(i) = (\phi_{r,k}(i), 1 \leq k \leq K)^T$ for each state i that we use is

$$\phi_{r,k}(i) = \left(5.0 + \frac{i - \hat{T}_r}{\hat{T}_r}\right)^{k-1}.$$

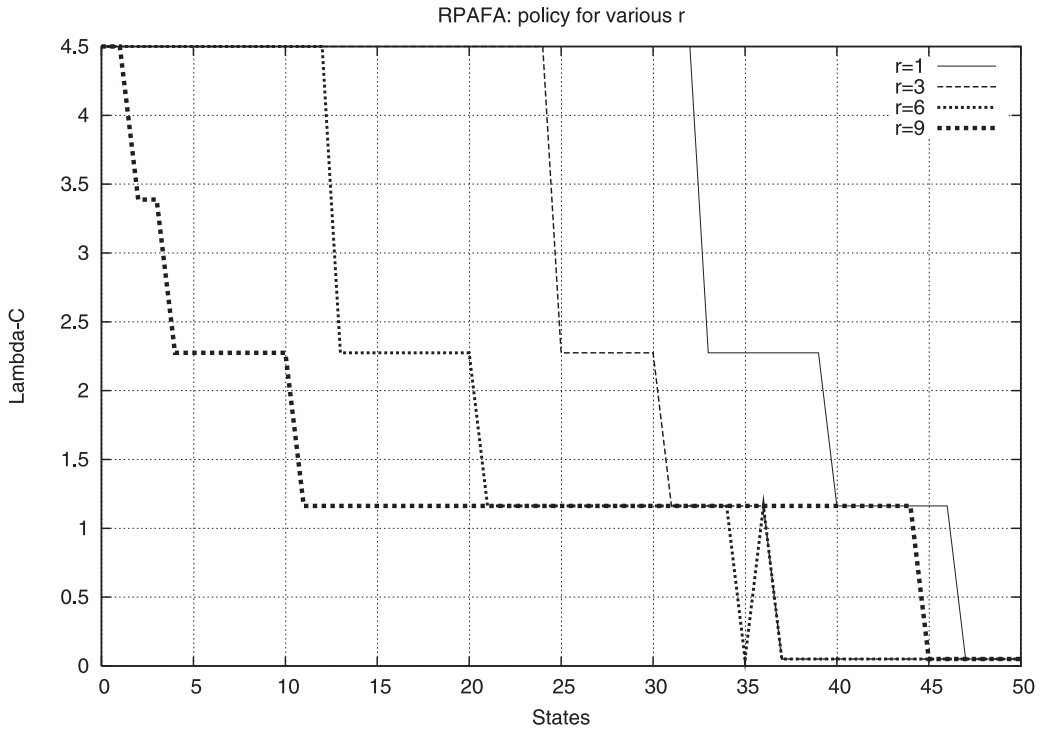

Figure 4. Actions of *maximum* probability obtained from converged RPAFA

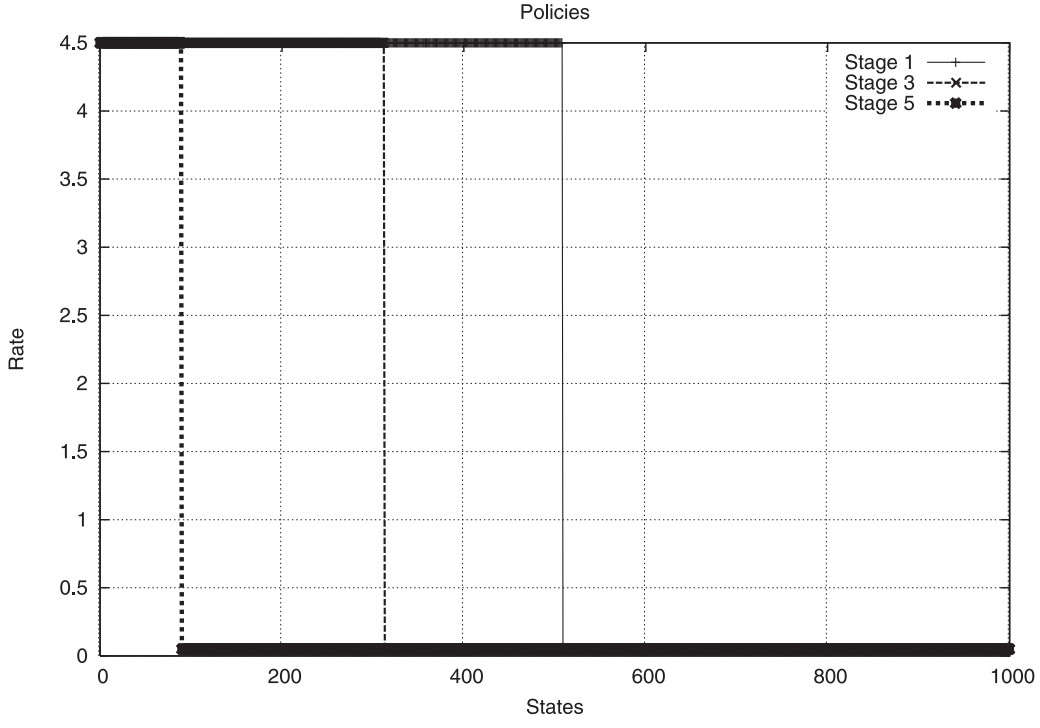

Figure 5. Converged policy using Algorithm DPAFA for discrete action setting

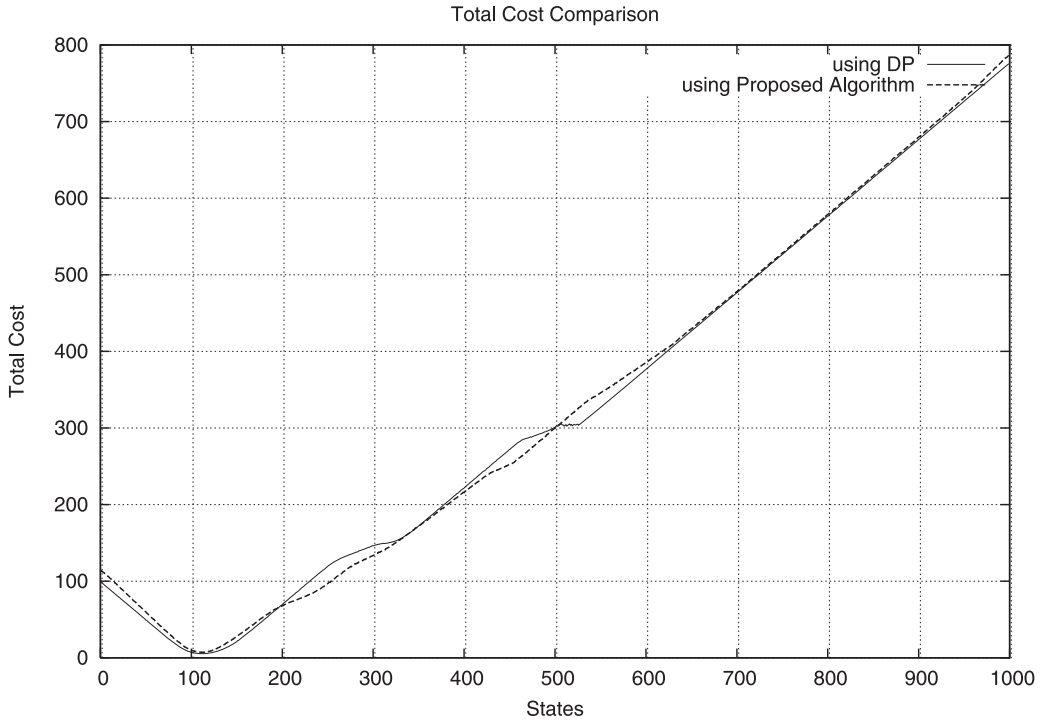

Figure 6. Finite-Horizon cost using Algorithm ACA for compact action setting

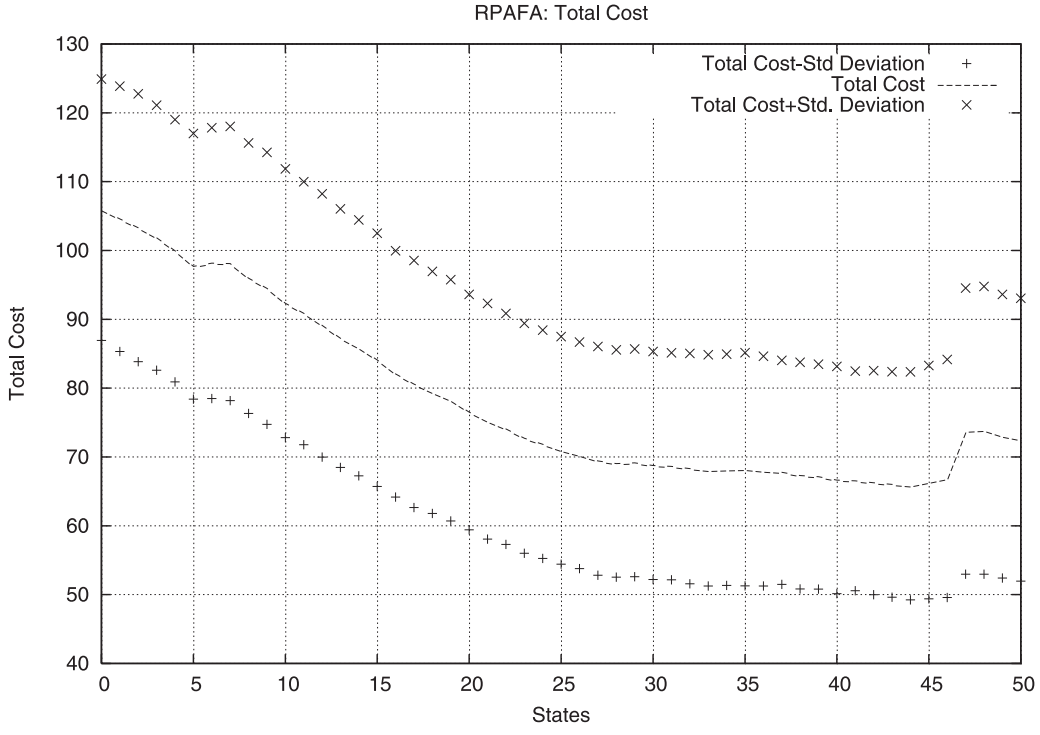

Figure 7. Finite-Horizon cost using Algorithm RPAFA for discrete action setting

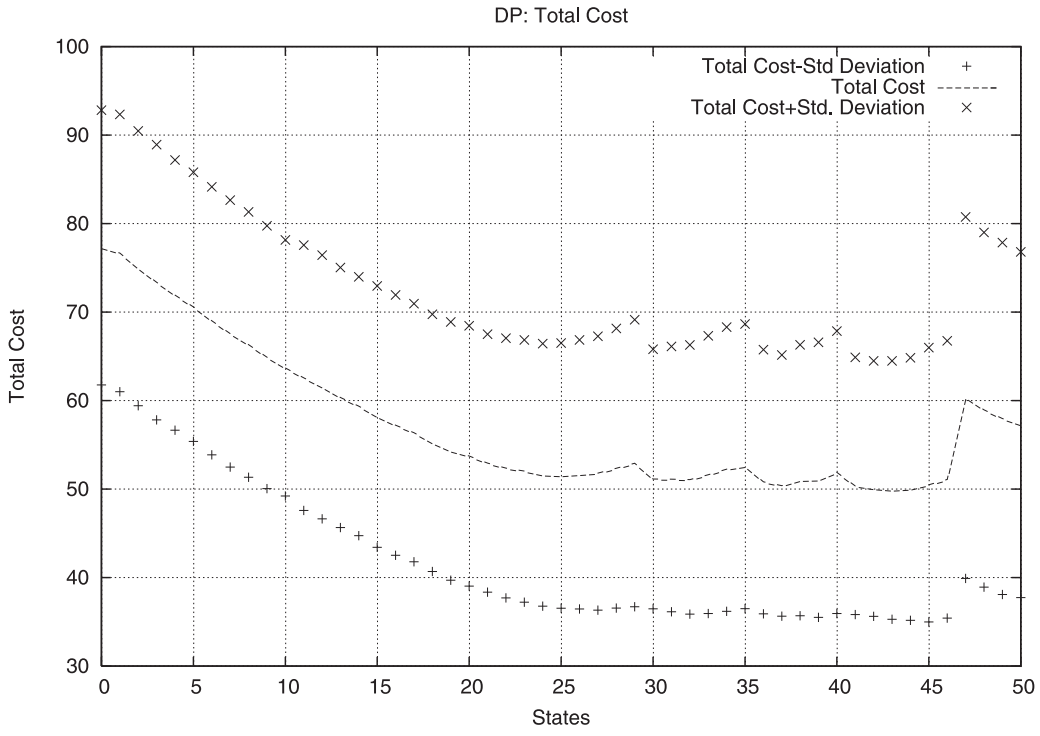

Figure 8. Finite-Horizon cost using Algorithm DPAFA for discrete action setting

Table 3. Probabilities $p_r = P(i_r = \hat{T}_r \pm 1)$

	$r = 2$	$r = 4$	$r = 6$	$r = 8$	$r = 10$
Target $\hat{T}_r$	37	31	25	19	0
DP	0.07±0.13	0.22±0.34	0.21±0.34	0.22±0.34	0.19±0.31
SAMW	0.03±0.05	0.14±0.25	0.13±0.23	0.10±0.19	0.10±0.18
ACA	0.07±0.13	0.25±0.37	0.25±0.38	0.25±0.37	0.13±0.22
RPAFA	0.01±0.03	0.11±0.20	0.13±0.22	0.14±0.26	0.14±0.24
DPAFA	0.07±0.13	0.21±0.34	0.20±0.32	0.21±0.33	0.19±0.30

We set $L = 50$, thus one requires 50 starts from each state-stage pair. We use $\delta_n = n^{-0.167}$, $b_n = n^{-0.55}$ whilst the 'actor' step-size $c_{n,r}$ is chosen to be $n^{-1+0.05r}$. We compute the matrices $(\Phi_r \Phi_r)^{-1}$ needed in Equation (11) before the algorithm iterations commence. The optimal policy computed by the proposed algorithm is shown in Figure 9. A comparison of the cost-to-go for each stage upon using the two policies is shown in Figure 10.

The costs-to-go shown are obtained from 10^5 independent sample trajectories $\{X_0, X_1, \ldots, X_T\}$ for each initial state $X_0 = i$ and with a different initial seed. Note that both the optimal policy and cost-to-go computed using the two algorithms are similar. In particular, as the number of stages increase, the policy shifts to the left because of the form of the cost function. The lowest value of cost obtained for both algorithms is for a state near 100, while the costs steadily increase for higher values of state. The proposed algorithm took 7800 seconds for 500 updates of the parameter vector on a Pentium III computer using code written in the C programming language.

For the $B = 50$ setting, Tables 1–3 show performance comparisons of the proposed algorithms for various metrics with mean and standard deviation taken over the (above-mentioned) 2×10^5 independent sample trajectories. Note that in the approximate DP algorithm, the transition probabilities using the method in [36] are easily computed in our setting. However, in most real-life scenarios, computing these probabilities would not be simple, and one may need to rely exclusively on simulation-based methods. For instance, as explained in

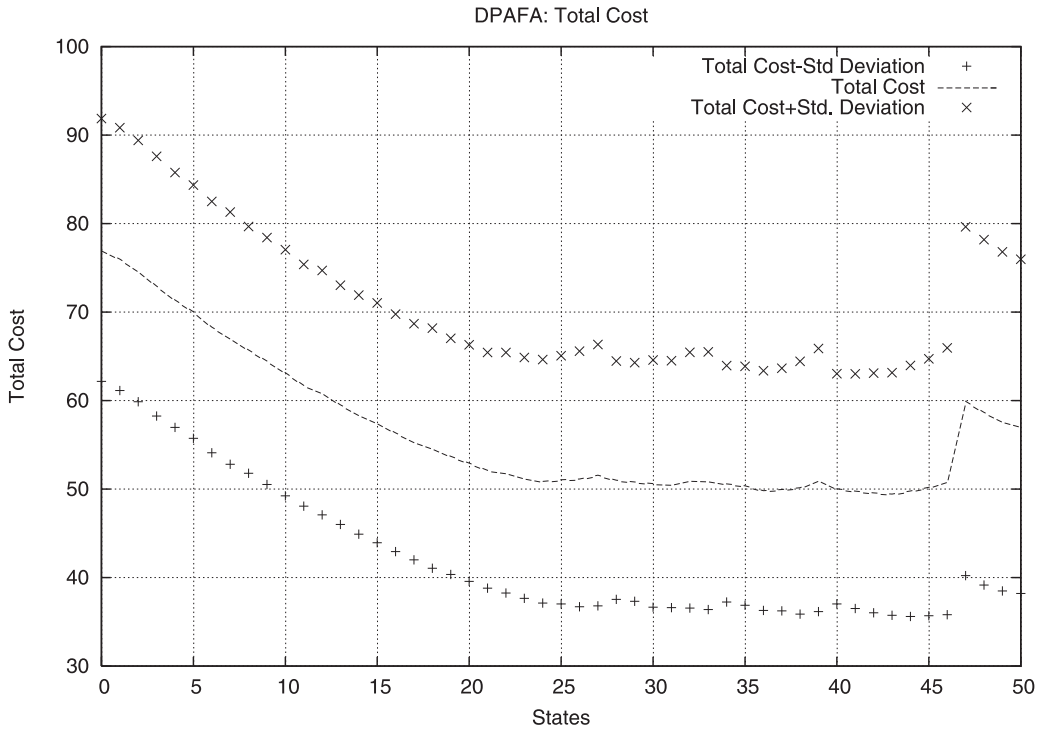

Figure 9. Optimal policy computed using Algorithm P-ACA

the remark following Algorithm ACA in Section 2.2, if one uses non-exponential service times, computing transition probabilities of the resulting Markov chain (under a given admissible policy) would not be straightforward. On the other hand, our simulation-based algorithms could also be applied in a straightforward manner in the latter scenario.

Table 2 shows the mean queue length $E(i_r)$ at instants $r = 2, 4, 6, 8$ and 10, with the corresponding standard deviation. With $\hat{T}_r$ defined as the 'target' state at instant r, Table 3 shows the probability $p_r = P(i_r = \hat{T}_r \pm 1)$ of the system being in states $\hat{T}_r \pm 1$ at the above values of r. Table 1 shows the mean one-stage cost $E(g_{r-1}(i_{r-1}, \mu_{r-1}(i_{r-1}), i_r)) = E(|i_r - \hat{T}_r|)$ incurred by the system during transition from i_{r-1} to i_r under action $\mu_{r-1}(i_{r-1})$.

Note that our goal here is to minimize the sum of these one-stage costs. As regards performance, we infer from Table 1 that our algorithm DPAFA outperforms SAMW even although the latter is better than RPAFA. The above conclusions can also be seen to hold from Table 3, where we compare the steady-state band probabilities $p_r = P(i_r = \hat{T}_r \pm 1)$ using the above algorithms. Since our objective is to minimize the expected absolute differences between states i_r and the target states $\hat{T}_r$, a better algorithm would be one that has a higher value of p_r. Here, DPAFA can also be seen to be better than SAMW.

Note also that our objective does not involve a minimization of the mean queue lengths themselves. Hence the mean queue length by itself (Table 2) is not a very meaningful performance metric here to compare the performance of the various algorithms. Nevertheless, we exhibit the mean queue length values in Table 2 for the sake of completeness.

It must be noted that our algorithm ACA is for compact action sets and as expected, shows the best results of all algorithms (see Tables 1 and 3) since it operates over a much wider range of actions. From the tables it can be seen that RPAFA does not show good performance. This happens because of the stochastic nature that becomes exemplified in the algorithm. Note that actions in RPAFA are chosen as per the randomized policies given by the algorithm. Ideally, the probability of selecting the best action must converge to one. However, what we observe is that these probabilities converge to somewhere around 0.8–0.9 in many cases. As a result, positive probabilities also become assigned to those actions that are not the best ones (using this algorithm). Thus, the optimal action, even if possessing the highest probability of selection, may not be selected each time and non-optimal actions may get selected. We observe that DPAFA shows better performance than RPAFA since it

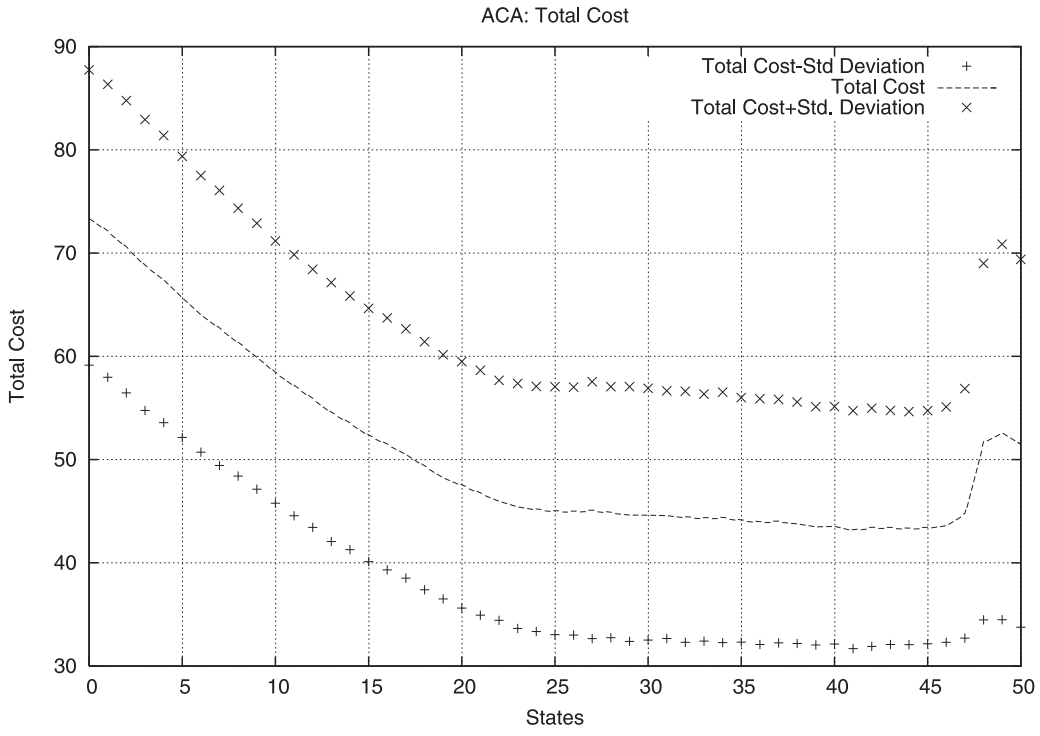

Figure 10. Comparison of the costs-to-go: P-ACA versus approximate DP

updates deterministic policies in the convex hull of feasible actions. Further, in the case of DPAFA, one does not require a 4−dimensional Hadamard perturbation as with RPAFA.

Note that the standard deviations in Table 3 can also be theoretically obtained (if p_r values are known) as follows. For a given r with $1 \leq r \leq T$, let $I_{\{i_r=\hat{T}_r\pm 1\}}$ be the indicator random variable taking a value 1 if $i_r = \hat{T}_r \pm 1$ and 0 otherwise. Now, if $p_r = E(I_{\{i_r=\hat{T}_r\pm 1\}}) = P(i_r = \hat{T}_r \pm 1)$, then the standard deviation σ_r is given by

$$\sigma_r = \sqrt{E(I^2_{\{i_r=\hat{T}_r\pm 1\}}) - E^2(I_{\{i_r=\hat{T}_r\pm 1\}})} = \sqrt{p_r(1-p_r)}.$$

It can be seen from the above that $\sigma_r > p_r$ if $p_r(1-p_r) > p_r^2$ or that $p_r(1-2p_r) > 0$. This will happen if $p_r > 0$ and $p_r < 1/2$, which is indeed true in all the cases in Table 3. This explains the reason for the standard deviation being higher than the mean in all the entries in Table 3.

5. Conclusions

We developed and proved convergence of certain actor-critic reinforcement learning algorithms for solving finite-horizon MDPs. We presented numerical experiments over a setting involving flow and congestion control in communication networks.

The algorithms consider two different settings: (a) finite state and compact action sets and (b) finite state and discrete action sets. One of the algorithms proposed for case (a) used a parameterized actor, thus reducing the memory requirement from at least $|S| \times T$ to only $K \times T$, where $K \ll |S|$, T is the horizon length and S is the state-space of the MDP. All our algorithms used variants of SPSA gradient estimates. In particular, the perturbation sequences for three of the proposed algorithms were derived from normalized Hadamard matrices and show fast convergence.

In order to further improve performance, efficient second-order simulation-based actor-critic algorithms that also estimate the Hessian in addition to the gradient could be developed [38, 39]. Outside of the actor-critic framework, it would be a good idea to try and trim the exponential storage requirement of the SAMW algorithm [37] as well as use a diminishing step-size to guide the recursion. Further, it is imperative to quantify (and bound) the number of sub-optimal actions taken by the simulation-based algorithm in order to have an idea of the optimization cost. This we intend to pursue in the framework of the SOFTMIX algorithm for the Stochastic Multi-armed Bandit [40].

6. Acknowledgements

This work was supported in part by Grants SR/S3/EE/43/2002-SERC-Engg and SR/S3/EECE/011/2007 from the Department of Science and Technology, Government of India.

7. References

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

[2] Bertsekas, D. 1995. *Dynamic Programming and Optimal Control, Volume I*. Belmont, MA: Athena Scientific.

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

[4] Bertsekas, D. and J. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

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

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

[7] Watkins, C. and P. Dayan. 1992. Q-learning. *Machine Learning* 8, 279–292.

[8] Barto, A., R. Sutton, and C. Anderson. 1983. Neuron-like elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13, 835–846.

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

[10] Konda, V. and J. Tsitsiklis. 2003. Actor-Critic Algorithms. *SIAM Journal on Control and Optimization* 42(4), 1143–1166.

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

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

[13] Spall, J. C. 1997. A one-measurement form of simultaneous perturbation stochastic approximation. *Automatica* 33(1), 109–112.

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

[15] Bhatnagar, S., E. Fernandez-Gaucherand, M. Fu, Y. He, and S. Marcus. 1999. A Markov decision process model for capacity expansion and allocation. *In Proceedings of 38th IEEE Conference on Decision and Control, Vol. 2, Phoenix, Arizona*, pp. 1380–1385.

[16] Chang, H., P. Fard, S. Marcus, and M. Shayman. 2003. Multitime Scale Markov Decision Processes. *IEEE Transactions on Automatic Control* 48(6), 976–987.

[17] Serin, Y. 1995. A nonlinear programming model for partially observed Markov decision processes: finite horizon case. *European Journal of Operational Research* 86, 549–564.

[18] Lin, A., J. Bean, and C. White, III. 2004. A hybrid genetic/optimization algorithm for finite horizon partially observed Markov decision processes. *INFORMS Journal on Computing* 16(1), 27–38.

[19] Zhang, C. and J. Baras. 2000. A hierarchical structure for finite horizon dynamic programming problems. *Technical Report TR2000-53, Institute for Systems Research, University of Maryland, USA*.

[20] Garcia, F. and S. M. Ndiaye. 1998. A learning rate analysis of reinforcement learning algorithms in finite-horizon. *In Proceedings of the Fifteenth International Conference on Machine Learning, Madison, USA*.

[21] Tsitsiklis, J. and B. Van Roy. 1999. Average Cost Temporal-Difference Learning. *Automatica* 35(11), 1799–1808.

[22] Parkes, D. C. and S. Singh. 2003. An MDP-Based Approach to Online Mechanism Design. *In Proceedings of 17th Annual Conference on Neural Information Processing Systems (NIPS'03)*.

[23] Parkes, D. C., S. Singh, and D. Yanovsky. 2004. Approximately Efficient Online Mechanism Design. *In Proceedings of 18th Annual Conference on Neural Information Processing Systems (NIPS'04)*.

[24] Chin, J. 2004. *Cisco Frame Relay Solutions Guide*. Cisco Press.

[25] Bhatnagar, S., M. Fu, S. Marcus, and P. Fard. 2001. Optimal structured feedback policies for ABR flow control using two-timescale SPSA. *IEEE/ACM Transactions on Networking* 9(4), 479–491.

[26] Sutton, R., D. McAllester, S. Singh, and Y. Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In* S. A. Solla and T. K. Leen and K.-R. Muller, (Ed.) *Advances in Neural Information Processing Systems–12*, pp. 1057–1063. Cambridge, MA: MIT Press.

[27] Marbach, P. and J. Tsitsiklis. 2001. Simulation-Based Optimization of Markov Reward Processes. *IEEE Transactions on Automatic Control* 46(2), 191–209.

[28] Boyan, J. 1999. Least-squares Temporal Difference Learning. *In Proceedings of Sixteenth ICML*, pp. 49–56.

[29] Abdulla, M. S. and S. Bhatnagar. 2007. Reinforcement learning based algorithms for average cost Markov decision processes. *Discrete Event Dynamic Systems: Theory and Applications* 17(1), 23–52.

[30] Konda, V. R. and J. N. Tsitsiklis. 2004. Convergence rate of linear two-time-scale stochastic approximation. *Annals Of Applied Probability* 14(2), 796–819.

[31] Kushner, H. and D. Clark. 1978. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. New York: Springer-Verlag.

[32] Kushner, H. and G. Yin. 1997. *Stochastic Approximation: Algorithms and Applications*. New York: Springer-Verlag.

[33] Neveu, J. 1975. *Discrete Parameter Martingales*. Amsterdam, the Netherlands: North Holland.

[34] M.W. Hirsch. 2003. Convergent activation dynamics in continuous time networks. *Neural Networks* 2, 1143–1166.

[35] Gerencser, L., S. Hill, and Z. Vago. 1999. Optimization over discrete sets via SPSA. *In Proceedings of the 38th IEEE Conference on Decision and Control-CDC99, Phoenix, Arizona, USA*, pp. 1791–1794.

[36] Ross, S. M. 2000. *Introduction to Probability Models, 7/e*. San Diego, CA: Academic Press.

[37] Chang, H.-S., M. C. Fu, and S. I. Marcus. 2007. An Asymptotically Efficient Algorithm for Finite Horizon Stochastic Dynamic Programming Problems. *IEEE Transactions on Automatic Control* 52(1), 89–94.

[38] 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.

[39] Bhatnagar, S. 2007. Adaptive Newton-Based Multivariate Smoothed Functional Algorithms for Simulation Optimization. *ACM Transactions on Modeling and Computer Simulation* 18(1), 2:1–2:35.

[40] Cesa-Bianchi, N. and P. Fischer. 1998. Finite-time regret bounds for the multi-armed bandit problem. *In Proceedings of 15th International Conference on Machine Learning (ICML)*.

Shalabh Bhatnagar *received his Masters and Ph.D degrees in Electrical Engineering from the Indian Institute of Science, Bangalore in 1992 and 1998 respectively. He is currently a Visiting Professor at the University of Alberta, Canada, until September 2009. His research interests are in areas of performance modeling, analysis, optimization and control of stochastic systems, in particular communication networks. He has published over 70 research papers in international journals and conferences. He is an Associate Editor of the IEEE Transactions on Automation Science and Engineering. He has received the MSR India Outstanding Young Faculty Award and the Young Scientist Award in Systems Theory from the Systems Society of India, both in 2007.*

Mohammed Shahid Abdulla *obtained his B.E. in Computer Engineering from Mangalore University in 2001. He further obtained his M.E. in Signal Processing from the Indian Institute Of Science, Bangalore in 2003 and finished his Ph.D. at the Department of Computer Science and Automation at the same institute in December 2007. He is currently a Researcher at General Motors India Science Lab in Bangalore, India.*