Simulation Based Algorithms for Markov Decision Processes and Stochastic Optimization

A Thesis
Submitted For the Degree of
Doctor of Philosophy
in the Faculty of Engineering

by
Mohammed Shahid Abdulla

Department of Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012
MAY 2008
To all whose help I’ve had
Acknowledgements

I thank Dr Shalabh Bhatnagar, my PhD advisor. In all my time as a student, I never had one idea that didn’t have his ear. I aspire to be able to bring the kind of rigour, commitment, and clarity to one’s work as he does.

I thank Professor M Narasimha Murty for suggesting pursuit of a PhD, discussing some applications of my research, and being generous in providing resources for the Stochastic Systems Lab. I am grateful to Professor Y Narahari for the many academic and co-curricular matters he has helped me in. I thank Dr James Spall of Johns Hopkins University for a clarification pertaining to Chapter 4.

My buddies over the years in IISc have my special gratitude: folks, I admire you and your company has made for the enjoyable time I’ve had. Dropping the honorifics that I use for some of them, my thanks to Babar Ahmad, Rizwan Ali, Avinash Achar, K Premkumar, MV Panduranga Rao, Sujit Kumar Chakrabarti, Siddhartha Tambat and Anindya Goswami among those currently at IISc. Fond memories remain of Rahul Jain, Gyanendra Kumar, Joel Pinto, Gururaj Puthraya, Arzad Alam Kherani and Dinesh Garg of the batches past. I’ve shared both laughs and doubts with my labmates Ranganath, Amrish Patel and Thirumaran - many thanks to them. Graduated students Asharaf, Ambedkar and Rahul Vaidya have my gratitude for keeping in touch. My thanks to Ms Lalitha, Ms Suguna and Mr George Fernandes of the CSA Office and IISc’s Deputy Registrar Mr K Panneer Selvam for going out of their way to help.

Umma, Uppa, Tahir and little Zulfath have been kind in taking a lenient view of my long absences and silences. Frankly, I’ve missed them a lot myself and their undiminished support has been crucial to this endeavour.
Abstract

In Chapter 2, we propose several two-timescale simulation-based actor-critic algorithms for solution of infinite horizon Markov Decision Processes (MDPs) with finite state-space under the average cost criterion. On the slower timescale, all the algorithms perform a gradient search over corresponding policy spaces using two different Simultaneous Perturbation Stochastic Approximation (SPSA) gradient estimates. On the faster timescale, the differential cost function corresponding to a given stationary policy is updated and averaged for enhanced performance. A proof of convergence to a locally optimal policy is presented. Next, a memory efficient implementation using a feature-vector representation of the state-space and TD(0) learning along the faster timescale is discussed. A three-timescale simulation based algorithm for solution of infinite horizon discounted-cost MDPs via the Value Iteration approach is also proposed. An approximation of the Dynamic Programming operator $T$ is applied to the value function iterates. A sketch of convergence explaining the dynamics of the algorithm using associated ODEs is presented. Numerical experiments on rate based flow control on a bottleneck node using a continuous-time queueing model are presented using the proposed algorithms.

Next, in Chapter 3, we develop three simulation-based algorithms for finite-horizon MDPs (FH-MDPs). The first algorithm is developed for finite state and compact action spaces while the other two are for finite state and finite action spaces. Convergence analysis is briefly sketched. We then concentrate on methods to mitigate the curse of dimensionality that affects FH-MDPs severely, as there is one probability transition matrix per stage. Two parametrized actor-critic algorithms for FH-MDPs with compact action sets are proposed, the ‘critic’ in both algorithms learning the policy gradient. We show
w.p 1 convergence to a set with the necessary condition for constrained optima. Further, a third algorithm for stochastic control of stopping time processes is presented. Numerical experiments with the proposed finite-horizon algorithms are shown for a problem of flow control in communication networks.

Towards stochastic optimization, in Chapter 4, we propose five algorithms which are variants of SPSA. The original one-measurement SPSA uses an estimate of the gradient of objective function $L$ containing an additional bias term not seen in two-measurement SPSA. We propose a one-measurement algorithm that eliminates this bias, and has asymptotic convergence properties making for easier comparison with the two-measurement SPSA. The algorithm, under certain conditions, outperforms both forms of SPSA with the only overhead being the storage of a single measurement. We also propose a similar algorithm that uses perturbations obtained from normalized Hadamard matrices. The convergence w.p. 1 of both algorithms is established. We extend measurement reuse to design three second-order SPSA algorithms, sketch the convergence analysis and present simulation results on an illustrative minimization problem. We then propose several stochastic approximation implementations for related algorithms in flow-control of communication networks, beginning with a discrete-time implementation of Kelly’s primal flow-control algorithm. Convergence with probability 1 is shown, even in the presence of communication delays and stochastic effects seen in link congestion indications. Two relevant enhancements are then pursued: $a$) an implementation of the primal algorithm using second-order information, and $b$) an implementation where edge-routers rectify misbehaving flows. Also, discrete-time implementations of Kelly’s dual algorithm and primal-dual algorithm are proposed. Simulation results $a$) verifying the proposed algorithms and, $b$) comparing stability properties with an algorithm in the literature are presented.
Following is a list of publications from this thesis:


Contents

Acknowledgements i
Abstract ii
Publications iv

1 Introduction 1
  1.1 Brief Explanation of Concepts ........................................ 1
    1.1.1 Markov Decision Processes ........................................ 1
    1.1.2 Need for Simulation-Based Methods ............................... 5
    1.1.3 On the Curse of Dimensionality ................................... 10
    1.1.4 Stochastic Approximation and Experimental Set-Up ............ 12
  1.2 Contributions .................................................................. 14

2 Reinforcement Learning Algorithms for Infinite Horizon MDPs 16
  2.1 Reinforcement Learning Based Algorithms For Average Cost MDPs .... 17
    2.1.1 Framework and General Algorithm .................................. 22
    2.1.2 Brief Outline of SPSA ............................................... 26
    2.1.3 Algorithms for Compact Action Sets ............................... 28
    2.1.4 Algorithms For Finite Action Sets using RSPs .................. 30
    2.1.5 Algorithms for Finite Action Sets using Deterministic Policies 31
    2.1.6 Convergence Analysis ................................................ 33
    2.1.7 A Variant of the TD(0) Algorithm ................................ 39
2.2 Solution of MDPs using Simulation-based Value Iteration

2.2.1 Algorithm

2.2.2 Outline of Proof

2.3 Numerical Experiments

2.3.1 Setting and Parameters

2.3.2 Simulation Results for ACA-\(r\), RPAFA-\(r\) and DPAFA-\(r\)

2.3.3 Simulation Results for Value-Iteration algorithm

3 Algorithms for Finite Horizon MDPs

3.1 Simulation–Based Optimization Algorithms for Finite Horizon MDPs

3.1.1 Framework and Notation

3.1.2 Finite Horizon Algorithm for Compact Action Sets

3.1.3 Finite Horizon Algorithms for Finite Action Sets

3.1.4 Convergence Analysis

3.2 Parametrized Actor-Critic Algorithms for FH-MDPs

3.2.1 Framework and Notation

3.2.2 Parametrized FH-MDP Algorithm 1

3.2.3 Parametrized FH-MDP Algorithm 2

3.2.4 Algorithm 3: Stopping Time Problem

3.3 Simulation Results

3.3.1 Look-up Table FH-MDP Algorithms

3.3.2 Parametrized FH-MDP Algorithms

4 Applications of Stochastic Approximation

4.1 SPSA Algorithms with Measurement Reuse

4.1.1 Algorithm SPSA2-1UR

4.1.2 Asymptotic Normality and Comparison

4.1.3 Algorithm SPSA2-1UH

4.1.4 Second-Order Algorithms

4.1.5 2SPSA-1UR
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Average Cost λ∗ as computed by the proposed algorithms</td>
<td>56</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison of DPAFA-r and RPAFA-r with Actor-Critic Policy Iteration</td>
<td>56</td>
</tr>
<tr>
<td>2.3</td>
<td>J∗ using Differential-Cost Approximation for B = 500</td>
<td>56</td>
</tr>
<tr>
<td>2.4</td>
<td>λ∗ using h-approximation for large B and K = 4</td>
<td>57</td>
</tr>
<tr>
<td>2.5</td>
<td>Comparison of Convergence using h-approximation for B = 50 and K = 4</td>
<td>57</td>
</tr>
<tr>
<td>3.1</td>
<td>observed E(i_r) for the proposed algorithms</td>
<td>105</td>
</tr>
<tr>
<td>3.2</td>
<td>Probabilities p_r = P(i_r = ˆT_r ± 1)</td>
<td>105</td>
</tr>
<tr>
<td>3.3</td>
<td>Mean one-stage costs E(</td>
<td>i_r − ˆT_r</td>
</tr>
<tr>
<td>3.4</td>
<td>Performance of FH-MDP algorithms</td>
<td>107</td>
</tr>
<tr>
<td>3.5</td>
<td>Performance of Stopping-time algorithm</td>
<td>107</td>
</tr>
<tr>
<td>4.1</td>
<td>2R Vs. 1UR: MSE and No. of Iterations</td>
<td>149</td>
</tr>
<tr>
<td>4.2</td>
<td>2H Vs. 1UH: MSE and No. of Iterations</td>
<td>150</td>
</tr>
<tr>
<td>4.3</td>
<td>1R Vs. 1UR: MSE and No. of Iterations</td>
<td>150</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison of 2SPSA-3UR</td>
<td>150</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison of 2SPSA-2UR</td>
<td>151</td>
</tr>
<tr>
<td>4.6</td>
<td>Comparison of 2SPSA-1UR</td>
<td>151</td>
</tr>
<tr>
<td>4.7</td>
<td>Performance of Primal Algorithm and variants</td>
<td>152</td>
</tr>
<tr>
<td>4.8</td>
<td>Performance of Dual and Primal-Dual Algorithms</td>
<td>152</td>
</tr>
<tr>
<td>4.9</td>
<td>Comparison with system analysed via DDEs</td>
<td>153</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Converged Policy computed by ACA-2</td>
<td>58</td>
</tr>
<tr>
<td>2.2</td>
<td>Convergence Behaviour for ACA-2</td>
<td>58</td>
</tr>
<tr>
<td>2.3</td>
<td>Randomized Stationary Policy computed using RP AFA-2</td>
<td>59</td>
</tr>
<tr>
<td>2.4</td>
<td>Convergence Behaviour for RP AFA-2</td>
<td>59</td>
</tr>
<tr>
<td>2.5</td>
<td>Policy computed using DP AFA-2</td>
<td>60</td>
</tr>
<tr>
<td>2.6</td>
<td>Optimal Policy for Source 1</td>
<td>60</td>
</tr>
<tr>
<td>2.7</td>
<td>Optimal Policy - Sum of Source Rates</td>
<td>61</td>
</tr>
<tr>
<td>2.8</td>
<td>Differential-Cost Function computed using ACA-2</td>
<td>61</td>
</tr>
<tr>
<td>2.9</td>
<td>Differential Cost Function computed using RP AFA-2</td>
<td>62</td>
</tr>
<tr>
<td>2.10</td>
<td>Differential Cost Function computed using DP AFA-2</td>
<td>62</td>
</tr>
<tr>
<td>2.11</td>
<td>Plots of (a) converged policies and (b) value functions</td>
<td>63</td>
</tr>
<tr>
<td>2.12</td>
<td>$V$ iterates from (a) classical VI and (b) modified TD(0)</td>
<td>63</td>
</tr>
<tr>
<td>3.1</td>
<td>Using Approximate DP: (a) Optimal Policy and (b) Finite-Horizon Cost</td>
<td>100</td>
</tr>
<tr>
<td>3.2</td>
<td>Converged Policy using Algorithm ACA for Compact Action setting</td>
<td>101</td>
</tr>
<tr>
<td>3.3</td>
<td>Converged Policy in Discrete Action setting (a) RP AFA and (b) DP AFA</td>
<td>101</td>
</tr>
<tr>
<td>3.4</td>
<td>Finite-Horizon Cost using Algorithm ACA for Compact Action setting</td>
<td>102</td>
</tr>
<tr>
<td>3.5</td>
<td>Finite-Horizon Cost in Discrete Action setting (a) RP AFA and (b) DP AFA</td>
<td>102</td>
</tr>
<tr>
<td>3.6</td>
<td>Optimal Policy computed using (a) Approximate DP (b) Algorithm 1</td>
<td>108</td>
</tr>
<tr>
<td>3.7</td>
<td>Comparison of the costs-to-go: Algorithm 1 vs. Approximate DP</td>
<td>108</td>
</tr>
<tr>
<td>3.8</td>
<td>Algorithm 3: (a) Optimal Policy Using VI (b) Cost Comparison with VI</td>
<td>109</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Brief Explanation of Concepts

1.1.1 Markov Decision Processes

Markov Decision Processes (MDPs) are a general framework for a large class of dynamic control problems under uncertainty. An MDP can be viewed as a state-valued stochastic process \( \{X_t\} \) that is in turn guided by a control process \( \{Z_t\} \) for \( t \in \mathbb{Z}_+ \cup \{0\} \), and follows the ‘controlled Markov property’

\[
P(X_{t+1} = x_{t+1}|\{X_k = x_k, Z_k = z_k\}^t_{k=0}) = P(X_{t+1} = x_{t+1}|X_t = x_t, Z_t = z_t), \tag{1.1}
\]

indicating dependence of the current transition (at instant \( t \)) on only the current state and action, \( X_t \) and \( Z_t \), respectively. Here, \( \{X_t\} \) is the trajectory of states seen by an observer while \( \{Z_t\} \) is the sequence of actions taken, with \( Z_t \) taken upon observing \( X_t \), at each instant \( t \). This mirrors the elementary (i.e., in the absence of \( Z_t \)) Markov Chain property. The controlled Markov property also holds in the case of non-autonomous systems with non-stationary transition probabilities. In the case of Finite-Horizon MDPs, we have in general,

\[
P(X_{t+1} = x_{t+1}|\{X_k = x_k, Z_t = z_k\}^t_{k=0}) = P_t(X_{t+1} = x_{t+1}|X_t = x_t, Z_t = z_t),
\]

indicating dependence of the current transition (at instant \( t \)) on only the current state and action, \( X_t \) and \( Z_t \), respectively. Here, \( \{X_t\} \) is the trajectory of states seen by an observer while \( \{Z_t\} \) is the sequence of actions taken, with \( Z_t \) taken upon observing \( X_t \), at each instant \( t \). This mirrors the elementary (i.e., in the absence of \( Z_t \)) Markov Chain property. The controlled Markov property also holds in the case of non-autonomous systems with non-stationary transition probabilities. In the case of Finite-Horizon MDPs, we have in general,
where the probability could depend on the ‘stage’ \( t \) that the process is currently in. Note that \( t < T \), where \( T \) is the ‘horizon’ of the MDP - indicating that \( X_T \) is the terminal state in the trajectory.

In any state \( s \in S \), action \( Z_t \) is chosen from the set \( C_s \) of feasible actions of states \( s \), i.e. for all \( s \in S \). Many times in the following, for ease of notation we shall assume that \( C_s \equiv C, \forall s \in S \). Compact action sets are considered to be of the form \( C_s = \prod_{i=1}^{N_s} [a^i_s, b^i_s] \), \( a^i_s, b^i_s \in \mathcal{R} \) with \( a^i_s < b^i_s \) (though more general sets can be accommodated) whilst finite action sets contain actions a.s.t. \(|C_s| < \infty\). Here \(|C_s|\) denotes the cardinality of the set \( C_s \).

There is a positive single-stage cost \( K(x_t, a, x_{t+1}) \) incurred upon taking action \( a \) in state \( x_t \) which could also, in general, depend on the next state \( x_{t+1} \). This is the case for infinite-horizon MDPs. For the finite-horizon case, the single-stage cost can be non-stationary, viz. \( K_t(x_t, a, x_{t+1}) \) in place of \( K(x_t, a, x_{t+1}) \). A policy \( \pi \) - or a policy iterate guided by index \( n \), i.e. \( \pi_n \) - indicates an action for each state \( s \) through its component \( \pi_n(s) \). If the action-set is compact, the policy \( \pi_n(x_t) \in C_{x_t} \), whereas in finite action-sets the method of taking actions may be dictated (in our treatment below) by Randomized Stationary Policies (RSPs) - or for finite-horizon MDPs, Randomized Policies (RPs). In RSPs, by abuse of notation, \( \pi_n(x_t) \) is a probability vector over \( C_{x_t} \) - thus \( Z_t \) would indicate a random variable with law \( \pi_n(x_t) \). In some cases, for finite action-sets, we shall also work with their closed convex-hulls that can be seen to be compact, and thus apply similar techniques as for compact action sets. This will be dealt with in the DPFA-k algorithms of Chapters 2 and 3. For such cases, the prescribed control \( \pi_n(s) \) belongs to this convex hull.

We consider three cost-criteria that are used to form the objective function in MDPs. We explain the discounted cost-to-go criterion first and will cover the other criteria after describing two fundamental algorithms to solve MDPs. For a discount factor \( 0 < \alpha < 1 \), the infinite-horizon discounted cost from state \( s \in S \) corresponding to the policy \( \pi_n(s) \) is given by \( V_{\pi_n}(s) \):

\[
V_{\pi_n}(s) := E\{ \sum_{t=0}^{\infty} \alpha^t K(X_t, \pi_n(X_t), X_{t+1})|X_0 = s} \]
where the expectation is over the joint law of \( \{X_t\}_{t \geq 0} \). The above is seen to satisfy the Poisson Equation (1.2) below. Let \( V^*(s) \) denote the optimal cost-to-go or value in state \( s \). Then \( V^*(s) = \inf_{\pi_n} V_{\pi_n}(s) \), where the infimum is taken over all admissible policies \( \pi_n \) i.e. those for which \( \pi_n(X_t) \in C_{X_t} \) and the controlled Markov property (1.1) is satisfied.

We motivate this thesis by presenting the two classical algorithms for computing optimal policy in MDPs. Assume an MDP with a finite state space \( S \) and, unless mentioned otherwise, a compact action set \( C \) for each state \( s \in S \). We assume that the resulting Markov chain under each admissible policy is irreducible. A more elaborate introduction on the same lines as here is in the textbook of (Chang et al., 2007a, Chapter 1).

1. Policy iteration is described as the process of iteratively finding the optimal value function or simply value function \( V^* : S \to \mathcal{R} \) as explained below. Let \( V_{\pi_n}(\cdot) \) be the value-function when \( \pi_n \) is the policy used. Then,

\[
V_{\pi_n}(s) = \sum_{r \in S} p(s, \pi_n(s), r) (K(s, \pi_n(s), r) + \alpha V_{\pi_n}(r)) \quad \forall s \in S, \tag{1.2}
\]

is first obtained as a solution to a system of linear equations for given policy update \( \pi_n \). Next, the policy is updated according to

\[
\pi_{n+1}(s) := \arg \min_{a \in C} \sum_{r \in S} p(s, a, r) (K(s, a, r) + \alpha V_{\pi_n}(r)), \tag{1.3}
\]

The procedure is repeated till the time convergence is achieved. Note here that in order to solve the system (1.2), the transition probabilities \( p(s, a, r) \) are assumed known. This algorithm is assured to converge in a finite number of steps to the least-cost optimal value function \( V^* \equiv V_{\pi^*} \) when the action-set \( C \) is finite, i.e. \( |C| < \infty \).

The Poisson equation (1.2) repeats in many forms for different cost criteria.

2. The value iteration algorithm, in contrast, is only asymptotically efficient and uses the single recursion:

\[
V_{n+1}(s) := \min_{a \in C} \sum_{r \in S} p(s, a, r) (K(s, a, r) + \alpha V_{n}(r)) \quad \forall s \in S, \tag{1.4}
\]
starting from a given $V_0(\cdot)$. However, its convergence to optimal value function $V^*$ occurs geometrically fast in the discount factor $\alpha$. The function $V^*$ satisfies the well-known *Bellman Equation* which one can infer by equating $V_{n+1} = V_n = V^*$:

$$V^*(s) := \min_{a \in C} \sum_{r \in S} p(s, a, r) (K(s, a, r) + \alpha V^*(r)) \; \forall s \in S.$$ 

One can see that the value function $V^*$ produced at the end of value iteration implicitly gives the optimal policy $\pi^*$.

We explain how policy iteration, which we use in the thesis for the two other remaining cost-criteria, can be modified. The infinite-horizon average cost $\lambda_{\pi_n}$ corresponding to the policy $\pi_n$ is defined for the entire MDP as:

$$\lambda_{\pi_n} := \lim_{T \to \infty} \frac{1}{T} \sum_{m=1}^{T} K(X_m, \pi_n(X_m), X_{m+1}).$$

Note that the limit above may not exist in general, in which case it is replaced with limsup. If, on the other hand (as is the case with our work), one assumes that the resulting Markov chain under any given stationary policy is ergodic, then the above limit will exist. The Poisson equation (1.2) differs somewhat for the average cost case and the corresponding value function is termed the differential cost function $h$:

$$h_{\pi_n}(i_0) + h_{\pi_n}(s) := \sum_{r \in S} p(s, \pi_n(s), r) (K(s, \pi_n(s), r) + h_{\pi_n}(r)) \; \forall s \in S$$

and the (1.3) above, with $h_{\pi_n}$ replacing $V_{\pi_n}$, is retained. An arbitrarily chosen reference state $i_0 \in S$ is used and one can show that $h_{\pi_n}(i_0) \to \lambda^*$ as $n \to \infty$, where $\lambda^*$ is the optimal average-cost that is associated with the optimal policy $\pi^*$, i.e., $\lambda^* = \lambda_{\pi^*}$.

For finite horizon MDPs, there is a deterministic termination instant, say, $T \in \mathbb{Z}_+$. The single-stage costs and policy $\pi_n$ can in general be stage-dependant, i.e., $K_l(s, \pi_n(l, s), r)$, $0 \leq l \leq T - 1$ in place of $K(s, \pi_n(l, s), r)$ earlier, and a terminal cost $K_T(s)$ when in state $s$ at instant $T$, in addition. Under policy $\pi_n \equiv \{\pi_n(l, s); 0 \leq l \leq T - 1; s \in S\}$, the value in
state $s$ has the expression $V_{\pi_n}(0, s) = E \left[ K_T(X_T) + \sum_{l=0}^{T-1} K_l(X_l, \pi_n(l, X_l), X_{l+1}) | X_0 = s \right]$, indicating the cost-to-go from stage 0 onwards. The following dynamic programming algorithm gives the optimal cost $V^*(s) = \inf_{\pi} V_{\pi}(0, s)$ as equal to $V(0, s)$ in the last step of the algorithm by proceeding backwards in time.

\begin{align*}
V(T, s) &= K_T(s), \\
V(l, s) &= \min_{a_l \in A_l(s)} E \left[ K_l(X_l, a_l, X_{l+1}) + V(l + 1, X_{l+1}) | X_l = s \right], \quad l = 0, 1, \ldots, T - 1.
\end{align*}

One can use a policy iteration procedure here as well. Let $V_{\pi_n}(T, s) := K_T(s)$. For $0 \leq l \leq T - 1$,

\begin{align*}
V_{\pi_n}(l, s) &:= \sum_{r \in S} p_l(s, \pi_n(s), r) \left( K_l(s, \pi_n(l, s), r) + V_{\pi_n}(l + 1, r) \right) \quad \forall s \in S, \quad (1.4) \\
\pi_{n+1}(l, s) &:= \arg \min_{a \in C} \sum_{r \in S} p_l(s, a, r) \left( K_l(s, a, r) + V_{\pi_n}(l + 1, r) \right). \quad (1.5)
\end{align*}

Here $V_{\pi_n}(l, s)$ should be interpreted as cost-to-go from the $l-$th stage of the process, when in state $s$, until termination when the policy used is $\pi_n$. Note that (1.4) is an assignment of the RHS to $V_{\pi_n}(l, s)$ and not a system of equations that (1.2) is.

\subsection*{1.1.2 Need for Simulation-Based Methods}

Simulation-based methods are needed since in most systems, information on transition probabilities $p(s, a, r)$ or $p_l(s, a, r)$ (as the case may be) is not available. In reinforcement learning, the self-tuning notion of adaptive systems in estimating these quantities separately is not adopted. Instead, learning (termed explicit expectation-formation) of the RHS of (1.2) is undertaken (cf. Borkar (2002)). This we perform by simulating multiple transitions from $s$ to a state $\eta(s, \pi_n(s)) \in S$, the second argument indicating that action $\pi_n(s)$ was taken.

Assume now that for a given $n$ in (1.2)-(1.3) we employ a sub-index $m$ to compute
expectations using the Monte Carlo method i.e., by taking samples

\[ K(s, \pi_n(s), \eta_{n,m}(s, \pi_n(s))) + V_{\pi_n}(\eta_{n,m}(s, \pi_n(s))). \]

This then is an asymptotic process and for each \( n \), the expectation is only seen as \( m \to \infty \). But the index \( n \) must itself iterate such that \( n \to \infty \), in order to compute the optimal policy. To overcome this problem, the two-timescale stochastic approximation technique (cf. Borkar (1997)) is used. Here, both the slower-timescale recursion (the policy-improvement step indexed by \( n \)) and the faster-timescale recursion (the expectation-formation step indexed by \( m \)) proceed in tandem via use of appropriate stepsizes. While not using separate stepsizes, the ‘optimistic TD(0)’ method of (Bertsekas and Tsitsiklis, 1996, §5.4) is another example of how policy improvement and approximate policy evaluation can proceed in tandem. Note that optimistic TD(0) assumes that the greedy action of (1.3) can be computed exactly, and is in that sense not a simulation-based algorithm.

In particular, the most important property that links diminishing stepsizes \( a(n) \) of \( n \) to \( b(m) \) of \( m \) is that \( a(n) = o(b(n)) \), or that \( \frac{a(n)}{b(n)} \to 0 \) as \( n \to \infty \). One can associate stepsize \( b(m) \) with the averaging process of \( m \), yet the current explanation doesn’t make it clear what stepsize \( a(n) \), guiding the slower recursion, to use. To see this, observe that for compact action-sets \( C \) a gradient search must be conducted separately in equation (1.3) and \( a(n) \) plays a part in guiding this minimization recursion. This gradient search we perform using the stochastic optimization algorithm called Simultaneous Perturbation Stochastic Approximation (SPSA). We adopt its many variants - deterministic-perturbations, two-measurement, and single-measurement - in the algorithms that we propose later in the thesis.

To establish the nomenclature, the recursion indexed by \( n \) is the ‘actor’, while \( m \) indexes the ‘critic’ recursion. It is true that each step in \( n \) can draw its average from only a finite \( m < L \) number of samples with ‘actor’ \( \pi_n \) fixed. However, this contributes to one of the attractive features of actor-critic algorithms: the reduction in variance in the critic-recursion due to accumulation of past measurements (cf. Konda and Tsitsiklis...
1.1 Brief Explanation of Concepts

(i.e. the measurements corresponding to past actor iterates \( \pi_{n-k} \), \( 1 \leq k \leq n-1 \). We note an exception here: in an algorithm proposed in Chapter 3, a reset to 0 affects the critic iterate in a manner that voids the variance-reduction foreseen in actor-critic methods by Konda and Tsitsiklis (2003), but does not otherwise impact the convergence of the algorithm.

Assuming that \( m \leq 1 \) for all \( n \), we can use the same index \( n \) and provide the generic form of all (actor-critic) algorithms that we propose:

\[
\begin{align*}
\pi_{n+1} & := \pi_n + a(n)G(\pi_n, V_n, \Delta_n) \\
V_{n+1} & := V_n + b(n)F(\pi_n, V_n, \eta_n),
\end{align*}
\]

with terms \( G(\cdot) \) and \( F(\cdot) \) being reinforcement terms and \( \pi_n, V_n \) replaced by their parametrized analogs in certain algorithms. The terms \( \Delta_n \) are the ‘simultaneous perturbations’ referred to in SPSA and may be generated randomly or by a deterministic method explained later. Note the absence of any MDP-related simulation in \( G(\pi_n, V_n, \Delta_n) \) whilst \( F(\pi_n, V_n, \eta_n) \) has such a contribution through \( \eta_n \). In particular, \( G(\pi_n, V_n, \Delta_n) \) will, in our algorithms, always be policy gradient estimates computed using SPSA.

Briefly introducing SPSA here will help in the later subsections: note that the two-measurement SPSA works as \( G_s(\pi_n, V_n, \Delta_n) \approx \frac{V_{+n}(s) - V_{-n}(s)}{2\delta \Delta_n(s)} \), where \( G_s(\cdot) \) is the \( s \)-th component of \( G \) and \( \delta > 0 \) is a small constant. The policy \( \pi_n^+ \) is a perturbed policy: \( \pi_n^+ = \pi_n + \delta \Delta_n \) with \( \pi_n^- \) asymmetrically defined. While exact \( V_{+n} \) cannot be evaluated due to lack of transition probability knowledge, the critic recursions supply the actor with iterates \( V^+(n) \) and \( V^-(n) \) that, as per the two-timescale update property, are a substitute for the exact values. One-measurement SPSA uses only one of these measurements: \( V^+(n) \). This thesis later proposes a modified form of this algorithm by using as numerator \( V^+(n) - V^+(n-1) \) (thus merely entailing storage of the measurement made at \( n-1 \) instead of obtaining a fresh estimate). However note that this modified algorithm is studied for the stochastic optimization problem and not for simulation-based MDP solutions, per se.
1.1 Brief Explanation of Concepts

Using transitions $\eta(s, \pi_n(s))$ alone are the hallmark of simulation-based MDP algorithms known as \textit{centralized learning} (cf. Borkar (2002)). Better-known alternatives simulate a \textit{trajectory} from $s$ (cf. Bertsekas and Tsitsiklis, 1996, Chap. 6)), with action prescribed by policy $\pi_n$ taken in each state $\eta_m(s, \pi_n)$, taking $\eta_0(s, \pi_n) = s$ for convention, seen onwards from $s$. Such a trajectory for infinite-horizon MDPs is also termed a \textit{single-endless} trajectory, and the methods dealing herein lose the advantages of parallelization, possible otherwise if the simulation model is known and replicable on multiple processors (this is implicitly assumed by, e.g., Konda and Borkar (1999)). It may be that there are natural initial states and simulating transitions from all states does not take this into account. However, note that use of the Poisson Equation (e.g. even in a trajectory method like Optimistic TD(1) of (Bertsekas and Tsitsiklis, 1996, §5.5)) requires a frequently-updated estimate of the cost-to-go from any possible state $\eta_m(s, \pi_n(s))$. While Konda and Borkar (1999) and Bhatnagar and Kumar (2004) do, textbook treatments of Bertsekas and Tsitsiklis (1996), Sutton and Barto (1998) do not concentrate much on the transitions-based simulation model. Similarly, the compact (non-discrete) action setting for $C$ is also not considered in these latter references.

Chang et al. (2007a) propose algorithms that are transitions-based but implicitly consider trajectories since there is a termination condition for the recursion that is satisfied when the end-of-horizon is reached. These algorithms also consider finite sampling budgets, and convergence depends on these budgets tending to $\infty$. Many algorithms of Chang et al. (2007a) are for finite-horizon MDPs, therefore involving finite-length trajectories. Such algorithms do yield transitions based-analogs, e.g. Abdulla and Bhatnagar (2007b) suggest a modification of one of the methods of Chang et al. (2007a) to make it transitions-based although this happens only when look-up tables storing estimates $V_n(s)$, $\forall s \in S$ are used.

In the most general case, the asynchronous policy iteration of (Bertsekas and Tsitsiklis, 1996, §2.2) has no simulation-based analog later in the textbook. It is true that trajectory-based algorithms can approximate infinite-horizon behaviour via ‘Parallel Rollout’ (cf. Chang et al. (2007a)) and ‘m-stage lookahead’ (cf. Bertsekas and Tsitsiklis (1996)), but
1.1 Brief Explanation of Concepts

This typically requires trajectory lengths to increase to $\infty$. Some features of trajectory-based methods may not be desirable. For instance, an algorithm that stores a counter in each state $s \in S$ indicating the number of times $s$ was visited during the trajectory simulation is proposed by Borkar (2002). This counter worsens the storage requirements, especially since single-endless trajectory based algorithms in the larger part of Bertsekas and Tsitsiklis (1996) are proposed to mitigate the 'curse of dimensionality' and reduce storage. While explaining the look-up table variants, additional per-state information in the form of step-sizes $\gamma_m(s)$ and eligibility vector $z_m(s)$ is used in the algorithms of (Bertsekas and Tsitsiklis, 1996, §5.3).

Also note that in the single-endless trajectory-based model, a global clock tick $n$ (or the index $n$) is assumed as being known since the algorithm performs an update (or an observation) at each state it visits. In contrast, in a parallelized asynchronous implementation, it is not reasonable to assume a global clock due to multiple processors updating the value-function estimates of each state. Thus, a transitions-based method (cf. Konda and Borkar (1999)) for solutions of MDPs would make better use of parallelization.

While the method of using lookup table iterates $J(n)$ in TD($\lambda$) is purely motivational (cf. (Bertsekas and Tsitsiklis, 1996, Chapter 5)), we note the possibility of constructing an entire (but variable-length) eligibility trace for a state $s$ in a transitions-based model. An eligibility trace in TD($\lambda$) is a scalar computed from temporal-difference values seen at all states in the trajectory till $s$. This can be done by storing the source of the latest incoming transition into $s$ and following backwards the chain of such transitions when an eligibility trace needs computing. We could thus have a second-line algorithm that aids the convergence of the proposed transition-based methods - we however do not pursue this here. Note that a suggested update scheme is similarly backwards in the order of transitions of (Bertsekas and Tsitsiklis, 1996, §5.5). For examples of trajectory-based algorithms: the method of $\lambda$–policy iteration of (Bertsekas and Tsitsiklis, 1996, §2.3) varies $\lambda$ such that the procedure with $\lambda = 1$ corresponds to policy iteration whilst $\lambda = 0$ matches value iteration. The Q-learning algorithm of (Bertsekas and Tsitsiklis, 1996,
§6.6), considered the simulation based analog of Value Iteration, also operates on a single endless-trajectory. Many simulation based algorithms, e.g. of Konda and Borkar (1999), (Bertsekas and Tsitsiklis, 1996, §6.6), and Borkar (2002) have to add ‘exploration’ noise separately, not leveraging the MDP’s state-transition variability.

The third chapter in this thesis proposes algorithms for finite-horizon MDPs (FH-MDPs) where the novelty lies in viewing (1.4)-(1.5) as a coupled optimization problem. Note also the connection of Finite-Horizon Markov Decision Processes as a special case of Gauss-Seidel (asynchronous) policy iteration (cf. Bertsekas and Tsitsiklis (1996)). We loosely try to motivate an analogy with the two-timescale stepsizes. Note that in the FH-MDP case, the ‘$n$—th recursion must wait till $m \to \infty$’ problem of Policy Iteration is now complicated - an extra ‘solution at stage $l$ must wait till stage $(l + 1)$ is solved’ requirement is now added. As the proofs in Chapter 3 indicate, the solutions of $l$—stage problems too can proceed in tandem.

1.1.3 On the Curse of Dimensionality

For a state-space with $|S| \gg 1$, there are clearly memory overheads in storing iterates $\pi_n$ and $V_n$ - this is known as the curse of dimensionality. The method of choice of (Bertsekas and Tsitsiklis, 1996, Chap. 6) is linear parametrization of the ‘critic’, viz. $V_n(s) \equiv \langle \mu_n, \phi(s) \rangle$, where the $\langle \cdot, \cdot \rangle$ indicates an inner-product, $\phi(s)$ is an immediately observable ‘feature vector’ of the state $s$, while $\mu_n$ is a coefficient vector. The work of Sutton et al. (1999) and Konda and Tsitsiklis (2003) also parametrizes the RSP ‘actor’ iterate $\pi_n$ using parameters, say, $\nu_n$ where the key point to note is the changing feature vectors of the ‘critic’ iterates: now $\phi(s, a, \nu_n)$ for $a \in C$. This method, however, is suited for finite action-sets only.

Our approach to mitigating the curse of dimensionality is covered in the later part of Chapter 3, where we do not estimate either the value-function $V_n$ or the $Q$—functions pertaining to policy iterate $\pi_n$. Instead the critic iterates correspond to estimates of the policy gradient. In particular, the projection of the policy-gradient estimate into the space
spanned by the actor’s feature vectors is considered. This retains the advantage of diminished variance mentioned by Konda and Tsitsiklis (2003) but cannot be conventionally termed actor-critic due to unchanging feature vectors. Due to the compact action set assumption the feature vector used in the critic is the same as in the actor recursion, in contrast to the special relation between the actor and critic parametrizations first proposed by Sutton et al. (1999). However, we consider only Finite-Horizon MDPs in the algorithm.

To an extent, the curse of dimensionality is voided by use of SPSA in all the algorithms of ours’ pertaining to MDPs. This is since in simulation based techniques, compact action-sets often represent a mitigation of the curse of dimensionality due to single iterate storage. The comparison between discrete action-sets and compact action-sets, though, is somewhat unequal. Dimension of a Randomized Stationary Policy (RSP) \( \pi_n \) (used by, e.g., Konda and Borkar (1999)) increases with cardinality \( |C| \) of the finite-action set whilst compact action-set policy iterates \( \pi_n \) can also be unwieldy in higher-dimension parameters, e.g. \( C \subset \mathbb{R}^{N_s} \) where \( N_s \gg 1 \). Yet, an \( N_s \)-element discretization for a scalar control would equal the complexity of an \( N_s \)-dimension control. To relate the two, storage for the policy iterate grows exponentially in the increase of dimension \( N_s \) but only linearly in the compact action-set case.

In lieu of RSPs, certain methods use \( Q \)-value iterates, or even empirical means approximating \( Q \)-values (as in bandit approaches adapted to MDPs, cf. Auer et al. (2002)). These have the problem of storing one advantage function per action, apart from some computational complexity in choosing the ‘best’ \( Q \)-value. For certain types of action-sets, the adaptation to a convex-hull SPSA method as proposed in DPAFA-\( k \) in Chapters 2 and 3 later represents an efficiency in this regard. In certain methods, the RSPs are parametrized using the Boltzmann-Gibbs parametrization, e.g. (Konda and Borkar 1999, Algorithm 3), which however is suboptimal due to the finite \( \beta_0 \) in the projection operator \( P_{\beta_0} \) used there.

To evaluate a policy under the constraint of linear parametrization, the sampling of an MDP should be according to the stationary probabilities imposed by policy iterate \( \pi_n \),
this is shown to be theoretically sound by Tsitsiklis and Van Roy (1997) (note that the ‘offline TD(0)’ of Bertsekas and Tsitsiklis, 1996, Chapter 5) does not, however, diverge from online-sampling). We nevertheless have encouraging results without adhering to this requirement, as seen in Chapter 2. An illustrative technique that avoids online sampling is the ‘iterative resampling’ of Bertsekas and Tsitsiklis, 1996, §5.2.2). We are able to obtain a proof for such a parametrization in Chapter 3, as we can simulate multiple trajectories of fixed length as in the LSTD(0) algorithm of Boyan (1999). With an increased memory requirement, this finite-length trajectory requirement can also be weakened - in that only transitions, in the same manner as look-up table algorithms, need to be simulated.

The amount of storage required for the cost-to-go iterates $V_n$ is also non-trivial, in that it is $O(|S|)$. We clarify the $O(|S|)$ here, used instead of the more justifiable $|S|$: note that two-measurement SPSA spawns two cost-to-go iterates $V^+(n)$ and $V^-(n)$. We shall utilize, in Chapter 3, the observation that for the compact action-set case policy $\pi_n \in \mathcal{R}$ (or each of its components, respectively) just as $V_n$ is and use the same parametrization there. Many sampling techniques usually associated with parametrized critics, e.g. TD($\lambda$), are oriented towards trajectory-based simulation. These are first introduced in their look-up table avatars (cf. Bertsekas and Tsitsiklis, 1996, Chapter 5)). However, even in these look-up table variants, the possibility of the system visiting only a small subset of states exists. This places a constraint on quantifying the rapid convergence of the algorithm, especially variants like first-visit TD(0) and many-visit TD(0). Further, the computational complexity of TD($\lambda$) or $Q-$learning’s implicit choice of action in each state $s$ (with exploratory noise) also plays a role. The RSP method avoids maximization of $Q-$values by letting an action be chosen by merely sampling the action-set $C_s$ using the policy iterate $\pi_n(s)$.

1.1.4 Stochastic Approximation and Experimental Set-Up

The Robbins-Munro stochastic approximation algorithm, in the setting of function root-finding, avoids making multiple, noisy function measurements at a single parameter setting. Adjustment for this lack of precision is made via a single, diminishing step-size like
1.1 Brief Explanation of Concepts

the \(a(n)\) or \(b(n)\) above in (1.6)-(1.7) (cf. Kushner and Yin, 1997, Chapter 1)). For the function minimization problem the Kiefer-Wolfowitz algorithm proposed estimating the gradient of the function by perturbing each of the \(N_s\) (recall that \(N_s\) is the dimension of \(C\)) components of the iterate separately and taking \(2N_s\) symmetric (or \(N_s+1\) one-sided) finite-difference measurements. In the case of Random Directions Kiefer-Wolfowitz (RDKW), only one randomly picked direction is considered at each iteration (cf. Kushner and Yin, 1997, Chapter 1)). The SPSA algorithm matches RDKW in that at each step the perturbation \(\delta \Delta_n\) is used to make only 2 measurements. We notice here that some form of memory is necessary to store the perturbations \(\Delta\), although it could even be a word made of 0/1−bits indicating perturbations of the type \(\Delta_n(s) \in \{+1, -1\}\). We refer to the book of Spall (2003) that covers all variants of SPSA, plus many other algorithms in the stochastic optimization space. The computational simplicity of SPSA makes it favourable over higher-end global-optimization algorithms like the recent Model-Reference adaptive search of Hu et al. (2007). One form of SPSA uses a single measurement at each iteration to infer the gradient but is susceptible to bias - especially when used in tandem with a diminishing weight \(\delta_k \to 0\) in lieu of the constant scale factor \(\delta\). An algorithm with perturbations \(\Delta_n\) drawn from normalized Hadamard matrices (cf. Bhatnagar et al. (2003)) was proposed to alleviate this bias. We propose in Chapter 4 an algorithm that re-uses a past measurement to repair this bias and show conditions under which the expected iterations to attain a certain Mean Square Error (MSE) level are lesser.

We note that in Chapters 2 and 3 our experimental settings involved congestion control at a single bottleneck node with simulated Poisson arrivals whose rates could be controlled. In particular, rate \(\pi^*(s)\) corresponded to the recommended rate of arrivals if queue-length at the node is \(s\). In the second part of Chapter 4, while we do not cast the queue-process as an MDP as in Chapters 2 and 3, we generalize the problem to a multiple-source, multiple-link network flow control problem as in the Kelly-Maulloo-Tan (KMT) primal algorithm of Kelly et al. (1998). We use a recursion known as asynchronous stochastic approximation, to tackle the issue of robustness to link-source communication delays in a communication network. The KMT algorithm is distributed, in that no source acquires
knowledge of other competing sources’ sending-rates. Another advantage is that every link only measures the aggregate flow through it, not having to separately measure or charge the many sources that use it.

An algorithm with rate-allocations converging fastest to the equilibrium is attractive, and towards this end we propose a second-order algorithm. A modification of the KMT algorithm with the same aim has been proposed earlier by Athuraliya and Low (2000) but with a different scale-factor technique and in the original continuous-time framework. The second-order estimates of gradient in the proposed algorithm are, incidentally, computed using the one-measurement variant of SPSA we explained earlier. Also attracting recent research are solutions to correct ‘denial-of-service’ attacks where a malevolent source misrepresents its utility to the network designer, while still using the true utility function in the KMT rate-updation module at its end (cf. Fan et al. (2005)). For this, an ‘edge-router’ (or the ISP) observes the sending rate and adds a penalty to the original usage-charges so that eventually the rate allocation stabilizes in line with the announced utility. While the KMT algorithm is the Primal algorithm for flow control, two other discrete-time implementations proposed are for the Dual and Primal-Dual algorithms.

1.2 Contributions

We enumerate the contributions of this thesis:

1. We propose and prove convergence of simulation-based algorithms for infinite horizon Markov Decision Processes (MDPs) with the average-cost criterion and compact action-sets. Adapting the gradient search method used for compact action-sets, we also propose variants of these algorithms for finite action-set MDPs under two scenarios - a) when search in the space of Randomized Stationary Policies is conducted and b) the same is performed directly over the closed convex-hull of these action-sets. The proposed algorithms are all analogs of Policy Iteration. However, the simulation-only requirement (in particular, using transitions alone) implies that
1.2 Contributions

1. Convergence to an optimal policy is asymptotic, unlike policy iteration. This behaviour of asymptotic convergence is harnessed to propose a simulation-based analog of Value-Iteration also.

2. We propose and prove convergence of simulation-based algorithms for finite-horizon MDPs with compact action-sets. We extend the algorithm to finite action-sets in like manner as the average-cost problem of 1. above. Later, we tackle the curse-of-dimensionality that affects finite-horizon MDPs particularly severely, by proposing two algorithms that operate in the space of linear parametrizations of both policy and policy-gradient iterates. These algorithms have trade-offs in memory and computation, with one a trajectory-based algorithm while the other a slightly more memory-intensive transitions-only analog. We have shown the w.p. 1 convergence of these latter algorithms to a constrained optima, which we believe are among only a few results showing convergence of sparse representations of MDPs.

3. Simultaneous Perturbation Stochastic Approximation (SPSA) was the technique used for policy-gradient search in all settings of 1. and 2. above. We propose an algorithm to improve SPSA’s performance, specifically of the one-measurement variant used in 2. above. The covariance matrix for the iterate process is identified and seen to be competitive with one-measurement SPSA. Cases are identified where the proposed variant can outperform the established two-measurement SPSA version. Next, the congestion-control example of 1. and 2. is considered to propose an asynchronous stochastic approximation based discretization of the Kelly-Maulloo-Tan (KMT) primal algorithm for flow control which, unlike the original, is made robust to link-source communication delays. We further propose a second-order implementation of KMT for faster convergence to the equilibrium allocation. Also, an analogous algorithm for edge-router rectification of misbehaving network flows is proposed. Here, a penalty is levied upon sources sending at higher-than-allocated rates into the network. Further, algorithms representing the Dual and Primal-Dual forms of KMT are also proposed.
Chapter 2

Reinforcement Learning Algorithms for Infinite Horizon MDPs

Abstract

This chapter first proposes, in §2.1, several two-timescale simulation-based actor-critic algorithms for solution of infinite horizon Markov Decision Processes (MDPs) with finite state-space under the average cost criterion. Two of the algorithms are for the compact (non-discrete) action setting while the rest are for finite-action spaces. On the slower timescale, all the algorithms perform a gradient search over corresponding policy spaces using two different Simultaneous Perturbation Stochastic Approximation (SPSA) gradient estimates. On the faster timescale, the differential cost function corresponding to a given stationary policy is updated and averaged for enhanced performance. A proof of convergence to a locally optimal policy is presented. Next, a memory efficient implementation using a feature-vector representation of the state-space and TD(0) learning along the faster timescale is discussed. The TD(0) algorithm does not follow an on-line sampling of states but is observed to do well on our setting.

We further propose, in §2.2, a three-timescale simulation based algorithm for solution of infinite horizon discounted-cost MDPs via the Value Iteration approach. An approximation of the Dynamic Programming operator $T$ is applied to the value function iterates.
This ‘approximate’ operator is implemented using three timescales, the slowest of which updates the value function iterates. On the middle timescale we perform a gradient search over the feasible action set of each state using SPSA gradient estimates, thus finding the minimizing action in $T$. On the fastest timescale, the ‘critic’ estimates, over which the gradient search is performed, are obtained. A sketch of convergence explaining the dynamics of the algorithm using associated ODEs is also presented.

In §2.3, numerical experiments on rate based flow control on a bottleneck node using a continuous-time queueing model are presented using the proposed algorithms of §2.1. We show performance comparisons of our algorithms with the two-timescale actor-critic algorithms of Konda and Borkar (1999). Our algorithms exhibit more than an order of magnitude better performance over those of Konda and Borkar (1999). For the simulation-based value iteration algorithm of §2.2, experiments on the same model are performed. The results obtained are verified against classical value iteration where the feasible set is suitably discretized. Over such a discretized setting, a variant of the algorithm of Van Roy et al. (1997) is compared and the proposed algorithm is found to converge faster.

### 2.1 Reinforcement Learning Based Algorithms For Average Cost MDPs

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

Policy iteration is one of the classical methods for solving MDPs (cf. (Bertsekas, 1995, section 8.3)) that is performed using two loops - a policy update on the outer loop being performed once the inner loop that computes the stationary value function for a given policy estimate has converged. The outer loop’s ‘wait’ for convergence of the inner loop is avoided when two timescale stochastic approximation is used. In such a setting, the policy evaluation recursions proceed on a faster timescale whereas the policy updates, using the above evaluation estimates, are performed on a slower timescale. A simulation based, two timescale, approach for policy iteration is considered by Konda and Borkar (1999) for MDPs with finite state and finite action spaces. Konda and Tsitsiklis (2003) parametrize the value function using a linear approximation architecture and two timescale algorithms are used where on the faster scale, a temporal difference (TD) type learning algorithm (cf. Tsitsiklis and Van Roy (1999a)) is used and on the slower scale, an approximate gradient search is performed.

For noisy measurement based parameter optimization, a gradient descent stochastic approximation algorithm known as simultaneous perturbation stochastic approximation (SPSA) was proposed by Spall (1992) that uses only two cost function measurements for any \(N\)-dimensional parameter vector. Further, Spall (1997) proposed a similar method that used only one measurement. The algorithm of Spall (1992) has been seen to perform well in a variety of settings by several authors. The algorithm of Spall (1997), however, does not perform as well (as Spall (1992)) because of the presence of additional bias terms in its gradient estimate. The algorithms of Spall (1992) and Spall (1997) perturb the parameter vector randomly in all components by using independent and identically distributed (i.i.d.), symmetric mean-zero random variables. Bhatnagar et al. (2003) base the perturbations on certain deterministic sequences for two-timescale SPSA algorithms.
were proposed and a simulation based optimization setting was considered. It was observed by Bhatnagar et al. (2003) that algorithms that use deterministic perturbations performed better than those that use randomized perturbations on the settings considered therein. In particular, the improvement in performance of one-simulation algorithms that use certain normalized Hadamard matrix based perturbations, over those that use randomized perturbations was found to be significant. The SPSA approach to gradient approximation has been used by Bhatnagar and Kumar (2004) to compute the optimal policy in infinite-horizon discounted-cost MDPs and by Bhatnagar and Abdulla (2007) for finding the optimal policy in the case of MDPs under the finite horizon total cost criterion.

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

While the example that we consider is one of flow control, other areas of communication networks also employ the average cost criterion in applications such as call admission control and routing, see Marbach et al. (2000). Obtaining good strategies for flow control and admission control are important as these affect the throughput of a general communication network. A survey of the application of MDPs to communication networks
(though not necessarily restricted to the long-run average cost criterion) is Altman (2001).

Our work addresses only the single controller scenario, whilst Altman (2001) covers many cases of the application of MDP models to distributed team and competitive game tasks. Inventory control is another area where use of the average-cost criterion is made (cf. He et al. (2000)). The reference Marbach and Tsitsiklis (2001) adopts a neuro-dynamic programming framework that is different from the look-up table approach that we employ in all our algorithms save one - the parametrized actor-critic algorithm. This latter algorithm that we propose uses a ‘feature extraction of the state space S, which is also a technique employed in Marbach and Tsitsiklis (2001). Further, both the algorithms of He et al. (2000) and Marbach and Tsitsiklis (2001) require simulation of the MDP trajectory: a single endless trajectory in the former while multiple finite-length trajectories in the latter. This is unlike the proposed algorithms which require repeated simulations of a single transition from every state \(i \in S\). Finally, unlike both these methods, the algorithms we propose are actor-critic algorithms.

Work in Ormoneit and Glynn (2002) proposes a solution where a trajectory of length \(m\) is used. We make two observations about this method. Firstly, while value-function storage is reduced by using estimates only at vicinity states \(z_s\), the policy \(\mu\) still requires storage for any arbitrary state \(x\) (cf. eq. (20) in Ormoneit and Glynn (2002)). Neither is the storage of policy \(\mu\) clear from an implementation for optimal portfolio choice problem proposed in Ormoneit and Glynn (2001). Secondly, the algorithm in Ormoneit and Glynn (2002) is not ‘online’ in a different sense, i.e. one freezes the length \(m\) of the trajectory before starting the algorithm. This qualitatively differs from the asymptotic behaviour (as \(n \to \infty\)) of the proposed algorithms. Another example of the former, which is asymptotically efficient in theory but a budget of \(m\) transitions must be chosen in advance, is in Chang et al. (2007b).

In problems such as capacity allocation in semiconductor manufacturing, the focus is largely on a finite time horizon. However, the motivation to extend this to the infinite-horizon average-cost scenario exists for two natural reasons: (a) the always-on nature of expensive manufacturing plants (requiring optimization over an infinite horizon), and (b)
the concern for optimal throughput (resulting in adoption of the average-cost criterion). A combination of a high-level infinite-horizon problem and a lower-level finite-horizon task is seen in Panigrahi and Bhatnagar (2004), where the upper tier process can be adapted to average cost criteria (due to the two reasons mentioned above) whilst the lower level task is a finite-horizon MDP (that takes into account, say, delivery deadlines).

Due to the ‘curse of dimensionality’, look-up table methods like ours do not scale well since the associated performance and policy vectors, $h$ and $\pi$, respectively, have sizes proportional to $|S|$. Yet they are advantageous since in many settings where simulation-based algorithms are employed, instead of the dynamic programming operator of Bertsekas (1995), one utilizes a computer simulation of the model. This would mean that one simulates individual transitions from any $i \in S$: this fact being naturally used in look-up table algorithms. Also, such transitions can be simulated in parallel we shall deal with a typical concern arising from such an implementation in the following.

The critic used is an estimate of the differential cost function $h(\cdot)$. In the general case, we perform updation of the critic by simulating transitions out of every state in the MDP state-space $S$. The controls belong to the control space $C$, a possibly infinite set, and are specified by the present iterate of the (parameterized) policy $\pi_n$. Such an implementation of the critic is amenable to parallelization when multiple processors are available, each capable of simulating the system transitions. Though the critic updates of Konda and Borkar (1999) are performed synchronously, an associated stability test to make the critic-updates asynchronous was suggested by Borkar and Meyn (2000), and that permits therefore the above-mentioned parallelization. The actor updates in the algorithms proposed by Konda and Borkar (1999) converge to the optimal policy using stochastic approximation with ‘reinforcement-signals’ of varied types. All these signals require actor-specific simulation to be performed that results in a computational effort of $O(|C|)$ in each update step of the actor. In contrast, we employ SPSA estimates of the gradient (as do Bhatnagar et al. (2003), Bhatnagar and Kumar (2004) and Bhatnagar and Abdulla (2007)) that do not cause any simulation load specific to the actor. Thus the processors responsible for updating the actor parameters here need not
simulate any transitions of the system. This results in a significant reduction in the overall computational effort (as is also seen from our experiments) in the case of our algorithms over those of Konda and Borkar (1999).

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

The rest of the section is organized as follows: §2.1.1 identifies the setting, notation and a generic form of the proposed algorithms. In §2.1.2 the Hadamard matrix based construction of deterministic perturbation sequences is presented. The algorithms are proposed in §2.1.3, §2.1.4 and §2.1.5 respectively, for which the convergence analysis is shown in §2.1.6. The linear approximation method (above) is described in §2.1.7 while §2.3 provides the numerical results in the setting of flow control in communication networks.

### 2.1.1 Framework and General Algorithm

We consider an MDP $\{X_t, t = 0, 1, \ldots\}$ where decisions are made at instants $t = 0, 1, \ldots$, using an associated control-valued process $\{Z_t\}$. Suppose $S \equiv \{1, 2, \ldots, s\}$ is the (finite) state space and $C$ is the control space. Suppose also that $U(i) \subseteq C$ is the set of all feasible controls in state $i$. Let $p(i, u, j), i, j \in S, u \in U(i)$ be the transition probabilities associated with this MDP. An admissible policy $\mu = \{\mu_0, \mu_1, \mu_2, \ldots\}$ with $\mu_t : S \mapsto C$ is one for which $\mu_t \in U(i), \forall t \in \{0, 1, \ldots\}, \forall i \in S$. We call $\mu = \{\mu_0, \mu_1, \mu_2, \ldots\}$ a stationary policy when $\mu_t = \pi, \forall t \in \{0, 1, \ldots\}$, i.e., if the policy is time/stage invariant. For simplicity, we denote using $\pi = (\pi_i, i \in S)^T$, a stationary policy. Suppose $K(i_t, u_t, i_{t+1})$ denotes the one-step transition cost when the current state is $i_t \in S$, the action chosen is $u_t \in U(i_t)$ and the next state is $i_{t+1} \in S$. The aim here is to find an admissible policy $\pi$ that minimizes the
2.1 Reinforcement Learning Based Algorithms For Average Cost MDPs

associated infinite horizon average cost \( \lambda_{\pi} \), where

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

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

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

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

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

(2.1)

when action \( \pi_i \) is used in state \( i \). A unique solution for (2.1) can be obtained if we set \( h_{\pi}(i_0) = \lambda_{\pi} \) for a prescribed \( i_0 \in S \) that we call as a reference state. Suppose we define

\[
G_{\pi}(i, h) = \sum_{j=1}^{s} p(i, \pi_i, j)(K(i, \pi_i, j) + h(j)),
\]

(2.2)

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

We consider two types of stationary policies \( \pi \) viz., randomized and deterministic. For a system operating under a deterministic policy \( \pi \), the action \( \pi_i \in U(i) \) is applied when in state \( i \). It is for such systems that equation (2.1) holds. In contrast, a randomized stationary policy (RSP) \( \pi \) is denoted (via abuse of notation) as the vector \( \pi = (\pi_i, \forall i \in S)^T \) with each \( \pi_i = (\pi_{ij}, 0 \leq j \leq N_i) \), where \( \pi_{ij} \) is the probability of picking action \( u_{ij} \in U(i) \).
2.1 Reinforcement Learning Based Algorithms For Average Cost MDPs

Here \( U(i) \) is a finite set with \((N_i + 1)\) elements \( \{u_i^0, u_i^1, \ldots, u_i^{N_i}\} \). In practice, an RSP is applied by generating, when in state \( i \), a \( U(i) \)-valued random variable \( \xi(i, \pi_i) \) distributed according to the probability mass function (p.m.f.) \( \pi_i \). Note that to generate \( \xi(i, \pi_i) \), only \( N_i \) of the \((N_i + 1)\) available weights \( \pi_j^i \) are sufficient. Thus, the vector \( \hat{\pi}_i = (\pi_1^i, \pi_2^i, \ldots, \pi_{N_i}^i) \), \( \forall i \in S \), is sufficient to denote the ‘control’ \( \pi_i \), since \( \pi_i^0 = 1 - \sum_{j=1}^{N_i} \pi_j^i \).

For the sake of uniformity in notation across all policies, we assume that \( \xi(i, \pi_i) \) is also used when the deterministic policy \( \pi \) is applied, except that in such a case \( \xi(i, \pi_i) = \pi_i \).

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

For the reasons explained above, the analogous Poisson equation for an RSP \( \pi \) differs slightly from (2.1), and is as follows:

\[
\lambda_{\pi} + h_{\pi}(i) = \sum_{j=1}^{s} \sum_{k=0}^{N_i} \pi_j^k p(i, u_j^k, j) \left(K(i, u_j^k, j) + h_{\pi}(j)\right), \forall i \in S. \tag{2.3}
\]

As in (2.1), a unique solution for (2.3) can be obtained if we set \( h_{\pi}(i_0) = \lambda_{\pi} \) for a prescribed \( i_0 \in S \). Further, \( G_{\pi}(i, h_{\pi}) \) can also be analogously defined.

We explain the mechanism used in the coupled stochastic recursions that we employ. On the faster timescale, our algorithms use either one or two simulations that are identified by the index \( r \) taking values in the sets \( \{2\} \) or \( \{1, 2\} \), respectively. We use the generic \( r \) to qualify statements that hold for both cases above. At the end of a given \( L \) steps of each simulation, we update \( \pi_i \) once and denote by \( \pi_i(n) \) the \( n \)-th update of \( \pi_i \). Here \( \pi_i(n) \) in general is a vector with \( N_i \) elements, viz. \( \pi_i(n) = (\pi_1^i(n), \ldots, \pi_{N_i}^i(n))^T \). Thus, the policy
\[ \pi(n) = (\pi_1(n), \pi_2(n), ..., \pi_s(n))^T \] is a column vector of size \( \sum_{i=1}^{s} N_i \). The two simulations mentioned above correspond to the system operating under two perturbed policies, \( \pi^1(n) \) and \( \pi^2(n) \), respectively. The differential-cost estimates, \( h_i^1(nL) \) and \( h_i^2(nL) \), of \( h_{\pi^1(n)}(i) \) and \( h_{\pi^2(n)}(i) \) respectively, (see (2.5) below) corresponding to the above perturbed policies are computed in the course of updates \( (n - 1)L \) to \( nL \) of the two simulations. We denote the general policy-gradient estimate as \( \tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = (\tilde{\nabla}^j G_{\pi(n)}(i, h_{\pi(n)}), 1 \leq j \leq N_i)^T \). Here, \( \tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) \) stands for the \( N_i \)-element vector gradient estimate of \( G_{\pi(n)}(i, h_{\pi(n)}) \) w.r.t. \( \pi_i(n) \). The policy gradient estimate depends on the ‘critic’ estimates \( h_i^r(nL) \). In particular, we propose forms of \( \tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) \) that use either \( h_i^2(nL) \) or both \( h_i^1(nL) \) and \( h_i^2(nL) \), respectively (see (2.11)-(2.14) below).

A projection operator \( P_i(\cdot) \) is used to ensure that the updated actions as given by the algorithms remain feasible. We use perturbed policies \( \pi'(n) \), obtained from \( \pi(n) \) (explained later, see (2.10)), for driving recursions (2.5) below.

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

\[ \pi_i(n + 1) = P_i \left( \pi_i(n) - c(n) \tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) \right). \] (2.4)

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

\[ h_i^r(nL + m + 1) = (1 - b(n))h_i^r(nL + m) \\
+ b(n)(K(i, \xi_{nL+m}^r(i, \pi_i^r(n)), \eta_{nL+m}^r(i, \xi_{nL+m}^r(i, \pi_i^r(n)))) \\
- h_i^r(nL + m) + h_{\eta_{nL+m}^r(i, \xi_{nL+m}^r(i, \pi_i^r(n)))}(nL + m)). \] (2.5)

The precise relationship between \( \tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) \) and \( h_i^r(nL) \) will be made clear as the algorithms are introduced in §2.1.3. It was observed by Bhatnagar et al. (2001a) that SPSA based algorithms in the setting of simulation optimization improved performance when an \( L \)-step additional averaging as above is used. The value of \( L \) can be chosen arbitrarily in practice. We use \( L = 100 \) in our experiments. Note that recursions
in (2.5) are common to all algorithms. These are similar to the adaptive-critic updates in the average-cost, finite-action setting of (Konda and Borkar, 1999, (3.5)), except that the above $L$-step averaging is not present in the algorithms of Konda and Borkar (1999).

All six proposed versions of algorithm (2.4)-(2.5) have differences that pertain to the dissimilarities in the structure of $U(i)$ or the gradient estimate $\tilde{\nabla_i} G_{\pi(n)}(i, h_{\pi(n)})$ of (2.4).

The requirements on the step-sizes in (2.4)-(2.5) are as follows:

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

$$\sum_n b(n) = \sum_n c(n) = \infty, \quad \sum_n b(n)^2, \sum_n c(n)^2 < \infty,$$

(2.6)

and

$$c(n) = o(b(n)),$$

(2.7)

respectively.

The extra averaging of the recursion (2.5) over $L$ iterations, besides being beneficial for convergence (cf. Bhatnagar and Kumar (2004) and Bhatnagar et al. (2003)), has an added relevance to any asynchronous implementation involving multiple parallel processors each updating the differential cost estimate for a given state. Note that it is sufficient if we choose $L$ such that $L > 2D$ where $D$ is the upper bound on inter-processor communication delay (cf. Konda and Borkar, 1999, Assumption A5)), since in the recursion in (2.4), at iteration $n$, the ‘current’ policy gradient estimate $\tilde{\nabla_i} G_{\pi(n)}(i, h_{\pi(n)})$ can still be used rather than the need to use any previous estimates $\tilde{\nabla_i} G_{\pi(m)}(i, h_{\pi(m)})$, $0 \leq m < n$.

### 2.1.2 Brief Outline of SPSA

Before we proceed further, we first motivate the use of SPSA based gradient estimates. 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 $F(\theta) = E[f(\theta, \xi_n)]$. The expectation
above is taken w.r.t. the common distribution of $\xi_n$, $n \geq 0$. 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, $\pm 1$-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 $n$th update of $\theta$. 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 (cf. Spall (1992)):

$$\tilde{\nabla}_i F(\theta(n)) = \left( \frac{f(\theta(n) + \delta_n \Delta(n), \xi^+_n) - f(\theta(n) - \delta_n \Delta(n), \xi^-_n)}{2\delta_n \Delta_i(n)} \right) . \tag{2.8}$$

This form of SPSA will be used in algorithms of §3.2 later.

Further, the one-measurement SPSA gradient estimate of Spall (1997) has the form in (2.9). Note that only one measurement (corresponding to $\theta(n) + \delta_n \Delta(n)$) is required here.

$$\tilde{\nabla}_i F(\theta(n)) = \frac{f(\theta(n) + \delta_n \Delta(n), \xi^+_n)}{\delta_n \Delta_i(n)} , \tag{2.9}$$

$i = 1, \ldots, N$. The analysis of SPSA works under much more general conditions on the distribution of $\Delta_i(n)$ (cf. Spall (1992), Spall (1997)). Note that unlike SPSA estimates, Kiefer-Wolfowitz gradient estimates require $2N$ (resp. $(N + 1)$) measurements when symmetric (resp. one-sided) differences are used. Bhatnagar et al. (2003) observed that in a setting of simulation based parameter optimization the use of deterministic perturbation sequences derived from certain normalized Hadamard matrices improves performance significantly in the case of one-measurement SPSA. This method of perturbation sequence generation we describe below.

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

Perturbation vectors $\Delta_i(n)$, $\forall i \in S$, $n \geq 0$, used in the perturbed policies $\pi^1_i(n)$ and $\pi^2_i(n)$ are obtained as under. The construction used is as by Bhatnagar et al. (2003, Section
2.1 Reinforcement Learning Based Algorithms For Average Cost MDPs

3). Consider \( H_d \) as a normalized Hadamard matrix (a Hadamard matrix is said to be normalized if all the elements of its first row and column are 1s) of order \( d \) with \( d \geq N_i + 1 \), \( \forall i \in S \). Let \( h^i(1), h^i(2), ... , h^i(N_i) \) be any \( N_i \) columns other than the first column of \( H_d \), and form a new \( d \times N_i \) dimensional matrix \( \tilde{H}_d \) which has the above as its columns. Let \( \tilde{\Delta}(p), p = 1, ..., d \) be the \( d \) rows of \( \tilde{H}_d \). Now set \( \Delta_i(n) = \tilde{\Delta}(n \mod d_i + 1), \forall n \geq 0 \). The perturbations are thus generated by cycling through the rows of \( \tilde{H}_d \). Matrices \( H_d \), with \( d = 2^{k_i} \), are systematically constructed as follows:

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

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

for \( k_i > 1 \), respectively.

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

2.1.3 Algorithms for Compact Action Sets

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

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

Let \( P_i^j(y) = \min(b^j_i, \max(a^j_i, y)) \), \( y \in \mathcal{R} \), be the projection of \( y \) onto the interval \( [a^j_i, b^j_i] \), \( 1 \leq j \leq N_i \). Further, for \( y = (y_1, y_2, ..., y_{N_i})^T \in \mathcal{R}^{N_i} \), let \( P_i(y) = (P_i^1(y_1), ..., P_i^{N_i}(y_{N_i}))^T \). Then \( P_i(y) \) denotes the projection of \( y \in \mathcal{R}^{N_i} \) to the set \( U(i), i \in S \). Using the method of §2.1.2 we obtain \( \{\pm 1\}^N_i \)-valued vectors \( \Delta_i(n) = (\Delta_i^1(n), ..., \Delta_i^{N_i}(n))^T \), \( \forall i \in S, n \geq 0 \) to
construct the two perturbed policies, \( \pi^1(n) \) and \( \pi^2(n) \) respectively. Here, analogous to \( \pi \), we have \( \pi^r(n) = (\pi_1^r(n), \pi_2^r(n), ..., \pi_s^r(n))^T \), with \( \pi_i^r(n) \) defined as follows:

\[
\begin{align*}
\pi_i^1(n) &= P_i(\pi_i(n) - \delta \Delta_i(n)), \\
\pi_i^2(n) &= P_i(\pi_i(n) + \delta \Delta_i(n)),
\end{align*}
\]

(2.10)

respectively. Let the iterates \( h_i^r(n) \), \( \forall i \in S, n \geq 0 \) be defined as in (2.5) with

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

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

\[
(\Delta_i(n))^{-1} = \left( \frac{1}{\Delta_1^i(n)}, ..., \frac{1}{\Delta_N^i(n)} \right)^T, \forall i \in S.
\]

We are now in a position to describe the first two algorithms. We adopt acronyms for all the algorithms in order to avoid expanded descriptions. Thus ‘ACA’ stands for Algorithms for Compact Action sets (similar to the notation used by Bhatnagar and Abdulla (2007)). The numeral 1 or 2 at the end of the algorithm name identifies if the algorithm needs 1 or 2 simulations, respectively.

**ACA-1**

For all \( i \in S \), substitute for the gradient estimate \( \tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) \) in (2.4):

\[
\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i0}^2(nL))}{\delta} (\Delta_i(n))^{-1}
\]

(2.11)

where for \( m = 0, 1, ..., L - 1 \), we perform the recursion (2.5) for \( r = 2 \) to compute the critic estimates \( h_i^2(nL) \) required above.
ACA-2

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

\[
\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i+1}^2(nL)) - (h_i^1(nL) + h_{i+1}^1(nL))}{2\delta} (\Delta_i(n))^{-1} \tag{2.12}
\]

where for \( m = 0, 1, \ldots, L - 1 \), we perform recursion (2.5) for both \( r = 1, 2 \) in order to compute \( h_i^2(nL) \) and \( h_i^1(nL) \), respectively. As stated before, ACA-2 uses two simulations while ACA-1 uses only one. Next, we present two analogs of this algorithm for finite action sets.

2.1.4 Algorithms For Finite Action Sets using RSPs

Suppose \( U(i), i \in S \), be finite action sets of the form \( U(i) = \{u_0^i, u_1^i, \ldots, u_n^i\} \), where \( u_j^i, 0 \leq j \leq N_i \) are feasible actions in state \( i \). We assume:

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

Let \( S_i = \{(y^1, \ldots, y^{N_i})^T | y^j \geq 0, \forall j \in \{1, 2, \ldots, N_i\}, \sum_{j=1}^{N_i} y^j \leq 1\} \) denote the simplex of the probabilities of selecting actions in \( U(i) \setminus \{u_0^i\} = \{u_1^i, \ldots, u_{N_i}^i\} \). We now define projection \( P_i : \mathcal{R}^{N_i} \mapsto S_i \), such that for any \( x \in \mathcal{R}^{N_i} \), \( P_i(x) \) is the closest point in \( S_i \) to \( x \). Let \( \tilde{\Delta}_i(n) \) be a vector of \( \{\pm 1\}^{N_i} \)-valued perturbations generated in the same manner as the \( \Delta_i(n) \) used in ACA-\( r \). We have seen earlier that \( \hat{\pi} = (\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_s)^T \) identifies an RSP \( \pi \) completely and that \( \hat{\pi}_i \in S_i, \forall i \in S \).

Let \( \hat{\pi}_1^i(n) = (P_i(\hat{\pi}_i(n) - \delta \hat{\Delta}_i(n)), \forall i \in S)^T \) and \( \hat{\pi}_2^i(n) = (P_i(\hat{\pi}_i(n) + \delta \hat{\Delta}_i(n)), \forall i \in S)^T \), respectively, denote the perturbed policies. Let

\[
(\hat{\Delta}_i(n))^{-1} = \left( \frac{1}{\hat{\Delta}_1^i(n)}, \ldots, \frac{1}{\hat{\Delta}_{N_i}^i(n)} \right)^T,
\]

as before. In the following, ‘RPAFA’ stands for Randomized Policy Algorithms over Finite Action sets.
RPAFA-1

Replace $\pi_i(n)$ and $\pi_i(n+1)$ in (2.4) with $\hat{\pi}_i(n)$ and $\hat{\pi}_i(n+1)$, respectively. As was done in ACA-1 (cf. (2.11)), for all $i \in S$ substitute for $\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)})$ of (2.4) with:

$$\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i_0}^2(nL))}{\delta} (\hat{\Delta}_i(n))^{-1}$$ (2.13)

where for $m = 0, 1, ..., L - 1$, we perform (2.5) with $r = 2$. Note that, as seen earlier in (2.5), the action taken in state $i$ i.e. $\xi_{nL+m}^2(i, \pi_i^2(n))$ is now a $U(i)$—valued random variable that selects actions according to the $(N_i + 1)$—element p.m.f. $\pi_i^2(n)$, unlike in ACA-2 where $\xi_{nL+m}^2(i, \pi_i^2(n))$ was the deterministic action $\pi_i^2(n)$ itself. This therefore results in a different form of the Poisson equation (cf. (2.3)) that is tracked by the recursion (2.5) for a given RSP as explained earlier.

RPAFA-2

Replace $\pi_i(n)$ and $\pi_i(n+1)$ in (2.4) with $\hat{\pi}_i(n)$ and $\hat{\pi}_i(n+1)$, respectively. Analogous to ACA-2 (cf. (2.12)), the policy gradient estimate is:

$$\tilde{\nabla}_i G_{\pi(n)}(i, h_{\pi(n)}) = \frac{(h_i^2(nL) + h_{i_0}^2(nL)) - (h_i^1(nL) + h_{i_0}^1(nL))}{2\delta} (\hat{\Delta}_i(n))^{-1}$$ (2.14)

where for $m = 0, 1, ..., L - 1$, recursion (2.5) is performed for both $r = 1, 2$. As with RPAFA-1, $\xi_{nL+m}^r(i, \pi_i^r(n))$ are also $U(i)$—valued random variables.

2.1.5 Algorithms for Finite Action sets using Deterministic Policies

In order to develop the remaining two algorithms for finite action sets, we make some further assumptions on the structure of $U(i)$, $\forall i \in S$ used in RPAFA-r above. Let $U(i)$ be the feasible action set such that $U(i) = \prod_{j=1}^{N_i} U^3(i)$, where $U^3(i)$ are finite sets $\{u_i^1(1), u_i^1(2), ..., u_i^1(N_i^1)\}$ and $|U^3(i)| = N_i^2$. An admissible control $\pi_i \in U(i)$, $\forall i \in S$ is
2.1 Reinforcement Learning Based Algorithms For Average Cost MDPs

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

In the proposed algorithm, the policy iterates $\pi_i(n)$ evolve in the convex hull $\hat{U}(i)$ to project into which the projection $P_i$ as used in ACA-$r$ is employed. Thus, the perturbed deterministic policies at the $n$-th iteration are $\pi^1(n) = (P_i(\pi_i(n) - \delta\Delta_i(n)), \forall i \in S)^T$ and $\pi^2(n) = (P_i(\pi_i(n) + \delta\Delta_i(n)), \forall i \in S)^T$, where $P_i : \mathcal{R}^{N_i} \mapsto \hat{U}(i)$ is such that $P_i^j(x_i^j) = \min(\max(u_i^j, x_i^j), b_i^j)$, $1 \leq j \leq N_i$. Note that the policy $\pi_r(n)$ to be applied in (2.5) need not be admissible. It is made so by another projection $\bar{P}_i(\pi_i(n))$ i.e. $\xi_{nL+m}(i, \pi_i^r(n)) = \bar{P}_i(\pi_i^r(n))$. Here, $\bar{P}_i(\pi_i^r(n)) = \arg\min_{u \in \hat{U}(i)} ||\pi_i^r(n) - u||$ is the map into the feasible set $U(i)$ of the control $\pi_i^r(n) \in \hat{U}(i)$. If the minimum above is attained at two points, one of the two is arbitrarily selected. In contrast, ACA-$r$ algorithms apply $\xi_{nL+m}(i, \pi_i^r(n)) = \pi_i^r(n)$ in recursion (2.5). The letters ‘DPFA’ in the following stand for Deterministic Policy Algorithms over Finite Action sets.

**DPFA-1**

Redefine $\eta_{nL+m}^r(i, \xi_{nL+m}^r(i, \pi_i^r(n)))$, $\forall n, m > 0$, $r = 2$, in (2.5) to be independent families of i.i.d. random variables with distribution $p(i, \bar{P}_i(\pi_i^r(n)), \cdot)$ with $\xi_{nL+m}^r(i, \pi_i^r(n)) = \bar{P}_i(\pi_i^r(n))$ as described above. DPFA-1 is now the same as ACA-1.

**DPFA-2**

This algorithm is similar to ACA-2, using $\eta_{nL+m}^r(\cdot, \cdot)$ and $\xi_{nL+m}^r(\cdot, \cdot)$ as described above, for both $r = 1$ and 2.
2.1.6 Convergence Analysis

We use Assumption (A) first and prove convergence for ACA-\(r\). The differential cost \(h_\pi(i), \forall i \in S\), for an infinite-horizon MDP operating under a stationary deterministic policy \(\pi\) can be obtained as the unique solution of the following linear system of \(s\) equations (i.e., the Poisson equation, also identified in (2.1)):

\[
G_\pi(i, h_\pi) = h_\pi(i_0) + h_\pi(i) = \sum_{j \in S} p(i, \pi_i, j)(K(i, \pi_i, j) + h_\pi(j)).
\] (2.15)

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

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

**Proof:** Let \(\tilde{b}(n) = b([\frac{n}{L}]), n \geq 0\), where \([\frac{n}{L}]\) denotes the integer part of \(\frac{n}{L}\). Then the recursion in (2.5) can be written as
\[
h_\pi^r(n + 1) = (1 - \tilde{b}(n))h_\pi^r(n) + \tilde{b}(n)(K(i, \xi^r_n(i, \tilde{\pi}_n^r(n))), \eta^r_n(i, \xi^r_n(i, \tilde{\pi}_n^r(n))))
- h_\pi^r(n) + h_\pi^r(i, \xi_n^r(i, \tilde{\pi}_n^r(n)))(n)
\]
where \(\tilde{\pi}_n^r(n) = \pi_n^r([\frac{n}{L}])\). The above is now analogous to recursion (3.5) of Konda and Borkar (1999). The proof now follows in a similar manner as (Konda and Borkar, 1999, Lemma 6.4).

Define \(\{s(n)\}\) as \(s(0) = 0, s(n) = \sum_{k=0}^{n-1} c(k), n \geq 1\). Let \(\Delta_i(t) = (\Delta_i^1(t), ..., \Delta_i^N(t))^T\) be defined according to \(\Delta_i(t) = \Delta_i(n)\) for \(t \in [s(n), s(n + 1)]\), \(n \geq 0, i \in S\). Suppose that for any bounded, continuous and real-valued function \(v(\cdot)\),
\[
\hat{P}_i^j(v(y)) = \lim_{\eta \to 0} \left( \frac{P_i^j(y + \eta v(y)) - y}{\eta} \right)
\] (2.16)
for \( j = 1, \ldots, N_\pi, i \in S \). Further for any \( y = (y_1, \ldots, y_N_\pi)^T \), let \( \dot{P}_i(y) = (\dot{P}^1_i(y), \ldots, \dot{P}^N_\pi(y_{N_\pi}))^T \).

For a given stationary deterministic policy \( \pi = (\pi_i, i \in S)^T \), suppose

\[
F_\pi(i, h) \triangleq \sum_{j \in S} p(i, \pi_i, j)(K(i, \pi_i, j) + h_j) - h_{i_0},
\]

where \( h \) is any vector \( (h_j, j \in S)^T \). The Poisson equation \((2.15)\) can now be written as

\[
h_\pi(i) = F_\pi(i, h_\pi) = G_\pi(i, h_\pi) - h_\pi(i_0). \tag{2.17}
\]

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

\[
\dot{\pi}_i^r(t) = 0, \tag{2.18}
\]

\[
\dot{h}_i^r(t) = F_{\pi^r}(i, h^r(t)) - h_i^r(t). \tag{2.19}
\]

Here \( \tilde{\pi}_i^1(t) = P_i(\tilde{\pi}_i(t) - \delta \Delta_i(t)) \) and \( \tilde{\pi}_i^2(t) = P_i(\tilde{\pi}_i(t) + \delta \Delta_i(t)) \), respectively for all \( i \in S \).

Note that \((2.18)\) corresponds to a fixed policy \( \tilde{\pi}^r \) that is independent of \( t \). Thus, \((2.19)\) can be written as (with a time invariant \( \tilde{\pi}^r \))

\[
\dot{h}_i^r(t) = F_{\pi^r}(i, h^r(t)) - h_i^r(t), \tag{2.20}
\]

\( \forall i \in S \). The ODE \((2.20)\) is (cf. \cite{KondaBorkar1999}, Lemma 6.3)) an asymptotically stable linear system with \( h_{\tilde{\pi}^r}(i), \forall i \in S \) as its unique asymptotically stable equilibrium point. Let \( \nabla_i h_\pi(i) \) denote the \( N_\pi \)-vector gradient of \( h_\pi(i) \) w.r.t. \( \pi_i \). Consider now the ODE:

\[
\dot{\pi}_i(s) = \dot{P}_i(- \nabla_i h_\pi(s)(i) - \nabla_i h_\pi(s)(i_0)) = \dot{P}_i(- \nabla_i G_\pi(s)(i, h_\pi(s))) \tag{2.21}
\]

for all \( i \in S \). Let \( M = \{ \pi \mid \dot{P}_i(\nabla_i h_\pi(s)(i) + \nabla_i h_\pi(s)(i_0)) = 0, \forall i \in S \} \) be the set of all fixed points of \((2.21)\). Also, given \( \epsilon > 0 \), let \( M^\epsilon = \{ \pi \mid \exists \tilde{\pi} \in M \ s.t. \| \pi - \tilde{\pi} \| < \epsilon \} \) be the set of all policies that are within a distance \( \epsilon \) from \( M \).
Applying standard martingale analysis on (2.5) using (2.6) and Gronwall’s inequality upon the ODE (2.20), we can see along similar lines as (Bhatnagar and Kumar, 2004, Corollary 4.2),

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

We now note that the slower time-scale recursion (2.4) can be rewritten as

\[
\pi_i(n + 1) = P_i(\pi_i(n) - \tilde{b}(n)\xi(n))
\]

(2.22)

where \( \xi(n) = o(1) \) since \( c(n) = o(\tilde{b}(n)) \). Now use (2.22), Lemma 2 and an application of Hirsch’s Lemma (reproduced as Lemma 5 below, due to the more general setting there).

Thus, ACA-r asymptotically track the trajectories of the ODE (2.18)-(2.19) along the faster timescale \( \{t(n)\} \).

One now needs to show that on the slower timescale, \( \pi(n) \) converges to \( M \) in the limit as \( \delta \to 0 \). We first consider the case of ACA-2. Define \( U^\alpha \equiv \prod_{i=1}^s U^\alpha(i) \) as the interior of the space \( U = \prod_{i=1}^s U(i) \) of all feasible policies.

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

**Proof:** We assume that \( \forall i \in S, N_i = N \), the general case being a minor modification. Fix \( 1 \leq k_0 \leq N \). We may rewrite recursion (2.4) as follows:

\[
\pi_i^{k_0}(m + 1) = \pi_i^{k_0}(m) - c(m)\nabla_i G_{\pi(m)}(i, h_{\pi(m)}) + c(m)Z_i^{k_0}(m), \forall i \in S, k_0 \in \{1, 2, ..., N\}
\]

where \( Z_i^{k_0}(m) \) is error due to projection \( P_i^{k_0} \). Define \( H_i^{k_0}(m) = -\nabla_i G_{\pi(m)}(i, h_{\pi(m)}) \), then using (2.12), we have

\[
H_i^{k_0}(m) = \frac{h_{\pi_1(m)}(i) + h_{\pi_1(m)}(i_0) - h_{\pi_2(m)}(i) - h_{\pi_2(m)}(i_0)}{2\delta \Delta_i^{k_0}(m)} + r_i^{k_0}(m),
\]

\( m \geq 0 \).
where \( r^k_i(m) = o(1) \) by Lemma 2. For a fixed \( d \in \mathbb{Z}^+ \), we may write the above iteratively as:

\[
\pi^k_i(m + d) = \pi^k_i(m) + \sum_{l=m}^{m+d-1} c(l) H^k_i(l) + \sum_{l=m}^{m+d-1} c(l) Z^k_i(l), \forall i \in S. \tag{2.23}
\]

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

\[
H^k_i(m) = -\nabla^k_i h_{\pi(m)}(i) - \nabla^k_i h_{\pi(m)}(i_0) - \sum_{q=1,q \neq k_0}^N \frac{\Delta^q_i(m)}{\Delta^k_i(m)} \nabla^q_i \left( h_{\pi(m)}(i) + h_{\pi(m)}(i_0) \right) + O(\delta). \tag{2.24}
\]

In the above, we assume that \( \pi_i(m) \in U^o(i), \forall i \in S \) and \( \delta > 0 \) is small enough so that \( \pi^*_i(m) \in U^o(i) \). For the case of \( \pi^*_i(m) \in U(i) \setminus U^o(i) \), the above would continue to hold except for a constant multiplying the first RHS term in (2.24) (cf. Corollary 4.3 of Bhatnagar and Kumar (2004)). Now from (2.23) and (2.24), we have

\[
\pi^k_i(m + d) = \pi^k_i(m) - \sum_{l=m}^{m+d-1} c(l) \nabla^k_i \left( h_{\pi(l)}(i) + h_{\pi(l)}(i_0) \right) - c(m) \sum_{l=m}^{m+d-1} \sum_{q=1,q \neq k_0}^N \frac{c(l)}{c(m)} \frac{\Delta^q_i(l)}{\Delta^k_i(l)} \nabla^q_i \left( h_{\pi(l)}(i) + h_{\pi(l)}(i_0) \right) + \sum_{l=m}^{m+d-1} c(l) Z^k_i(l) + O(\delta). \tag{2.25}
\]

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

\[
\left| \sum_{l=m}^{m+d-1} \sum_{k=1,k \neq k_0}^N \frac{c(l)}{c(m)} \frac{\Delta^k_i(l)}{\Delta^k_i(l)} \nabla^k_i \left( h_{\pi(l)}(i) + h_{\pi(l)}(i_0) \right) \right| \to 0. \tag{2.26}
\]
Therefore, (2.25) is equivalent in an asymptotic sense to the recursion

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

(2.27)

except for an additional error term which however is \(O(\delta)\). It can now be seen, as in pp. 191-194 of Kushner and Clark (1978), that the above is a discretization of the projected ODE (2.21). Finally, note that \(M\) is an asymptotically stable attractor set for the ODE (2.21) with \(\sum_{i \in S} G_\pi(i, h_\pi)\) itself serving as the associated strict Liapunov function. The claim now follows. \(\Box\)

With the same arguments, the above theorem also holds for RPAFA-2, the difference being that since randomized policies are now used, the Poisson equation tracked on the faster scale by a given randomized policy is (2.3) instead of (2.1) as above. The modifications required in the proof for the other algorithms are identified in the following:

**ACA-1**

Using the gradient estimate in (2.11), we may modify the proof in Theorem 1 with

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

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

\[
H_i^{k_0}(n) = -\frac{(h_{\pi(n)}(i) + h_{\pi(n)}(i_0))}{\delta \Delta_i^{k_0}(n)} - \nabla_i^{k_0} h_{\pi(n)}(i) - \nabla_i^{k_0} h_{\pi(n)}(i_0)
- \sum_{q=1,q \neq k_0}^N \frac{\Delta_q^q(n)}{\Delta_i^{k_0}(n)} \nabla_i^q \left( h_{\pi(n)}(i) + h_{\pi(n)}(i_0) \right) + O(\delta)
\]

(2.28)

which is similar to (2.24), except for the additional first term in the RHS above. Note that \(d = 2^{\lceil \log_2 N \rceil + 1}\) here. A similar argument as the one using (2.26) holds. Further, using
Corollary 2.6 of Bhatnagar et al. (2003), we obtain
\[
\left| \frac{1}{\Delta^k_i(l)} \sum_{l=m}^{m+d-1} \frac{c(l)}{c(m)} \left( h_{\pi(l)}(i) + h_{\pi(l)(i_0)} \right) \right| \to 0 \quad \text{as } m \to \infty,
\]
(2.29)
as well. This makes the algorithm equivalent to (2.27) with asymptotically diminishing error terms and the result follows. With the changes mentioned earlier, Theorem 1 can be seen to hold for RPAFA-1 as well.

**DPAFA-1**

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

\[
0 \leq |\delta_i^j(n)| \leq \tilde{\delta} < \infty, \forall n \geq 0, i \in S, 1 \leq j \leq N,
\]
for some \( \tilde{\delta} > 0 \) that is a function of the ‘fineness’ of the grid. We now perform Taylor series expansion about \( \pi(n) \) as in (2.28) to obtain

\[
\frac{h_{\pi^2(n)}(i) + h_{\pi^2(n)(i_0)}}{\delta \Delta_i^k(n)} = \frac{h_{\pi(n)}(i) + h_{\pi(n)(i_0)}}{\delta \Delta_i^k(n)} + \frac{\delta_i^k(n)}{\delta} \left( \nabla^k_i h_{\pi(n)}(i) + \nabla^k_i h_{\pi(n)(i_0)} \right) + \sum_{q=1, q \neq k_0}^N \frac{\delta_i^q(n) \Delta_i^q(n)}{\delta \Delta_i^k(n)} \nabla^k_i \left( h_{\pi(n)}(i) + h_{\pi(n)(i_0)} \right) + O(\delta)
\]

We use the limit in (2.29) for the first term in the RHS above. For \( k \neq k_0 \), note that

\[
\left| \frac{1}{\Delta^k_i(l)} \sum_{l=m}^{m+d-1} \frac{\delta_i^k(l)}{\delta} \frac{c(l)}{c(m)} \Delta_i^k(l) \nabla_i^k \left( h_{\pi(l)}(i) + h_{\pi(l)(i_0)} \right) \right| \to 0 \quad \text{as } \delta \to 0.
\]
(2.30)

In the above, the fact that \( \frac{c(l)}{c(m)} \to 1 \) as \( m \to \infty, l \in \{m, m+1, ..., m+d-1\} \) is used. Note that as the partition is made ‘finer’, \( \delta_i^k(l) \to \delta \), the above reduces to the case of (2.25). The algorithm will then converge to a local minimum in the limit as \( \delta \to 0 \). For a fixed \( \delta > 0 \), the algorithm can be seen to converge either to the local minimum or a point in its neighborhood. A similar analysis holds for DPAFA-2.
2.1.7 A Variant of the TD(0) Algorithm

We propose a variant of the two-timescale TD(0) algorithm of Konda and Tsitsiklis (2003). Here we approximate the differential cost \( h(\cdot) \) using a linear approximation architecture (cf. Bertsekas and Tsitsiklis (1996) and Tsitsiklis and Van Roy (1997)) as

\[
\tilde{h}(i) = \phi(i)^T \nu,
\]

where \( \nu, \phi(i) \in \mathbb{R}^K \) for some \( K \geq 1 \). Here, \( \phi(i) \) is called the feature vector of state \( i \), (see Tsitsiklis and Van Roy (1997), Tsitsiklis and Van Roy (1999a) and Konda and Tsitsiklis (2003)). We have the following recursion in place of (2.5):

\[
\forall i \in S, \quad \nu_{r_n} \hat{L} + ms + i + 1 = \nu_{r_n} \tilde{L} + ms + i + b(n) \phi(i) \cdot \left( K(i, \pi_i(n), \eta_{nL+m}(i, \xi_{nL+m}(i, \pi_i(n)))) + \tilde{h}_{nL+ms+i}^r(i) - \tilde{h}_{nL+ms+i}^r(i) - \tilde{h}_{nL+ms+i}^r(i) \right) (2.31)
\]

for \( 1 \leq m \leq L \) and \( \hat{L} = Ls \). The \( n \)-th policy update is performed using \( \tilde{h}_{nL}^r(i) = \phi(i)^T \nu_{nL}^r, \forall i \in S \) in place of \( h_{nL}^r(i) \) in recursion (2.4) for all algorithms.

Note that in (2.31), the states \( i = 1, 2, ..., s \) are thus sampled cyclically, one after another. Thus, \( s \) updatings of the vector \( \nu^r \) would be performed before the algorithm visits state \( i \) again, and \( sL \) updatings would be performed between two consecutive updates of the policy \( \pi(n) \) using (2.4). Such a method corresponds to the implementation of the analogous ‘critic’ recursion (2.5) in the previous six algorithms. However, the recursion in (2.31) requires less memory due to the use of a \( K \)-dimensional coefficient vector \( \nu^r \) in place of a size \( s \) vector \( h^r \) in (2.5), where typically \( K \ll s \).

Note that in visiting states cyclically (in a given lexicographic order), we are sampling the state space with a distribution that is not the same as the stationary distribution imposed by policy \( \pi^r(n) \) in (2.31), as is asserted by (Bertsekas and Tsitsiklis, 1996, Section 6.3) and (Tsitsiklis and Van Roy, 1997, Section 9). Nevertheless, the method is good in practice as borne out by the simulation results.

We give an outline of the theoretical impediments in proving convergence for (2.31). Consider an \( s \times K \) matrix \( \Phi \) where \( \phi_i^T, \forall i \in S \) are the rows. Just as Tsitsiklis and Van Roy (1997) and Tsitsiklis and Van Roy (1999a), \( \Phi \) is assumed to have full column rank, a requirement implying that none of the features are correlated. Using Lemma 4 of
Tsitsiklis and Van Roy (1999a) with appropriate modifications, we have to establish that the (vector) linear ODE

\[ \dot{\nu}(t) = \frac{1}{s} \Phi^T A^r \Phi \nu(t) + \frac{1}{s} \Phi^T B^r \]  

(2.32)

is asymptotically stable, where \(A^r\) and \(B^r\) are \(s \times s\) and \(s \times 1\) matrices, respectively, used in the ‘exact’ critic ODE (cf. (2.19))

\[ \dot{h}(t) = A^r h(t) + B^r \]  

(2.33)

Here, using (2.15), we see that \(A^r = P_{\pi^r(n)} - I - I_0\) where \(I_0(k, i_0) = 1\) for \(1 \leq k \leq s\) and 0 elsewhere whereas \(B^r_i = \sum_{j=1}^{s} p(i, \pi^r_i, j) K(i, \pi^r_i, j), \forall i \in S\). Using (Perko, 1998, Theorem 2, pp.56), \(A^r\) will have negative real parts in all its eigen values by virtue of the asymptotic stability of (2.33) (see Lemma 6.3 of Konda and Borkar 1999 for a proof of asymptotic stability of (2.33)). However, \(A^r\) need not be negative definite in general, which would have been a sufficient condition to render (2.32) asymptotically stable, since then \(\Phi^T A^r \Phi\) would also be negative definite. As a result, (2.31) cannot be guaranteed to track an asymptotically stable ODE for all policies \(\pi^r(n)\). However, as already mentioned, the above algorithm shows good numerical performance on our setting.

### 2.2 Solution of MDPs using Simulation-based Value Iteration

We use here the notation from \(\{2.1.1\}\) previously. Consider the MDP \(\{X_t, t = 0, 1, \ldots\}\) with decisions at instants \(t \in Z^+ \cup \{0\}\) constituting the associated control process \(\{Z_t\}\), \(S \equiv \{1, 2, \ldots, s\}\) being the (finite) state space and \(C\) the control space. Suppose that \(U(i) \subseteq C\) is the set of all feasible controls in state \(i\), and let \(p(i, u, j), i, j \in S, u \in U(i)\) be the transition probabilities associated with this MDP. An admissible policy \(\mu = \{\mu^0, \mu^1, \mu^2, \ldots\}\) with \(\mu^t : S \mapsto C, t \geq 0\), is such that \(\mu^t_i \in U(i)\), and \(\mu\) is stationary if \(\mu^t = \gamma, \forall t\). Suppose
2.2 Solution of MDPs using Simulation-based Value Iteration

$K(i_t, u_t, i_{t+1})$ denotes the one-step transition cost when the current state is $i_t$, the action chosen is $u_t \in U(i_t)$ and the next state is $i_{t+1}$. The aim here is to find a stationary admissible policy $\pi$ that minimizes for each state $i \in S$ the associated infinite horizon discounted cost $V_\pi(i)$, where

$$V_\pi(i) = E\left\{ \sum_{t=0}^{\infty} \alpha^t K(i_t, \pi_{i_t}, i_{t+1}) \mid i_0 = i \right\},$$  \hspace{1cm} (2.34)

and $0 < \alpha < 1$ is the discount factor. For any state $i \in S$, the minimum infinite-horizon discounted-cost $V^*_i$ satisfies the Bellman equation

$$V^*_i = \min_{u \in U(i)} E\{ K(i, u, \eta(i, u)) + \alpha V^*_{\eta(i,u)}(n) \},$$  \hspace{1cm} (2.35)

the random variable $\eta(i, u) \in S$ being the state to which the system transits upon application of control $u$ in state $i$. The expectation above is over $\eta(i, u)$. To compute $V^* = (V^*_i, i \in S)^T$ we may apply the Dynamic Programming algorithm recursively:

$$V_i(n+1) = \min_{u \in U(i)} E\{ K(i, u, \eta(i, u)) + \alpha V^*_{\eta(i,u)}(n) \},$$  \hspace{1cm} (2.36)

Note that $V(n) = (V_i(n), \forall i \in S)^T$ are iterates unlike defined values $V_\pi = (V_\pi(i), \forall i \in S)^T$ in (2.34). It is known that as $n \to \infty$, the iterates $V_i(n)$ converge to $V^*_i$ exponentially fast. We may rewrite (2.36) as $V_i(n+1) = T_i(V(n))$ where $T : \mathcal{R}^S \to \mathcal{R}^S$ such that $T = (T_i, \forall i \in S)^T$ is the Dynamic Programming (DP) operator and the vector $V^*$ is the unique fixed point of $T$. This method of continually applying $T$ is termed as successive approximation or value iteration (cf. §8.2 of Bertsekas (1995)). The optimal policy $\pi^*$ (where $V_{\pi^*} = V^*$) can be inferred from (2.35), in general, provided the system transition probabilities $p(i, \pi^*_i, \cdot)$ are known, which is not true in most real-life applications. Even when $p(i, \pi_i, \cdot)$ are available but not obtainable via a closed form expression, storing these values when $|S|$ is large is difficult. Even so, such a possibility is ruled out in the case where $U(i)$ are compact or countably infinite. Denote $\pi_i(n)$ as the ‘greedy’ policy w.r.t.
the iterates $V(n)$ of (2.36), i.e.

$$
\pi_i(n+1) = \arg\min_{u \in U(i)} E\{K(i, u, \eta(i, u)) + \alpha V_{\eta(i, u)}(n)\}
$$

(2.37)

then we may infer that $\pi_i(n) \to \pi_i^*$ although tight bounds of the form

$$
V_i(n) + c_i(n) \leq V_i^* \leq V_i(n) + \bar{c}_i(n),
$$

where $c_i(n) \leq \bar{c}_i(n)$, available in value iteration hold if $\pi(n)$ belong to a space with a well-defined metric topology, e.g. $\pi(n) \in \mathcal{R}^s$, $n \geq 0$. In this work, we assume a compact action set $U(i) \subset \mathcal{R}^{N_i}$, $\forall i \in S$.

Simulation based approaches to value iteration in the literature go under the rubric critic-only algorithms. See, for example, §1 of [Konda and Tsitsiklis (2003)], where the drawbacks of such schemes are also identified. In a typical simulation based implementation of $T_i$, first the estimation of $E\{K(i, u, \eta(i, u)) + \alpha V_{\eta(i, u)}(n)\}$ is performed, for each action $u \in U(i)$. This procedure is not feasible when $U(i)$ is a compact or even a countably infinite set. This is followed by finding the minimum of such terms over $U(i)$, we call this minimum $\tilde{V}_i(n+1)$, and (possibly) storing the minimizing $u$ as iterate $\tilde{\pi}_i(n+1)$. Note that this minimization step has computational complexity $O(\sum_{i=1}^s |U(i)|)$. An example of critic-only value iteration are the algorithms in §6.2 and §6.3 of [Van Roy et al. (1997)], that employ online $TD(0)$ learning, both without and with exploration of $U(i)$, respectively. However, these algorithms use value function approximation with finite action-set whereas we target the look-up table case with compact action-sets. An important difference in approach is that [Van Roy et al. (1997)] simulate the system using a single endless trajectory to enable learning of the coefficients involved in value function approximation. As we explain in §2.3.3, for a trivial case of function-approximation, this is analogous to simulating single transitions out of each state $i \in S$ repeatedly. Further, the setting of [Van Roy et al. (1997)] differs from the $TD(0)$ variant of §2.1.7 above. The greedy (or greedy-cum-exploratory) action $\pi_i(n+1)$ is found after simulating $|U(i)|$ transitions from $i$, after which the state reached by taking action $\pi_i(n+1)$ is transited to. Two examples of
other systems where such a method is employed are to be found of Tsitsiklis and Van Roy (1999b) and Singh and Bertsekas (1997).

In contrast, we use the actor-critic approach to approximate $T$. Gradient search is performed using a slower timescale recursion for finding the minimizing control in (2.36). Thus, given value function estimates $V(n)$, the iterates $\pi(n + 1)$ represent an approximation to $\pi(n + 1)$ of (2.37). To perform gradient descent we employ SPSA estimates of the true gradient

$$\nabla_u E\{K(i, u, \eta(i, u)) + \alpha V_{\eta(i,u)}(n)\}$$

in a stochastic approximation recursion, similar to the approach of Bhatnagar and Kumar (2004) and algorithms ACA-r of §2.1 above. Note that the difficulty here arises since it is not only the expectation $E\{\cdot\}$ that needs to be evaluated but also the value function itself, before the gradient is estimated. Clearly, the objective function here does not possess any analytically obtainable expression and hence needs to be estimated. In particular, to obtain control $\pi_i(n + 1)$ we evaluate $E\{K(i, u^r, \eta(i, u^r)) + \alpha V_{\eta(i,u^r)}(n)\}$, $r = 1, 2$ at two ‘perturbed’ actions, $u^1$ and $u^2$ that will be made precise later. A faster time-scale recursion estimates the two $E\{\cdot\}$ terms required above by independently simulating $L$ transitions from state $i$ using actions $u^1$ and $u^2$, respectively, thus permitting two ‘parallel’ simulations. Note that in the normal course, given a control $u \in U(i)$, one would wait for convergence of iterates towards the $E\{\cdot\}$ terms required above. Instead we only perform a fixed $L$ steps of averaging the random variables

$$\{K(i, u^r, \eta_m^r(i, u^r)) + \alpha V_{\eta_m^r(i,u^r)}(n)\}, 0 \leq m \leq L - 1, r = 1, 2,$$

where $\eta_m^r$ now stands for the two parallel $L$-element random processes having the law $p(i, u^r, \cdot)$. Note that in the above, the convergence analysis permits us to fix $L$ at 1 - although better performance was obtained when a larger $L$ was used. We use this averaging parameter $L$ to good effect in the later algorithms – the convergence properties are not impacted due to this device, and the performance improves. Convergence in the proposed algorithm is achieved because of the different step size schedules or timescales as with algorithms
of §2.1, Bhatnagar and Abdulla (2007), Bhatnagar and Kumar (2004), Bhatnagar et al. (2003), Konda and Borkar (1999), and Konda and Tsitsiklis (2003). While in critic-only methods the resultant optimal policies are not explicitly computed, a highlight of the proposed algorithm is that due to its actor-critic structure, the convergence criteria can now be designed based on both the policy iterates \( \tilde{\pi} \) and the value-function iterates \( V \). This helps to accommodate problem-specific sensitivity to these quantities.

### 2.2.1 Algorithm

Suppose \( U(i), i \in S \) are of the form \( U(i) = \prod_{j=1}^{N_i} [a^i_j, b^i_j] \), where \( a^i_j, b^i_j \in \mathbb{R} \). We make the assumption, as in §2.1.3 above, that \( \forall i, j \in S, u \in U(i) \), both \( K(i, u, j) \) and \( p(i, u, j) \) are continuously differentiable in \( u \).

Let \( P^i_j(y) = \min (b^i_j, \max (a^i_j, y)), y \in \mathbb{R} \) be the projection of \( y \) onto the interval \( [a^i_j, b^i_j] \), \( 1 \leq j \leq N_i \). Further, for \( y = (y_1, y_2, ..., y_{N_i})^T \in \mathbb{R}^{N_i} \), let \( P_i(y) = (P^1_i(y_1), ..., P^N_i(y_{N_i}))^T \). Then \( P_i(y) \) denotes the projection of \( y \in \mathbb{R}^{N_i} \) to the set \( U(i), i \in S \). Also define an operator \( P(\cdot) \) as

\[
P(x) = (P_1(x_1), P_2(x_2), ..., P_s(x_s))^T
\]

where \( x_j \in \mathbb{R}^{N_i}, 1 \leq j \leq s, \) and \( x = (x_1, x_2, ..., x_s)^T \).

The recursion that tracks (2.36) is the following: For all \( i \in S \) and \( n \geq 0 \):

\[
V_i(n + 1) = V_i(n) + a(n)(\tilde{V}_i(n + 1) - V_i(n))
\]

(2.38)

where

\[
\tilde{V}_i(n + 1) = \frac{1}{2}(\tilde{V}_i^1((n + 1)L) + \tilde{V}_i^2((n + 1)L))
\]

is an approximation to \( T_i(V(n)) \) and \( a(n) \) is a diminishing step size. Here, \( L \) is the number of instants over which additional averaging in recursion (2.40) is performed. The iterate \( \tilde{V} \) will be made precise below in the description of (2.40). Recursion (2.39) updates \( \pi(\cdot) \):

\[
\tilde{\pi}_i(n + 1) = P_i \left( \tilde{\pi}_i(n) + b(n) \left( \frac{\tilde{V}_i^1((n + 1)L) - \tilde{V}_i^2((n + 1)L)}{2\delta \Delta_i(n)} \right) \right).
\]

(2.39)
Here the $\Delta_i(\cdot)$ are deterministic normalized Hadamard-matrix based perturbations generated according to the method in §2.1.2 above whereas $\delta > 0$ is a small constant. To produce the iterates $\bar{V}_i^1((n+1)L)$ and $\bar{V}_i^2((n+1)L)$ consider perturbed policies

$$\bar{\pi}_i^1(n) = P_i(\bar{\pi}_i(n) - \delta \Delta_i(n))$$

and

$$\bar{\pi}_i^2(n) = P_i(\bar{\pi}_i(n) + \delta \Delta_i(n)).$$

Now perform the following iterations for $r = 1, 2$ and $m = 0, 1, \ldots, L - 1$:

$$\bar{V}_i^r(nL + m + 1) = \bar{V}_i^r(nL + m) + c(n) \left( K(i, \bar{\pi}_i^r(n), \eta^r_{nL+m}(i, \bar{\pi}_i^r(n))) + \alpha V_{\eta^r_{nL+m}(i, \bar{\pi}_i^r(n))} - \bar{V}_i^r(nL + m) \right), \quad (2.40)$$

Here, we run two simulations corresponding to $r = 1$ and $r = 2$ in parallel. Note that in the above, $\eta^1$ and $\eta^2$ represent two stochastic processes with elements of the form $\eta_k(i, u), k \geq 0$ which are $S -$ valued random variables, independently generated using $p(i, u, \cdot)$. The requirements on the step-sizes $a(n), b(n)$ and $c(n)$ used in (2.38), (2.39) and (2.40), respectively, are as follows:

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

$$\sum_n a(n) = \sum_n b(n) = \sum_n c(n) = \infty, \quad \sum_n a(n)^2, \sum_n b(n)^2, \sum_n c(n)^2 < \infty, \quad (2.41)$$

Thus $a(n) \to 0$ the fastest and $c(n) \to 0$ the slowest among the three step-size schedules. As a result, the timescale corresponding to $\{a(n)\}$ is the slowest while that corresponding to $\{c(n)\}$ is the fastest. This is because beyond some finite $N_0 > 0$ (i.e., $n > N_0$), the increments in the recursion governed by $\{a(n)\}$ are uniformly the smallest.
2.2 Solution of MDPs using Simulation-based Value Iteration

while those in recursion governed by \(\{c(n)\}\) are uniformly the largest among the three. Hence recursions governed by \(\{c(n)\}\) converge the fastest (even though they exhibit higher variability in their iterates) among the three recursions. Note that we could replace (2.38) with the direct-assignment recursion \(V_i(n + 1) = \tilde{V}_i(n + 1)\). However, using stochastic averaging with step-size \(a(n)\) results in graceful convergence properties in our experiments. Such a choice between the averaging of (2.38) and direct-assignment is an example of the bias-variance dilemma. This method is also called a small-step version of value iteration (cf. [Bertsekas and Tsitsiklis, 1996, eq. (5.5.2)]).

2.2.2 Outline of Proof

We explain briefly the intuition behind the algorithm, using the Ordinary Differential Equation (ODE) approach for easy illustration and only supply pointers to related proofs already in the literature.

Suppose we define the \(\mathcal{R}\)-valued function \(\tilde{V}_i(W, u)\) (for any \(W \in \mathcal{R}^a\) and \(u \in U(i)\)) as follows:

\[
\tilde{V}_i(W, u) \triangleq E\{K(i, u, \eta(i, u)) + \alpha W_{\eta(i,u)}\}.
\]

It is easy to see that, for a given stationary admissible policy \(\pi\), the vector function \(\tilde{V}(W, \pi) = (\tilde{V}_1(W, \pi_1), \tilde{V}_2(W, \pi_2), \ldots, \tilde{V}_s(W, \pi_s))^T\) is in fact the appropriate constrained DP operator \(T_\pi\). Also, (2.36) may be written as

\[
V_i(n + 1) = T_i(V(n)) = \min_{u \in U(i)} \tilde{V}_i(V(n), u).
\]

Recall that the fixed point of the operator \(T_\pi\), \(V_\pi = T_\pi(V_\pi)\) is the solution of the Poisson Equation:

\[
V_\pi(i) = E\{K(i, \pi_i, \eta(i, \pi_i)) + \alpha V_\pi(\eta(i, \pi_i))\},
\]

(2.42)

where \(V_\pi(k)\) represents the \(k\)-th element of the vector \(V_\pi\).
2.2 Solution of MDPs using Simulation-based Value Iteration

Using a proof technique resembling that of Lemma 2 above, for any \( n \geq 0 \) the asymptotically stable ODE that the faster recursion (2.40) tracks is, for \( t \geq 0 \)

\[
\dot{\tilde{V}}_i^r(t) = \tilde{V}(V(t), \pi^r_i(t)) - \tilde{V}_i^r(t).
\]  

(2.43)

Note that the terms \( \pi^r(t) \) and \( V(t) \) are quasi-static at the time scale corresponding to (2.40) as recursions (2.39) and (2.38) proceed on slower timescales (cf. (2.41)). Hence we may replace \( \pi^r(t) \) with \( \pi^r \) and \( V(t) \) with \( V \), respectively. The asymptotically stable equilibrium point of (2.43) would be

\[
\tilde{V}_i^r = \tilde{V}(V, \pi_i^r)
\]  

(2.44)

By virtue of (2.41) again, (2.40) would be seen by (2.39) (and (2.38)) as essentially equilibrated. Using the iterates \( \tilde{V}_i^r((n+1)L) \), a better estimate \( \tilde{\pi}(n+1) \) to the policy \( \pi(n+1) \) of (2.37) will be produced via the SPSA update rule of (2.39).

We note that the SPSA based estimate in (2.39) performs correct gradient descent towards \( \pi_i(n+1) \) (the proof technique for such a claim resembles Theorem 1 above). This implies that (2.39) tracks the asymptotically stable (projected) ODE:

\[
\dot{\tilde{\pi}}_i(t) = \hat{P}_i(-\nabla_u \tilde{V}_i(V, u)),
\]  

(2.45)

whose equilibrium point is the desired minimum. Projection \( \hat{P}_i = (\hat{P}^j_i, 1 \leq j \leq N_i)^T \) is as in (2.16) above for any bounded, continuous and real-valued function \( v \). Further, (2.38) views recursions (2.39) and (2.40) as having converged and so considers an estimate to \( V_i(n+1) = T_i(V(n)) \) as available at its \( (n+1) \)–th update. Note that recursion (2.38) has no direct use for the \( \tilde{V}_i^r((n+1)L) \) terms of (2.40) and instead requires an approximation to \( T_i(V(n)) \). However, we can employ the iterates \( \tilde{V}_i^r \) in (2.38) through the term \( \tilde{V}_i(n+1) \), as we describe below. The novelty here is that this role of iterates \( \tilde{V} \) is in addition to the role played as ‘critic’ for recursion (2.39).

Note that iterate \( \tilde{\pi}(n+1) \) does approximate \( \pi(n+1) \) (by virtue of (2.45)), and we
may use $\tilde{V}(V(n), \tilde{\pi}(n+1))$ as $\tilde{V}_i(n+1)$ in (2.38). However, we would need additional simulation to estimate the former term. This we avoid by considering the quantities $\tilde{V}_i(V(n), \tilde{\pi}^r(n))$, for $r = 1, 2$, approximated by the iterates $\tilde{V}_i^r((n+1)L)$. Using Taylor series expansion (as in the proof of Theorem 1 above), we see that

$$V_{\tilde{\pi}^1(n)}(i) + V_{\tilde{\pi}^2(n)}(i) = 2V_{\tilde{\pi}(n)}(i) + O(\delta)$$

and use this fact in (2.38). The non-zero $O(\delta)$ term does not impede convergence (cf. (2.27) in Theorem 1 above). Further, it is an estimate of $\tilde{V}(V(n), \tilde{\pi}_i(n))$ that is available to us and not the more recent $\tilde{V}(V(n), \tilde{\pi}_i(n+1))$. This represents the loss in accuracy due to not estimating $\tilde{V}(V(n), \tilde{\pi}(n+1))$ separately by simulation.

2.3 Numerical Experiments

2.3.1 Setting and Parameters

We consider the continuous time queuing model for flow control in communication networks of Bhatnagar et al. (2001b). A single bottleneck node is fed with two arrival streams, one an uncontrolled Poisson stream and the other a controlled Poisson process. Service times are assumed i.i.d. with exponential distribution. We assume that the node has a buffer size $B < \infty$. Given a constant $T > 0$, we assume that the continuous-time queue length process $\{q_t, t > 0\}$ at the node is observed every $T$ instants and on the basis of this information the controlled source tunes the rate at which it sends packets so as to minimize a certain cost. Suppose $q_n$ denotes the queue length observed at time $nT, n \geq 0$. The controlled source thus sends packets according to a Poisson process with rate $\lambda_c(q_n)$ during the time interval $[nT, (n+1)T]$. The rate is then changed to $\lambda_c(q_{n+1})$ at instant $(n+1)T$ upon observation of state $q_{n+1}$ - we assume for simplicity that there is no feedback delay. The aim then is to find a stationary optimal rate allocation policy that minimizes the associated infinite horizon average cost. We compare performance of the algorithms in this setting.
Initially, we choose $B = 50$ and take the compact action set $U(i), \forall i \in S$ (where $\lambda_c(i) \in U(i)$) to be the interval $[0.05, 4.5]$ for the algorithms ACA-$r$. In the finite action setting, the algorithms RPAFA-$r$ compute the optimal stationary randomized policy over the discrete action set, $U(i), \forall i \in S$, consisting of evenly spaced values within the action set for ACA-$r$. Thus, for each state $i$, $U(i)$ is the set $\{0.05, 1.1625, 2.275, 3.3875, 4.5\}$ and the probability of applying action $u_i^k, 0 \leq k \leq 4$ is obtained from the vector $(\pi_i^0, \pi_i^1, ..., \pi_i^4)$, where $\pi_i^k \geq 0, \forall k$ and $\sum_{k=0}^4 \pi_i^k = 1$. For DPAFA-$r$, all deterministic policies $\pi$, when applied, belong to a similar discrete action set as RPAFA-$r$, i.e.,

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

The uncontrolled traffic rate $\lambda_u$ is chosen as 0.2 and the service rate $\mu$ for incoming packets is set to 2.0. For a system operating under a stationary policy $\pi$, we use the cost function

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

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

The value of $\delta$ needed in (2.4) is set to 0.1. We arbitrarily set the initial policy as $(\pi_i(0) = 2.275, \forall i \in S)^T$ for ACA-$r$ and DPAFA-$r$ and the initial stationary randomized policy to the uniform distribution $(\pi_i^k(0) = 0.2, 0 \leq k \leq 4)^T$ for each state $i \in S$ for RPAFA-$r$. The value of $L$ in all algorithms is taken to be 100 and the reference state (in the algorithms) $i_0$ is 25.

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

$$err_n = \max_{i \in S, 1 \leq k \leq 50} |\pi_i(n) - \pi_i(n - k)|$$
where \( \pi(n) \) is the policy obtained from the \( n \)-th update. Similarly, convergence for the RPAFA-\( r \) algorithms was measured using

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

The step-size sequences \( \{c(n)\} \) and \( \{b(n)\} \) needed in (2.4)-(2.5) were chosen as

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

### 2.3.2 Simulation Results for ACA-\( r \), RPAFA-\( r \) and DPAFA-\( r \)

We stop algorithms ACA-\( r \) and DPAFA-\( r \) when \( err_n \leq 0.1 \). This is achieved within \( n = 2.2 \times 10^3 \) policy updates for both ACA-\( r \) and DPAFA-\( r \). Note that for the given settings, maximum possible \( err_n \) is 4.45. In Figure 2.1 we plot the converged rates for \( T = 5, 10 \) and 15 respectively for ACA-2. We observe that for given \( T \), the source rates are inversely related to the queue length values since the cost function imposes a high penalty for states away from \( \frac{B}{T} \). Further, the difference between the highest and lowest rates decreases as \( T \) increases since for lower values of \( T \), the controller has better control over the system dynamics than when \( T \) is large.

In Figure 2.2 we show the convergence plot for the source rates corresponding to states 0, 15, 25, 35 and 50 for ACA-2 with \( T = 5 \). In Figure 2.3 we plot the source rates with highest probability of selection as given by the converged randomized policy computed by RPAFA-2 for \( T = 5, 10 \) and 15 respectively. The value of \( err_n \leq 0.1 \) is achieved within \( 1.3 \times 10^4 \) policy updates for both RPAFA-1 and RPAFA-2. In Figure 2.4 we show the convergence plot for the probability of selection of source rates corresponding to state 25 for RPAFA-2 with \( T = 5 \). In Figure 2.5 we plot the converged rates for \( T = 5, 10 \) and 15 respectively for DPAFA-2. Similar behaviour was observed in the case of algorithms that use one simulation.

We also compute the optimal policy for the case where there are multiple sources feeding the bottleneck node. In a simulation where three controlled sources feed the node
instead of one, Figure 2.6 depicts the optimal policy of one among the sources, computed using ACA-2. The action sets $U(i)$ for all three sources are taken as $[0.015, 1.8]$. Figure 2.7 plots the net arrival rate (from all three sources) into the bottleneck node. This rate, being the sum of contributions from each of the sources, takes values in $[0.045, 5.4]$. The individual source rates show here a trend that is similar to the single-source case, but the policy graph obtained is not as smooth. This is because the algorithm cannot detect contributions to the net source rate from individual sources. However, as expected, the plot for the net source rate (sum of the source rates from all three sources in an interval) in Figure 2.7 is similar to that of the single-source case (Figure 2.1).

The optimal differential-cost function $h^*(i)$, obtained using the converged policies, is shown in Figures 2.8, 2.9 and 2.10 using algorithms ACA-2, RPAPA-2 and DPAPA-2 respectively, for $T = 5, 10$ and $15$ in each. The function is computed as follows: After convergence of the policy vector in (2.4), the inner recursion (2.5) is run with $b(n) = \frac{1}{n}$, for $2 \times 10^4$ iterations using source rates obtained from the converged policy $\pi^*$ found earlier. The $h^*$ curves for RPAPA-2 and DPAPA-2 have higher values than ACA-2 because of the significantly broader range of actions using which the latter algorithm operates.

The long run average costs computed using the converged policies of all three algorithm types, for three values of $T$, are indicated in Table 2.1. Note that the average costs for the discrete action setting are higher than those for the compact action case, again due to the wider range of actions in the compact action case. Further, among the algorithms with discrete action sets, DPAPA-$r$ algorithms have lower average costs compared to RPAPA-$r$.

**Comparison with other Actor-Critic algorithms**

Next we compare the performance of algorithms DPAPA-$r$ and RPAPA-$r$ with the average-cost actor-critic policy iteration algorithms given by Konda and Borkar (1999). In Table 2.2, AC-4, AC-5, and AC-6 stand for the relevant algorithms 4, 5 and 6 respectively, of Konda and Borkar (1999). The critic recursion employed for all the (above) three algorithms is (2.46) which resembles (2.5) except that its estimates $h(n)$ approximate the differential-cost function for the policy estimate $\pi(n)$ rather than the perturbed policies.
\( \pi^r(n) \) and do not perform the additional \( L \)-step averaging. Also, the \( \pi(n) \) are randomized policies in AC-4, AC-5, and AC-6, respectively. The actor recursions vary for each algorithm, they are (2.47) for AC-4, (2.48) for AC-5 and (2.49) for AC-6, respectively. In these recursions, \( e^l_i \) is an \( N_i \)-dimensional unit vector with 1 in the \( l \)-th place and the \( S \)-valued random variable \( \bar{\eta}_n(i, u^l_i) \) represents the state to which the system transits when action \( u^l_i \) is taken in state \( i \). Note that algorithms AC-5 and AC-6 require apriori choices of certain parameters used in the policy updates. In particular, AC-5 (cf. (2.48)) needs a zero-mean noise required to push the estimates away from the boundary of the simplex - \( \psi^l_i \) - which in this case is chosen to be the uniform distribution \( U(-0.5, 0.5) \). Simulation results for AC-5 under a somewhat different noise condition have been presented by Borkar and Konda (2004) for the discounted cost case. Similarly, AC-6 operates by sampling the discrete action set using a Boltzmann distribution scheme in which the inverse-temperature weights, \( \pi^l_i(n) \), represent the policy. In this experiment, these weights are projected to within \([-10.0, 10.0] \) via the operator \( P \).

\[
\begin{align*}
&h_i(n + 1) = (1 - b(n))h_i(n) + b(n)(K(i, \xi_n(i, \pi_i(n))), \eta_n(i, \xi_n(i, \pi_i(n)))) \\
&- h_{i_0}(n) + h_{i_0(i, \xi_n(i, \pi_i(n)))}(n) \tag{2.46}
\end{align*}
\]

\[
\begin{align*}
\hat{\pi}_i(n + 1) &= \hat{P}(\hat{\pi}_i(n) + c(n) \sum_{l=1}^{N_i} (K(i, u^0_i, \bar{\eta}_n(i, u^0_i)) - K(i, u^l_i, \bar{\eta}_n(i, u^l_i))) \\
&+ h_{\bar{\eta}_n(i, u^0_i)}(nL) - h_{\bar{\eta}_n(i, u^0_i)}(nL)e^l_i) \tag{2.47}
\end{align*}
\]

\[
\begin{align*}
\hat{\pi}_i(n + 1) &= \hat{P}(\hat{\pi}_i(n) + c(n) \sum_{l=1}^{N_i} \{ (h_{i_0}(nL) + h_i(nL) - K(i, u^l_i, \bar{\eta}_n(i, u^l_i))) \\
&- h_{\bar{\eta}_n(i, u^l_i)}(nL)) \hat{\pi}^l_i + \psi^l_i(n) \} e^l_i) \tag{2.48}
\end{align*}
\]
2.3 Numerical Experiments

\[ \pi_i(n + 1) = P(\sum_{l=1}^{N_i} \pi_l^n(n) + c(n)(h_{i0}(nl) + h_i(nl) - K(i, u_i^L, \eta_n(i, u_i^L)) + h_{i0}(nl))e_l^i) \]  
(2.49)

Analogy with Perturbation Analysis

We motivate an analogy between the Continuous Sensitivity term in the Perturbation Analysis of [Cao, 2003, eq. (20)] and Algorithm 4 of [Konda and Borkar, 1999] (eq. (2.47) above). To reproduce using notation from Cao (2003), continuous sensitivity is captured by the equation:

\[ \frac{d\eta}{d\delta} = \pi(Qg + h), \]  
(2.50)

where \( \eta \) is the average-cost obtained employing policy \( \theta \) (every policy \( \theta \) has an associated matrix \( P \)). Take \( P' \) as another transition probability matrix for some feasible policy \( \theta' \), then \( Q = P' - P \). Here \( \delta \) is change in the direction of \( P' \), i.e., choosing a policy corresponding to matrix \( P + \delta(P' - P) \). Also, \( \pi \equiv \pi(\theta) \) is the stationary probability vector corresponding to \( \theta \) whilst \( g \) is the differential-cost vector. Vector \( h \) is the difference in the one-step transition costs, \( h = K(\theta') - K(\theta) \).

We now change notation appropriately and compare with [Konda and Borkar, 1999] for the discrete action case. Consider here the action set \( U_i \equiv \{u_i^0, ..., u_i^l, ..., u_i^{N_i}\} \) for the state \( i \in S \). Also, let \( \bar{\pi}(n) \) be the policy iterate where \( \bar{\pi}^{N_i}_i(n) = 1 \) (policy that w.p. 1 takes action \( u_i^{N_i} \) in state \( i \)) whilst retains the structure of \( \pi \) elsewhere i.e. \( \bar{\pi}_j^l(n) = \pi_j^l(n) \), for \( j \neq i \) and actions \( u_j^l \in U_j \). As a consequence \( \bar{P} \equiv P(\bar{\pi}(n)) \) differs from \( P \equiv P(\pi(n)) \) only in row \( i \), just as expected one-step cost vector \( K(\bar{\pi}(n)) \) differs from \( K(\pi(n)) \) only in the \( i \)-th element. Thus expanding on (2.50), we have

\[ \frac{d\eta}{d\delta_i} = q(\pi(n)) \cdot ((P(\bar{\pi}(n)) - P(\pi(n))) \cdot h(\pi(n)) + K(\bar{\pi}(n)) - K(\pi(n))). \]

Here, \( \delta_i \) corresponds to an infinitesimal change in probability weights \( \pi_i^l(n) \) towards \( u_i^{N_i} \) in state \( i \), \( \forall i \in S \). Also, \( q(\pi(n)) \) is the vector of stationary probabilities corresponding
to policy \( \pi(n) \). The two-timescale stochastic approximation framework assumes that the recursion tasked with computing \( h(\pi(n)) \) as having equilibrated at step \( n + 1 \), hence the use of \( h(\pi(n)) \) directly.

Now note that the special structure of \( P(\bar{\pi}(n)), P(\pi(n)), K(\bar{\pi}(n)) \) and \( K(\pi(n)) \) results in cancellations, to give

\[
\frac{d\eta}{d\delta_i} = q_i(\pi(n)) \cdot \left( \sum_{j \in S} (P_{i,j}(\pi'(n)) - P_{i,j}(\pi(n))) \cdot h_j(\pi(n)) \right) + q_i(\pi(n)) \cdot (K(i, \pi'_i(n)) - K(i, \pi_i(n)))
\]

where \( P_{i,j} \) corresponds to the \((i, j)\)-th element of the matrix. Thus, in a simulation-based recursion - dropping the scale-factor \( q_i(\pi(n)) \) - this corresponds to:

\[
\nabla_i h(\pi(n)) \approx h_n(\bar{\eta}(i, u_i^{N_i})) - h_n(\bar{\eta}(i, u_i^1)) + K(i, u_i^{N_i}, \bar{\eta}(i, u_i^{N_i})) - K(i, u_i^1, \bar{\eta}(i, u_i(n))),
\]

where \( u_i^1 \) corresponds to the action sampled from policy \( \pi_i(n) \). Now note that this is analogous to eq. (2.47) above.

**Observations on Performance**

We compute certain steady-state performance metrics for comparison. We define \( \tilde{B} = [\frac{B}{2} - 1, \frac{B}{2} + 1] \) and denote \( P(\tilde{B}) \) as the stationary probability of the queue being in the states \( \{\frac{B}{2} - 1, \frac{B}{2}, \frac{B}{2} + 1\} \). Also, \( \sigma^2 \) indicates the variance of the queue size about the mean queue length \( E[q_n] \), given that the system is operating under the converged policy. The quantity \( err_n \) is also tabulated. Note that the maximum possible \( err_n \) in the discrete action setting is \( \sqrt{|1|} = 1.414 \). The varying number of policy updates \( n \) for each algorithm is justified as follows: we run \( AC-r, r = 4, 5, 6 \) for \( L \times 10^4 = 10^6 \) updates, so as to match RPAFA-1 in the number of function evaluations. For the same reason, RPAFA-2 (resp. DPAFA-2) is run for half the number of policy updates as RPAFA-1 (resp. DPAFA-1), resulting in RPAFA-1 and RPAFA-2 (resp. DPAFA-1 and DPAFA-2) taking roughly the same time to perform the prescribed number of policy updates.
It is observed that both DPFAA-r and RPAFAA-r display lower average cost ($\lambda^*$) than the three AC-r algorithms, although the value of $err_n$ is lower (in fact, $\leq 10^{-2}$) for the latter algorithms. As borne out in the earlier results (cf. Table 2.1), DPFAA-r performs better than RPAFAA-r. The computation times $t$ shown are measured on a Pentium III computer. Though comparable in the number of function evaluations and simulation of the critic recursion (2.46), the three AC-r algorithms take more than an order of magnitude time than all our algorithms. This difference is contributed to by the additional simulation effort needed in the policy update step of the AC-r algorithms, i.e., in simulating the $\tilde{\eta}_n(\cdot,\cdot)$ terms in each one of (2.47), (2.48) and (2.49), respectively. Note that the $\sigma^2$ values obtained using AC-4 and AC-5 are comparable to the values obtained using our algorithms. However, this is mainly because the mean queue lengths in our algorithms are higher as these are closer to their target means than those of the AC-r algorithms. Inspite of the above, AC-6 has a significantly higher $\sigma^2$ (as compared to our algorithms). Moreover, our algorithms show significantly better performance in terms of all other performance metrics described.

Next, we increase the size of the state space and use the $TD(0)$ variant of ACA-2. Table 2.3 indicates, for the $B = 500$ case, the average costs obtained after 500 updates of the policy vector, using a polynomial function approximation of order $K - 1$ to the differential-cost i.e., $\phi_k(i) = \left(\frac{i - i_0}{i_0}\right)^k$, $0 \leq k \leq K - 1$. The times required for each algorithm are also indicated. The computation times are seen to be roughly linear in $K$ but in general higher than ‘exact’ ACA-2 since the function-approximation procedure is computationally intensive. In particular, every step of the critic’s computation (cf. (2.31)) involves a size-$K$ vector addition and dot-product on top of the simulation effort required in simulating transitions out of each state. The advantage of the approximation method, however, lies in the diminished storage requirement. Since the policy vector is also of size $B$, differential-cost function approximation operates using roughly one-third of the memory space required for the ‘exact’ algorithms.

Using $K = 4$ and $B = 2000, 5000$ and $10,000$, respectively, the average cost obtained using polynomial approximation of differential-cost in ACA-2 is indicated in Table 2.4.
This is compared with the cost obtained using ‘exact’ ACA-2. The average cost in these cases is found to be higher, since the policy obtained uses only approximations of the \( h \) function for gradient-finding. The number of iterations and time required for convergence of all the algorithms, when implemented using the approximation method with \( K = 4 \) and \( B = 50 \), is shown in Table 2.5.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( \lambda^*(T=5s) )</th>
<th>( \lambda^*(T=10s) )</th>
<th>( \lambda^*(T=15s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACA-2</td>
<td>3.98</td>
<td>5.08</td>
<td>6.18</td>
</tr>
<tr>
<td>ACA-1</td>
<td>4.0</td>
<td>5.09</td>
<td>6.17</td>
</tr>
<tr>
<td>DPAFA-2</td>
<td>4.58</td>
<td>5.95</td>
<td>7.38</td>
</tr>
<tr>
<td>DPAFA-1</td>
<td>4.88</td>
<td>5.96</td>
<td>7.38</td>
</tr>
<tr>
<td>RPAFA-2</td>
<td>5.68</td>
<td>6.29</td>
<td>9.48</td>
</tr>
<tr>
<td>RPAFA-1</td>
<td>5.62</td>
<td>7.17</td>
<td>9.03</td>
</tr>
</tbody>
</table>

Table 2.1: Average Cost \( \lambda^* \) as computed by the proposed algorithms

<table>
<thead>
<tr>
<th>Metric/Polynomial order</th>
<th>Exact</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J^* )</td>
<td>5.68</td>
<td>16.0</td>
<td>14.7</td>
<td>13.8</td>
<td>18.4</td>
<td>18.4</td>
</tr>
<tr>
<td>( t ) sec</td>
<td>176</td>
<td>175</td>
<td>190</td>
<td>204</td>
<td>211</td>
<td>216</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of DPAFA-\( r \) and RPAFA-\( r \) with Actor-Critic Policy Iteration

**2.3.3 Simulation Results for Value-Iteration algorithm**

The details of the continuous time queuing model remain the same as before. The aim here is to find a stationary optimal rate allocation policy that minimizes the associated
2.3 Numerical Experiments

<table>
<thead>
<tr>
<th>$B$</th>
<th>$\lambda^*(\text{Look-up Table})$</th>
<th>$\lambda^*(K = 4)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>6.3</td>
<td>8.7</td>
</tr>
<tr>
<td>5000</td>
<td>7.5</td>
<td>29.1</td>
</tr>
<tr>
<td>10000</td>
<td>8.3</td>
<td>60.1</td>
</tr>
</tbody>
</table>

Table 2.4: $\lambda^*$ using $h$-approximation for large $B$ and $K = 4$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># Iterations</th>
<th>Time in seconds</th>
<th>$\lambda^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPAFA-1</td>
<td>14500</td>
<td>586</td>
<td>10.7</td>
</tr>
<tr>
<td>RPAFA-2</td>
<td>1620</td>
<td>135</td>
<td>10.6</td>
</tr>
<tr>
<td>DPAFA-1</td>
<td>9550</td>
<td>406</td>
<td>9.7</td>
</tr>
<tr>
<td>DPAFA-2</td>
<td>750</td>
<td>70</td>
<td>6.9</td>
</tr>
<tr>
<td>ACA-1</td>
<td>9300</td>
<td>384</td>
<td>8.14</td>
</tr>
<tr>
<td>ACA-2</td>
<td>700</td>
<td>79</td>
<td>5.4</td>
</tr>
</tbody>
</table>

Table 2.5: Comparison of Convergence using $h$-approximation for $B = 50$ and $K = 4$

Infinite horizon discounted cost. We choose $B = 50$ and take the compact action set $U(i)$ (where $\lambda_c(i) \in U(i)$, $\forall i \in S$) to be the interval $[0.05, 4.5]$.

The discount factor $\alpha$ is 0.9. As in the infinite-horizon average cost case, the uncontrolled traffic rate $\lambda_u$ is chosen as 0.2 with the service rate $\mu$ for incoming packets is set to 2.0. For a system operating under a stationary policy $\pi$, we use the cost function

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

The value of $\delta$ needed in (2.39) is set to 0.1. We arbitrarily set the initial value function iterate $V_i(0) = 0$, $\forall i \in S$ and policy iterate as $(\hat{\pi}_i(0) = 2.275$, $\forall i \in S)^T$ for the proposed algorithm. The value of $L$ needed in (2.40) is taken to be 100.

As a metric to characterize the convergence of all algorithms, we measured a quantity $err_V(n)$ at value function update $n$ in the following manner:

$$err_V(n) = \max_{i \in S, 1 \leq k \leq 50} |V_i(n) - V_i(n - k)|.$$

Similarly, convergence in terms of policy is obtained using

$$err_{\pi}(n) = \max_{i \in S, 1 \leq k \leq 50} |\pi_i(n + 1) - \pi_i(n + 1 - k)|.$$
The step-size sequences \( \{a(n)\} \), \( \{b(n)\} \) and \( \{c(n)\} \) needed in (2.38), (2.39) and (2.40), respectively, were chosen as

\[
\begin{align*}
a(0) &= b(0) = c(0) = 1, \\
a(n) &= \frac{1}{n}, \\
b(n) &= \frac{1}{n^{0.5}}, \\
c(n) &= \frac{1}{n^{0.55}},
\end{align*}
\]

\( \forall n \geq 1 \), respectively.

Convergence of the form \( \text{err}_V(n) \leq 0.1 \) and \( \text{err}_\pi(n) \leq 0.1 \) is achieved within 150 and 300 updates of (2.38), respectively. In Figure 2.11a, we plot the converged source rates for \( T = 5, 10 \) and 15 respectively. Similarly, converged value functions (using the \( \text{err}_V \) criterion) are shown in Figure 2.11b. For a given \( T \), the source rates are inversely related to the queue length values since the cost function imposes a high penalty for states away from \( \frac{B}{T} \). As before, the difference between the highest and lowest rates decreases as \( T \) increases.

Convergence for the classical value iteration algorithm (2.36) for \( \text{err}_V(n) \leq 0.1 \),
2.3 Numerical Experiments

is achieved within 100 $V(n)$ updates for all $T$. The value functions obtained therein are shown in Figure 2.12a, which match the corresponding results of the proposed algorithm shown in 2.11b. We made modifications to $U(i)$ to compute the transition probabilities $p(i, u, j), \forall i, j \in S, u \in U(i)$ needed in the RHS of the Poisson Equation $\sum_{j=1}^{S} p(i, u, j)(K(i, u, j) + \alpha V_j(n))$. The discrete action set $U(i), \forall i \in S$, consists of actions spaced at $5 \times 10^{-3}$ within the compact interval $[0.05, 4.5]$, thus making for $|U(i)| = 225, \forall i \in S$. The transition probabilities were computed in a method similar to that of Bhatnagar and Abdulla (2007). On a Pentium 4 computer, classical value iteration (for the above discretized set) took 2 seconds whereas the proposed algorithm took 12 seconds (over the compact action set). This is due to the time required for simulating transitions, and a faster simulator will bring such a comparison closer.

For purposes of comparison, we propose a variant of the algorithm in §6.2 and §6.3 of Van Roy et al. (1997) for the current situation of a look-up table. We first present the

![Figure 2.3: Randomized Stationary Policy computed using RPAFA-2](image)

![Figure 2.4: Convergence Behaviour for RPAFA-2](image)
algorithm and then its proposed variant to suit our framework. The algorithm proposed by Van Roy et al. (1997) simulates a single endless trajectory, i.e. \( i_{n+1} = \eta_n(i_n, u_n(i_n)) \) and updates a coefficient vector that helps approximate the value function for each state \( i \in S \). This coefficient vector is \( r(n) \in \mathcal{R}^K \) where \( K \ll s \), and the approximate value function is given by \( \tilde{V}_i(n) = (\Phi r(n))(i) \), \( \forall i \in S, n \geq 0 \). Here, the \( s \times K \) matrix \( \Phi \) consists of the feature vectors \( \phi(i) \), \( \forall i \in S \) as the rows and is of full rank. The \( i \)-th element of the \( s \times 1 \) vector \( \Phi r(n) \) is indicated by \( (\Phi r(n))(i) \). Below is the algorithm of §6.2 of Van Roy et al. (1997):

\[
    r(n+1) = r(n) + a(n)\phi(i_n)(K(i_n, u_n(i_n), i_{n+1}) + \alpha(\Phi r(n))(i_{n+1})) - \phi r(n)(i_n), \quad (2.51)
\]
where the decision \( u_n(i_n) \) is the ‘greedy’ decision, i.e.

\[
  u_n(i_n) = \arg\min_{u \in U(i_n)} E\{K(i_n, u, \eta(i_n, a)) + \alpha(\phi_r(n))(\eta(i_n, u))\}. \tag{2.52}
\]

Now, consider the value function iterates \( V(n) \) in place of the coefficient vector \( r(n) \) in (2.51). Also make the feature vector \( \phi(i) \), \( \forall i \in S \) equal to the unit vector \( e_i \in \mathcal{R}^s \), then the algorithm (2.51) reduces to:

\[
  V(n+1) = V(n) + a(n) \left( \sum_{i=1}^{S} \{K(i, u_n(i), \eta(i, u_n(i))) + \alpha V_{\eta(i,u_n(i))}(n)\}e_i - V(n) \right), \tag{2.53}
\]

where recall that \( u_n(i) \) is just the \( \pi_i(n+1) \) in (2.37). Note that whereas (2.53) requires \( u_n(i) \) for all \( i \in S \), (2.51) uses only \( u_n(i_n) \). Using the fact that \( \Phi^T \Phi \) is the identity matrix, one can see that (2.53) is a variant for the more recent algorithm for approximate fixed point computation given by Choi and Van Roy (2006) as well, which employs a Kalman
filter approach. In most simulation based settings, the term $u_n(i)$ can only be identified by simulating transitions out of $i$ using all $u \in U(i)$, possibly averaging over multiple transitions for the same action $u \in U(i)$. Note that (2.53) corresponds to a stochastic approximation implementation of the operator $T$, using the diminishing stepsize $a(n)$. Van Roy et al. (1997, §6.3) suggests that a greedy selection like (2.52) does not work in practice and exploration of $U(i)$ is employed. Further, de Farias and Van Roy (2000) theoretically formalize this using a $T_B$ operator which employs a Boltzmann distribution to select $u_n(i_n)$ from $U(i_n)$. We implement the variant of the algorithm of Van Roy et al. (1997), using the exploration suggested by de Farias and Van Roy (2000), and discretizing each $U(i)$ into 45 equally spaced actions. The algorithm uses $a(n) = \frac{1}{n+6}$ and requires roughly 4000 updates and almost 800 seconds in all cases of $T$ for the $V(n)$ vector to converge to $err_V \leq 0.1$. Figure (2.12) plots the value function iterates obtained after convergence.
2.3 Numerical Experiments

![Plots of (a) converged policies and (b) value functions](image)

Figure 2.11: Plots of (a) converged policies and (b) value functions

![Value Function: Exact Value Iteration](image)

Figure 2.12: $V$ iterates from (a) classical VI and (b) modified TD(0)
Chapter 3

Algorithms for Finite Horizon MDPs

Abstract

In §3.1, we develop three simulation-based algorithms for finite horizon Markov decision processes (FH-MDPs). The first algorithm is developed for finite state and compact action spaces while the other two are for finite state and finite action spaces. Convergence analysis is briefly sketched.

In §3.2, we concentrate on methods to mitigate the curse of dimensionality that affects FH-MDPs severely, as there is one probability transition matrix per stage. We propose two parametrized actor-critic algorithms for FH-MDPs with compact action sets, the ‘critic’ in both algorithms learning the policy gradient. We show w.p 1 convergence to a set with the necessary condition for constrained optima. Further, a third algorithm for stochastic control of stopping time processes is presented.

In §3.3, illustrative numerical experiments with the three algorithms of §3.1 are shown for a problem of flow control in communication networks. Similarly for §3.2, simulation results on the same setting attest to the performance advantages of all three proposed algorithms.
3.1 Simulation–Based Optimization Algorithms for Finite Horizon MDPs

Reinforcement learning algorithms have generally been developed and studied in the infinite horizon MDP framework of Bertsekas and Tsitsiklis (1996), Sutton and Barto (1998), Van Roy (2001) under the discounted cost or the long-run average cost criteria. For instance, TD learning, Q-learning, actor critic algorithms etc., have been developed in the infinite horizon framework. However, in many real life scenarios, finite horizon decision problems assume serious significance. For instance, in the design of a manufacturing fab, one requires planning over a finite decision horizon (cf. Bhatnagar et al. (1999), Chang et al. (2003)). 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.

As an example of application, 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 modelled as an MDP. There even exists one such proprietary scheme for ATM networks, termed Cisco ForeSight (CFS) (cf. (Chin, 2004, pp. 118-119)) - our choice of the queue sampling period $T = 5s$ 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. Yet there is a possibility that such a burst will swamp the router queue. A router’s GTS module could control this behaviour of a source, by knowing typical token-bucket replenish times.

• Albeit on a slower time-scale than the above burst-stream scenario, the time localized behaviour 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, 6 PM to 9 AM) via a suitable finite-horizon GTS module at the router.

Most of the work on developing computationally efficient algorithms for finite horizon problems, however, assumes that model information is known. For instance, Serin (1995) formulates the problem of solving a finite horizon MDP under partial observations as a nonlinear programming problem 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 by Lin et al. (2004). Zhang and Baras (2000) propose a hierarchical structure using state aggregation for solving finite horizon problems.

We develop two-timescale simulation-based actor-critic algorithms for finding optimal policies in finite horizon MDPs (FH-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. Garcia and Ndiaye (1998) develop three variants of the Q-learning algorithm for the finite horizon problem assuming lack of model information. However, the FH-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 Garcia and Ndiaye (1998). In
particular, we do not embed the FH-MDP into an infinite horizon one. The solution
procedure that one obtains using the approach of Garcia and Ndiaye (1998) is at best
only approximate whereas our solution procedure does not involve such approximations.
The duration $T$, of the decision horizon remains fixed in our case. The aim at each stage
$i, i = 0, 1, ..., T − 1$ is to find the optimal decision rule for that stage 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 $\pi$,
or a proxy thereof, at each update step. We show that in the limit as the number of
updates goes to infinity, our algorithms converge 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 of Spall
(1992), and the one-simulation version that uses Hadamard matrix based perturbations
(proposed by Bhatnagar et al. (2003)).

Comparison is also due with respect to algorithms in Tadepalli and Ok (1998). At their
most advanced, algorithms in Tadepalli and Ok (1998) use Dynamic Bayesian Networks
(DBNs) to store a model of the MDP. Then, local regression is used to infer the value-
function of the states in a vicinity. This latter scheme is analogous to Ormoneit and Sen
(2002) employing mother kernel functions for a region of states. Representation of the
Transition Probability Matrix is, in fact, avoided in reinforcement learning algorithms.
The method of H-Learning of Tadepalli and Ok (1998), which is not mere policy eval-
uation, is compared with TD($\lambda$) of Tsitsiklis and Van Roy (1997). A handicap of the
local regression method is that piecewise linearity is lost when discounted-cost MDPs are
considered.

The proposed algorithms for MDPs with compact action sets perform a search in
the space of deterministic policies. The algorithms in §3.2 later address the key concern
that the size of both the policy and cost-to-go table require at least $|S| \times T$ in storage.
This is an overhead compared to infinite-horizon MDPs where only size $|S|$ tables are needed, indicating that the memory requirements in FH-MDPs assume severity. 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, respectively. 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.

In §3.3, we illustrate numerical experiments in the framework of flow control in communication networks wherein we compute the optimal costs and policies using these algorithms and compare the performance of the various algorithms using certain performance metrics. For the finite action case, we also compute these quantities using DP for purposes of comparison. The rest of this section is organized as follows: §3.1.1 provides the framework and notation while algorithms follow in §3.1.2 and §3.1.3. Further, §3.1.4 contains the convergence analysis.

### 3.1.1 Framework and Notation

Consider an MDP \( \{X_r, r = 0, 1, ..., T\} \) now with decision horizon \( T < \infty \). Suppose \( \{Z_r, r = 0, 1, ..., T - 1\} \) be the associated control valued process. Decisions are made at instants \( r = 0, 1, ..., T - 1 \), and the process terminates at instant \( T \). Let state space at epoch \( r \) be \( S_r, r = 0, 1, ..., T \) and let the control space at epoch \( r \) be \( C_r, r = 0, 1, ..., T - 1 \). Note that \( S_T \) is the set of terminating states of this process. Let \( U_r(i_r) \subset C_r, r = 0, 1, ..., T - 1 \), be the set of all feasible controls in state \( i_r \), in period \( r \). Let \( p_r(i, a, j), i \in S_r, a \in U_r(i), j \in S_{r+1}, r = 0, 1, ..., T - 1 \) denote the transition probabilities associated with this MDP. The transition dynamics of this MDP is now governed according to:

\[
P(X_{r+1} = i_{r+1}|X_r = i_r, Z_r = a_r, ..., X_0 = i_0, Z_0 = a_0) = p_r(i_r, a_r, i_{r+1}),
\]

\( r = 0, 1, ..., T - 1 \), for all \( i_0, i_1, ..., i_T, a_0, a_1, ..., a_{T-1} \), in appropriate sets. We define an admissible policy \( \pi \) as a set of \( T \) functions \( \pi = \{\mu_0, \mu_1, ..., \mu_{T-1}\} \) with \( \mu_r : S_r \rightarrow C_r \) such that \( \mu_{r,i} \in U_{r,i}, \forall i \in S_r, r = 0, 1, ..., T - 1 \). Thus at (given) instant \( r \) with the system
3.1 Simulation–Based Optimization Algorithms for Finite Horizon MDPs

In say state $i$, the controller under policy $\pi$ selects the action $\mu_r, i$. Let $K_r(i, a, j)$ 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 $j \in S_{r+1}$, respectively, for $r = 0, 1, ..., T - 1$. Also, let $K_T(k)$ 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 $\pi = \{\mu_0, \mu_1, ..., \mu_{T-1}\}$ that minimizes for all $i \in S_0$,

$$V_0^i(\pi) = E\left\{K_T(X_T) + \sum_{r=0}^{T-1} K_r(X_r, \mu_r, X_r, X_{r+1})|X_0 = i\right\}.$$  \hfill (3.1)

The expectation above is over the joint distribution of $X_1, X_2, ..., X_T$. The dynamic programming algorithm for this problem is now given as follows (cf. Bertsekas (1995)): for all $i \in S_T$,

$$V_T^i(\pi) = K_T(i)$$ \hfill (3.2)

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

$$V_r^i(\pi) = \min_{a \in U_r(i)} \left\{\sum_{j \in S_{r+1}} p_r(i, a, j)(K_r(i, a, j) + V_{r+1}^j(\pi))\right\},$$ \hfill (3.3)

respectively.

**An Overview and Motivation**

Note that using dynamic programming, the original problem of minimizing, over all admissible policies $\pi$, the $T$-stage cost-to-go $V_0^i(\pi), \forall i \in S_0$, as given in (3.1), is broken down into $T$ coupled minimization problems given by (3.2)-(3.3) with each such problem defined over the corresponding feasible set of actions for each state. Here, ‘coupled’ means that the solution to the $r-$th 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 $p_r(i, a, j)$. Here, 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 epoch \( r (r \in \{0, 1, \ldots, T - 1\}) \) is \( i \) and action \( a \) is picked, we assume that the next state \( j \) can be obtained through simulation, even though \( p_r(i, a, j) \) 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 (described previously in \( \S 2.1.2 \)) to obtain simulation-based actor-critic algorithms that solve all the \( T \) coupled minimization problems \((3.2)-(3.3)\), under the (above) lack of model information constraint. For the case of infinite horizon problems, Konda and Borkar (1999), Konda and Tsitsiklis (2003) and Bhatnagar and Kumar (2004) use two-timescale stochastic approximation algorithms. As explained in \( \S 1 \), these algorithms are the simulation based analogs of the policy iteration algorithm. The algorithms of Konda and Borkar (1999) and Bhatnagar and Kumar (2004) 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), and updating the policy itself along the slower scale. The algorithm of Konda and Tsitsiklis (2003) 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. The construction of deterministic perturbation matrices \( H \equiv H_{P \times P} \) of dimension \( P \times P \) is described in \( \S 2.1.2 \) and we use the same in the proposed algorithms. Using such matrices, the space of perturbation vectors has only \( P = 2^{\lceil \log_2(N+1) \rceil} \) elements as against \( 2^N \) 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 \((3.2)-(3.3)\), in the finite horizon setting, involves \( T \) coupled minimization problems with the \( r \)th 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 (3.2)-(3.3) 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 viz., compact and finite action sets, respectively. In the following, for notational 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 respectively denote the state and control spaces. 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.

### 3.1.2 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 \( U_i = \prod_{j=1}^{N}[a_i^j,b_i^j] \). State \( i \) or stage \( r \) may influence the value of \( N \) (we however consider it fixed).

**Assumption 1** For all \( i,j \in S, a \in U_i \), both \( K_r(i,a,j) \) and \( p_r(i,a,j) \) are continuously differentiable w.r.t. \( a \). Also, \( |K_T(i)| < \infty \), \( \forall i \in S \).

An admissible policy is the vector \( \pi^\triangle_r(\mu_{r,i}, i \in S, r = 0,1,...,T-1)^T \), 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 \( P_i^j(y) = \min (b_i^j, \max (a_i^j, y)) \), \( y \in R \), the projection of \( y \) onto the interval \( [a_i^j,b_i^j] \), \( j = 1,...,N \). Also, for any \( z \in R^N \), say \( z = (z^1,...,z^N)^T \), let \( P_i(z) = (P_i^1(z^1),...P_i^N(z^N))^T \). Then \( P_i(z) \) denotes the projection of \( z \) onto the set \( U_i \), \( i \in S \) analogous to the \( P_i \) of \([2.1.6]\). Let \( P = (P_i, \forall i \in S)^T \) and note that, in general, different \( P_{r,i}(\cdot) \) are possible. We shall use \( P_i(\cdot) \) (resp. \( P(\cdot) \)) as the projection operators in our algorithms, to project each action update (resp. policy update) to the feasible set of actions (resp. policies). To accommodate the non-stationarity of the policy, we add a subscript \( r \) denoting the stage and define policy updates \( \pi(n) \) as \( (\mu_{r,i}(n), \forall i \in S, n \geq 0, 0 \leq r \leq T-1)^T \). In particular, each \( \mu_{r,i}(n) \) is an \( N \)-dimensional vector whose elements are identified as \( (\mu_{r,i}^j(n), 1 \leq j \leq N)^T \). Similarly, let \( \Delta_{r,i}(\cdot) = (\Delta_{r,i}^j(\cdot), 1 \leq j \leq N)^T \) be
the $\{\pm 1\}^N$-valued perturbation vectors in the SPSA estimates that are obtained using appropriate normalized Hadamard matrices of $2.1.2$.

For each $n \geq 0$, let $\{X_r(n), r = 0, 1, \ldots, T\}$, denote a simulated trajectory of the process governed by policy $\pi(n) = (P_i(\mu_r,i(n) + \delta \Delta_{r,i}(n)), i \in S, r = 0, 1, \ldots, T-1)^T$. Here, $\delta > 0$ is a given small constant (we use $\delta = 0.1$ in our experiments). Strictly speaking, the condition $\delta \to 0$ as $n \to \infty$ is needed to show w.p. 1 convergence – although a small $\delta$ does guide the algorithm to a neighbourhood of the equilibrium point. We iterate over $n$, the policies $\pi(n)$ in our algorithms. Let $\{b(n)\}$ and $\{c(n)\}$ be two step-size schedules that satisfy

$$\sum_n b(n) = \sum_n c(n) = \infty, \quad \sum_n b(n)^2, \sum_n c(n)^2 < \infty, \quad (3.4)$$

and

$$c(n) = o(b(n)), \quad (3.5)$$

respectively. Thus $\{c(n)\}$ goes to zero faster than $\{b(n)\}$ does and corresponds to the slower timescale since 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 albeit graceful convergence. Likewise, $\{b(n)\}$ is the faster scale.

Let $\{\eta_{m,r}(i, a)\}, i \in S, a \in U(i), r = 0, 1, \ldots, T-1$ be independent families of i.i.d. random variables such that $\eta_{m,r}(i, a), m \geq 0$ 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 $\eta_{m,r}(i, a)$ would denote the $m$-th sample of the state at the $(r+1)$-th stage to which the system makes a transition. For economy of notation, we shall simply denote by $\eta_{mL+r, m}(i, \tilde{\mu}_{r,i}(n))$ where $\tilde{\mu}_{r,i}(n)$ are components of policy $\pi(n)$. Note that while the acronyms of Chapter 2 are shared, all proposed algorithms are for the finite-horizon case.

**Algorithm for Compact Action Sets (ACA)**

- **Step 0 (Initialize):** Fix $\mu_{r,i}(0), \forall i \in S, r = 0, 1, \ldots, T-1$. Choose integers $L$ and
(large) $M$. 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 $\Delta_{r,i}(0) := \hat{H}(1)$, $\forall i \in S, r = 0, 1, \ldots, T - 1$. Set $\tilde{\pi}_0 = (P_t(\mu_0, i) + \delta \Delta_{r,i}(0))$, $i \in S, r = 0, 1, \ldots, T - 1$. Set $V_{r,i}(q) = 0$, $\forall 0 \leq r \leq T - 1, 0 \leq q \leq L - 1, i \in S$.

- 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 $p_r(i, \tilde{\mu}_{r,i}(n), \cdot)$. Set $V_{T,i}(nL + m) = K_T(i)$, $\forall i \in S$. Next, for all $r = T - 1, T - 2, \ldots, 0$ and $i \in S$, update

$$V_{r,i}(nL + m + 1) := (1 - b(n))V_{r,i}(nL + m) + b(n)(K_r(i, \tilde{\mu}_{r,i}(n), \eta_{nL+m,r}) + V_{r+1,\eta_{nL+m,r}}(nL + m)). \quad (3.6)$$

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

$$\mu_{r,i}(n + 1) := P_i \left( \mu_{r,i}(n) - c(n) \frac{V_{r,i}(nL)}{\delta} (\Delta_{r,i}(n))^{-1} \right) \quad (3.7)$$

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

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

**Remark:** Note that even though 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 $P \geq 1$. Suppose $r$ Poisson (traffic) sources send packets to various nodes in this network. The rates at which these sources send packets is tuned based on the state of the system. The above decisions (on rates of the sources) are made every $T$ instants of time. The queue length at each one of the $P$ nodes is observed every $T$ instants of time and based on this information, the rates of the Poisson sources are tuned. Suppose $Q_k(n)$ and $R_k(n)$, $k = 1, \ldots, P$ 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 is difficult, and – in connection with implementing DP for comparison purposes – we give an account of this in §3.3 below. However, the ‘next’ state $\eta_{r,i}(n, a)$ can be obtained by simply simulating the above network and running it for $T$ time instants with initial state $X_0$ (say $X_0 = (Q_1, R_1, \ldots, Q_P, R_P)$ and action $a$ (say 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 (3.6), termed the ‘critic’, proceed on the faster timescale $b(n)$ (hence converge faster). In contrast, (3.7), known as the ‘actor’, proceeds on the slower timescale $c(n)$. The critic and actor correspond to the policy evaluation and improvement steps respectively of the policy iteration method. The additional averaging (over $L$ iterations) in the critic recursion (3.6) over the two timescale averaging, for a given policy update $\pi(n)$ is seen to improve performance. The index $M$ in the algorithm captures the flexibility of the user stopping the algorithm at a pre-calculated number of iterations - or choosing a convergence criterion to terminate. Next, we present two analogs of this algorithm for the case when action sets are finite.
3.1 Simulation–Based Optimization Algorithms for Finite Horizon MDPs

3.1.3 Finite Horizon Algorithms for Finite Action Sets

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

Assumption 2 Costs \( K_r(i, a, j), i, j \in S, a \in U_i \), are bounded for \( r = 0, 1, \ldots, T - 1 \). Further, \( K_T(i) \) is bounded for all \( i \in S \).

Deterministic Policy Update Algorithm for Finite Action Sets (DPAFA)

Assume all elements of \( U_i \) lie in \( \mathbb{R}^N \) 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 \( \bar{\mu}_{r,i}(n) \) prescribed in any state \( i \) using this algorithm need not be admissible. Hence we project this 'perturbed' action \( \bar{\mu}_{r,i}(n) \) 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 \( \bar{\mu}_{r,i}(n) \), we arbitrarily choose one of them.

Randomized Policy Update Algorithm for Finite Action Sets (RPAFA)

Here we work with randomized policies instead of deterministic ones as in the two cases before. Let \( \pi^j_{r,i} \) be the probability of selecting action \( u^j_i \) in state \( i \) at instant \( r \) and let \( \hat{\pi}_{r,i} \) be the vector \( (\pi^j_{r,i}, i \in S, u^j_i \in \bar{U}_i \backslash \{u^0_i\})^T, r = 0, 1, \ldots, T - 1 \). Thus given \( \hat{\pi}_{r,i} \) as above, the probability \( \pi^0_{r,i} \) of selecting action \( u^0_i \) in state \( i \) at instant \( r \) is automatically specified as \( \pi^0_{r,i} = 1 - \sum_{j=1}^{q} \pi^j_{r,i} \).

A randomized policy \( \hat{\pi} \) is given by \( \hat{\pi} = (\hat{\pi}_{r,i}, i \in S, r = 0, 1, \ldots, T - 1)^T \). 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 \( P : \mathbb{R}^q \leftrightarrow \bar{S} \) denote the projection map that projects \( \hat{\pi}_{r,i}(i) \) to the simplex \( \bar{S} \) after each update of the algorithm below. Note that unlike in ACA, projection \( P \) is independent of \( (r, i) \).
Algorithm RPAFA

- **Step 0 (Initialize):** Fix $\pi_{r,i}(0), \forall i \in S, u_i^j \in \bar{U}_i$, $r = 0, 1, \ldots, T - 1$ as the initial probabilities for selecting actions in state $i$. Set $\hat{\pi}_{r,i}(0) = (\pi_{r,i}(0), i \in S, u_i^j \in \bar{U}_i \setminus \{u_i^0\})^T$, $r = 0, 1, \ldots, T - 1$. 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 $\hat{\Delta}_{r,i}(0) := \hat{H}(1), \forall i \in S, r = 0, 1, \ldots, T - 1$. Set $\hat{\pi}(0) = (P(\hat{\pi}_{r,i}(0) + \delta\hat{\Delta}_{r,i}(0)), i \in S, r = 0, 1, \ldots, T - 1)^T$ 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, $\phi_{r,i}(0) \in \bar{U}_i$, according to policy $\hat{\pi}(0)$. Set $V_{r,i}(q) := 0, \forall 0 \leq r \leq T - 1, 0 \leq q \leq L - 1, i \in S$ 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 $p_r(i, \phi_{r,i}(nL + m), \cdot)$. Set $V_{r,i}(nL + m) = K_T(i), \forall i \in S$. Next, for all $r = T - 1, T - 2, \ldots, 0, i \in S$, update

\[
V_{r,i}(nL + m + 1) := (1 - b(n))V_{r,i}(nL + m) + b(n)(K_r(i, \phi_{r,i}(nL + m), \eta_{nL+m,r})
+ V_{r+1,nL+m,r}(nL + m)). \tag{3.8}
\]

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 $i \in S, r = 0, 1, \ldots, T - 1, n \geq 0$, we have

\[
\hat{\pi}_{r,i}(n + 1) = P \left(\hat{\pi}_{r,i}(n) - c(n)\frac{V_{r,i}(nL)}{\delta}(\hat{\Delta}_{r,i}(n))^{-1}\right) \tag{3.9}
\]

Set $n := n + 1$. If $n = M$, go to Step 3;
else, for all $i \in S, 0 \leq r \leq T - 1$, set $\hat{\Delta}_{r,i}(n) := \hat{H}(n \mod P + 1)$ as the new Hadamard matrix generated perturbation. Set $\hat{\pi}(n) := (P(\hat{\pi}_{r,i}(n) + \delta\hat{\Delta}_{r,i}(n)), i \in S, r = 0, 1, \ldots, T - 1)$ and $\hat{\pi}(n)$ as the new randomized policy.

- **Step 3:** Stop.
$S, r = 0, 1, ..., T - 1)^T$ as the new perturbed randomized policy. Next, for all $r = 0, 1, ..., T - 1$, simulate the action to use above, $\phi_{r,i}(nL + m) \in \bar{U}_i$, 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.

### 3.1.4 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 for the finite action case.

**Convergence Analysis for Compact Action Setting**

The analysis proceeds using the standard ordinary differential equation (ODE) approach for stochastic approximation algorithms (cf. Kushner and Clark (1978), Kushner and Yin (1997)). Conditions (3.4)-(3.5) 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 (3.4). The first condition in (3.4) is required to perform a appropriate 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 (3.5), the two-timescale algorithm in fact tracks the trajectories of a singular ODE. We show the analysis under Assumption $[\text{H}]$. Let $\hat{n} = \lceil \frac{n}{L} \rceil$ denote the integer part of $\frac{n}{L}$. Let $\hat{\pi}_r(n) \equiv \pi_r(\hat{n})$, with elements $\hat{\mu}_{r,i}(n) = \mu_{r,i}(\hat{n})$ for $r = 0, 1, ..., T - 1$, and $\hat{b}(n) = b(\hat{n})$, respectively. We have

**Lemma 3** Under Assumption $[\text{H}]$, the iterates $V_{r,i}(n)$, $r = 0, 1, ..., T - 1$, $n \geq 0$, of the algorithm remain uniformly bounded with probability one.

**Proof:** Note that by Assumption $[\text{H}]$ since $U_i$, $i \in S$, are compact sets and $S$ is finite,
Similarly, \( \sup_{i \in S} |K_T(i)| \triangleq K_T < \infty \). Now note that \( \sup_{i \in S, n \geq 0} |V_{T,i}(n)| = K_T < \infty \). Note that by (3.4), \( \tilde{b}(n) \to 0 \) as \( n \to \infty \). Also, \( \tilde{b}(n) > 0, \forall n \geq 0 \). Thus \( \exists n_0 \) s.t. \( \forall n \geq n_0, \tilde{b}(n) \in (0, 1] \).

From (3.6) and the above for \( n \geq n_0 \), we have

\[
|V_{r,i}(n + 1)| \leq (1 - \tilde{b}(n))|V_{r,i}(n)| + \tilde{b}(n)(|K_r(i, \tilde{\mu}_{r,i}(n), \eta_{n,r})| + |V_{r+1,\eta_{r+1}}(n)|) \tag{3.10}
\]

for \( r = 0, 1, \ldots, T - 1 \). Now for \( r = T - 1 \), for \( n \geq n_0 \), the RHS of (3.10) can be seen to be a convex combination of \( |V_{T-1,i}(n)| \) and a quantity uniformly bounded with probability one by \( \tilde{K} + K_T \). Since \( |V_{T-1,i}(0)| < \infty, \forall i \in S \), it follows that \( \sup_{i \in S, n \geq 0} |V_{T-1,i}(n)| < \infty \).

The above argument can now be repeated successively for \( t = T - 2, T - 3, \ldots, 0 \). Hence the iterates (3.6) remain uniformly bounded with probability one. \( \square \)

For an action \( a \in U_i \) consider the operator:

\[
F_r(i, a, W) \triangleq \sum_{j \in S} p_r(i, a, j)(K_r(i, a, j) + W(j)),
\]

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

Let \( F_{l} = \sigma(\tilde{\mu}_{r,i}(p), V_{r,i}(p), p \leq l; \eta_{p,r}, p \leq l, i \in S, r = 0, 1, \ldots, T - 1), l \geq 1 \), 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

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

We now have

**Lemma 4** The sequences \( \{M_{l,r}(i), l \geq 1\}, i \in S, r = 0, 1, \ldots, T - 1 \), converge w.p. 1.

**Proof:** Note that \( M_{l,r}(i) \) are \( \mathcal{F}_l \)-measurable for all \( l \geq 1 \). Further, it can be seen that

\[
E[M_{l+1,r}(i)|\mathcal{F}_l] = M_{l,r}(i),
\]
with probability one. From (3.4), Assumption 1 and Lemma 3 that \( \{M_l,r(i), F_l\} \) are martingale sequences with almost surely convergent quadratic variation processes. It then follows from Proposition VII.2.3 (c) of Neveu (1975) that \( \{M_l,r(i), l \geq 1\} \) are almost surely convergent sequences. □

Consider the system of ODEs: For all \( i \in S, r = 0, 1, ..., T - 1, \)

\[
\dot{\hat{\mu}}_{r,i}(t) = 0,
\]

(3.11)

\[
\dot{V}_{r,i}(t) = F_r(i, \hat{\mu}_{r,i}(t), V_{r+1}(t)) - V_{r,i}(t),
\]

(3.12)

with \( V_{T,i}(t) = K_T(i), \forall i \in S, t \geq 0. \) Note that (3.11) corresponds to a fixed policy \( \bar{\pi} \) that is independent of \( t. \) Hence, in place of (3.12), consider the ODE

\[
\dot{V}_{r,i}(t) = F_r(i, \bar{\mu}_{r,i}, V_{r+1}(\bar{\pi})) - V_{r,i}(t),
\]

(3.13)

\( \forall i \in S, r = 0, 1, ..., T - 1, \) with \( V_{T,i}(t) = K_T(i), \forall i \in S, t \geq 0, \) as before. The ODE (3.13) is an asymptotically stable linear system with \( V_{r,i}(\bar{\pi}) \) as its unique asymptotically stable equilibrium point where \( V_r(\bar{\pi}) = (V_{r,i}(\bar{\pi}), i \in S)^T \) satisfies the Poisson equation

\[
V_{r,i}(\bar{\pi}) = F_r(i, \bar{\mu}_{r,i}, V_{r+1}(\bar{\pi})).
\]

(3.14)

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

\[
\hat{P}^j_i(v(y)) = \lim_{\eta \to 0} \left( \frac{P^j_i(y + \eta v(y)) - y}{\eta} \right)
\]

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

\[
\hat{P}_i(y) = (\hat{P}^1_i(y^1), ..., \hat{P}^N_i(y^N))^T.
\]
3.1 Simulation–Based Optimization Algorithms for Finite Horizon MDPs

The operators $\hat{P}_i(\cdot)$ are required to force the evolution of the corresponding ODE (below) within the feasible set of actions $U_i$ (cf. [Kushner and Clark, 1978, pp. 191] for a discussion on similar operators). Let $\nabla_r W(\pi)$ denote the $N$-vector gradient of any function $W(\pi)$ w.r.t. $\mu_{r,i}$. Consider the ODE

$$
\dot{\mu}_{r,i}(t) = \hat{P}_i(-\nabla_{r,i} V_{r,i}(\pi(t)))
$$

(3.15)

for $i \in S, r = 0, 1, ..., T - 1$. Let $K = \{\pi \mid \hat{P}_i(\nabla_{r,i} V_{r,i}(\pi)) = 0, \forall i \in S, r = 0, 1, ..., T - 1\}$ be the set of all fixed points of (3.15). Also, given $\epsilon > 0$, let $K^\epsilon = \{\pi \mid \exists \pi_0 \in K s.t. \|\pi - \pi_0\| < \epsilon\}$ be the set of all policies that are within a distance $\epsilon$ from $K$. Let $\{t(n)\}$ be defined as $t(0) = 0, t(n) = \sum_{j=0}^{n-1} b(j), n \geq 1$. 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 $\bar{w}_r(t) = (\bar{w}_{r,i}(t), i \in S)^T, r = 0, 1, ..., T - 1$, defined by $\bar{w}_{r,i}(t(n)) = V_{r,i}(n)$ with the maps $t \mapsto V_{r,i}(t)$ being continuous linear interpolations on $[t(n), t(n + 1)], n \geq 0$. Given a constant $\bar{T} > 0$, define $T_0 = 0$ and $T_n = min\{t(m)|t(m) \geq T_{n-1} + \bar{T}\}, n \geq 1$. 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 $\bar{w}_{n,r,i}(t), t \in I_n, n \geq 0$, defined as $\bar{w}_{n,r,i}(T_n) = \bar{w}_{n,r,i}(t(m_n))$ and

$$
\dot{\bar{w}}_{n,r,i}(t) = F_r(i, \hat{\mu}_{r,i}(t), \bar{w}_{n,r,i+1}(t)) - \bar{w}_{n,r,i}(t)
$$

(3.16)

with $\bar{w}_{n,r,i}(T_n) = \bar{w}_{n,r,i}(t(m_n)) = V_{r,i}(m_n)$. Also, $\bar{w}_{n,r}(t) \equiv (\bar{w}_{n,r,i}(t), i \in S)^T$. Now note that

$$
\bar{w}_{r,i}(t(n + 1)) = \bar{w}_{r,i}(t(n)) + \int_{t(n)}^{t(n+1)} F_r(i, \hat{\mu}_{r,i}(t), \bar{w}_{r}(t))dt
$$

$$
+ \int_{t(n)}^{t(n+1)} (F_r(i, \hat{\mu}_{r,i}(t(n)), \bar{w}_{r}(t(n))) - F_r(i, \hat{\mu}_{r,i}(t), \bar{w}_{r}(t))) dt + (M_{n+1,r}(i) - M_{n,r}(i)).
$$

(3.17)
3.1 Simulation–Based Optimization Algorithms for Finite Horizon MDPs

By Lemma 3, the third term on RHS of the above equation (within integral) can be seen to be of order $O(\tilde{b}(n)^2)$. Also, the last term on RHS above is $o(1)$ because of Lemma 4.

Further

$$\bar{w}_{n,r,i}(t) = \bar{w}_{n,r,i}(T_n) + \int_{T_n}^{t} F_r(i, \tilde{\mu}_{r,i}(s), \bar{w}_{n,r}(s)) ds.$$ (3.18)

From (3.17)-(3.18), an application of Gronwall’s inequality gives

$$\lim_{n \to \infty} \sup_{t \in I_n} |\bar{w}_{n,r,i}(t) - \bar{w}_{r,i}(t)| = 0$$ (3.19)

with probability one. Now note that the first iteration (3.7) of the algorithm can be rewritten as

$$\mu_{r,i}(n + 1) = P^i(\mu_{r,i}(n) - \tilde{b}(n)\xi(n))$$ (3.20)

where $\xi(n) = o(1)$ since $c(n) = o(\tilde{b}(n))$. From (3.19)-(3.20), the algorithm ACA can be seen to asymptotically track the trajectories of the ODE (3.11)-(3.12) along the faster timescale $\{t(n)\}$.

This we explain via a result of Hirsch (1989): consider an ODE in $\mathcal{R}^{[S] \times T}$ given by

$$\dot{z} = f(z),$$ (3.21)

for a Lipschitz continuous $f : \mathcal{R}^{[S] \times T} \to \mathcal{R}^{[S] \times T}$ s.t. (3.21) has a globally asymptotically stable attractor $\mathcal{Y}$. Given $\epsilon > 0$, let $\mathcal{Y}^\epsilon$ denote the $\epsilon$–neighborhood of $\mathcal{Y}$ i.e., $\mathcal{Y}^\epsilon = \{x \mid \|x - y\| < \epsilon, y \in \mathcal{Y}\}$. Given $T, \delta > 0$ (note that we use $\delta$ as distinct from the parameter of SPSA), we call a bounded, measurable $x(\cdot) : \mathcal{R}^+ \cup \{0\} \to \mathcal{R}^{[S] \times T}$, a $(T, \delta)$–perturbation of (3.21) if there exist $0 = T_0 < T_1 < T_2 < \cdots < T_n \uparrow \infty$ with $T_{n+1} - T_n \geq T \forall n$ and solutions $z^n(y), y \in [T_n, T_{n+1}]$ of (3.21) for $n \geq 0$, s.t. $\sup_{y \in [T_n, T_{n+1}]} \|z^n(y) - x(y)\| < \delta$. We now recall the result from Hirsch (1989):

**Lemma 5** Hirsch’s Lemma Given $\epsilon, T > 0$, $\exists \delta > 0$ s.t. for all $\delta \in (0, \delta)$, every $(T, \delta)$–perturbation of (3.21) converges to $\mathcal{Y}^\epsilon$. □
Thus, from Lemma 5, one obtains

**Lemma 6** For all \( i \in S, r = 0, 1, ..., T - 1 \), w.p. 1

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

We thus have,

**Theorem 2** Given \( \epsilon > 0, \exists \delta_0 > 0 \) such that \( \forall \delta \in (0, \delta_0] \), the algorithm ACA converges to \( K^\epsilon \) in the limit as \( M \to \infty \) with probability one.

Proof: Suppose \( \pi(n) \overset{\Delta}{=} (\mu_{r,i}(n), i \in S, r = 0, 1, ..., T - 1)^T \in \prod_{r=0}^{T-1} \prod_{i \in S} U_r^o(i) \), where \( U_r^o(i) \) denotes the interior of \( U_r(i) \). Choose \( \delta > 0 \) small enough so that \( \bar{\pi}(n) = (\mu_{r,i}(n) + \delta \Delta_{r,i}(n), i \in S, r = 0, 1, ..., T - 1)^T \), i.e., the perturbed policy lies within the feasible region.

From Lemma 6 and the Poisson equation for finite horizon MDPs (3.14), the recursion (3.7) of the algorithm can be seen to be asymptotically analogous to the recursion

\[
\mu_{r,i}(n + 1) = P_i \left( \mu_{r,i}(n) - c(n) V_{r,i}(\bar{\pi}(n)) \right). \tag{3.22}
\]

Now performing a Taylor series expansion of \( V_{r,i}(\bar{\pi}(n)) \) around \( \pi(n) \), one can show using Theorem 2.5-Corollary 2.6 of Bhatnagar et al. (2003) that (3.22) is asymptotically equivalent to

\[
\mu_{r,i}(n + 1) = P_i \left( \mu_{r,i}(n) - c(n) \nabla_{r,i} V_{r,i}(\pi(n)) \right). \tag{3.23}
\]

Constructing \( t(n) \) using \( c(n) \) (instead of \( \tilde{b}(n) \) as done to motivate Lemma 6), and letting \( \delta \to 0 \) and \( M \to \infty \), (3.23) can be seen to be a discretization of the ODE (3.15). For \( \pi(n) \) on the boundary of \( \prod_{r=0}^{T-1} \prod_{i \in S} U_{r,i} \) (for generic \( U_{r,i} \)), either \( \bar{\pi}(n) = (\mu_{r,i}(n) + \delta \Delta_{r,i}(n), i \in S, r = 0, 1, ..., T - 1)^T \) for sufficiently small \( \delta \) as before (in which case the above continues to hold) or else there exists an \( r_0 \in \{0, 1, ..., T - 1\} \), and a state \( i \in S \) and \( j \in \{1, 2, ..., N\} \) such that \( P_i^j (\mu_{r_0,i}(n) + \delta \Delta_{r_0,i}(n)) = \mu_{r,i}(n) \). 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 (cf. pp. 79 of Kushner and Yin (1997)). However, in our algorithm, since \( \Delta_{r,i}^j(n) \) 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
\[
\nabla_{r,i} V_{r,i}(\pi) \leq \hat{P}_i(-\nabla_{r,i} V_{r,i}(\pi)) < 0
\]
outside the set \( K \). It is easy to see that \( K \) serves as an asymptotically stable attractor set for (3.15) with \( \sum_{r=0}^{T-1} \sum_{i \in S} V_{r,i}(\pi) \) as the associated strict Liapunov function. The claim follows.

\[\square\]

**Convergence Analysis for Finite Action Setting**

First we analyze DPAFA and use Assumption 2 here. 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 \( K_r(i, u, j) \) and \( p_r(i, u, j) \) by \( \bar{K}_r(i, u, j) \) and \( \bar{p}_r(i, u, j) \), respectively, for \( i, j \in S, u \in U_i \), such that \( \bar{K}_r(i, u, j) = K_r(i, u, j) \) and \( \bar{p}_r(i, u, j) = p_r(i, u, j) \), \( u \in \bar{U}_i \) and such that \( \bar{K}_r(i, u, j) \) 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 \( \bar{K}_r(i, u, j) \) 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 by Gerencser et al. (1999) for measurement based optimization in a resource allocation problem except that in the latter, the algorithm is restarted after projecting the iterates to the discrete set.

The analysis in the RPAFA 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 (3.3), the faster timescale...
3.2 Parametrized Actor-Critic Algorithms for FH-MDPs

Policy evaluation using parameterized cost-to-go estimates in Markov Decision Processes (MDPs) are relevant due to their wide applications in communication (cf. Marbach et al. (2000)) and finance (cf. Tsitsiklis and Van Roy (1999b)) where state-spaces of the systems are typically large. In particular, policy evaluation algorithms that approximate the cost-to-go from each state using linear parameterization have been proposed for the infinite-horizon criteria of average cost by Tsitsiklis and Van Roy (1999a), for discounted cost by Tsitsiklis and Van Roy (1997) and further for stopping time problems by Tsitsiklis and Van Roy (1999b). A crucial enhancement to the basic method is pursued by Boyan (1999). However, with regard to finite-horizon total cost MDPs (FH-MDPs), there has not been any treatment of a similar nature. This may be due to the convenience that full $T$-length trajectories can be simulated in FH-MDPs, without having to resort to use of the key Poisson equation. Such an advantage is not available to infinite-horizon MDPs, however, where parametrized cost-to-go estimates are further employed to compute the optimal policy using a policy gradient method (cf. Konda and Tsitsiklis (2003)).

For FH-MDPs, a simulation-based solution that stored the cost-to-go estimates in a look-up table was proposed in §3.1. A key feature there was that online sampling of the states was not necessary, i.e. simulating an entire trajectory was not required, repeated simulation of a single transition from each state being sufficient. However, at issue here...
is the size of such a look-up table, since when compared to infinite-horizon MDPs, the memory requirements in FH-MDPs assume severity due to the absence of stationarity. A look-up table of size $|S| \times T$ is needed, where $T \in \mathbb{Z}^+$ is the length of the finite horizon and assuming that all states are feasible in all stages. It is this memory requirement that the proposed algorithms address. The lack of stationarity also manifests itself in the absence of interpretable bounds on the cost-to-go approximation - a distinguishing feature of the work of Tsitsiklis and Van Roy (1997) and Tsitsiklis and Van Roy (1999a). Among applications of FH-MDPs with an emphasis on cost-to-go approximation are problems in Online Mechanism Design (cf. Parkes et al. (2004)). In a spirit similar to Tsitsiklis and Van Roy (1997) and related work, the proposed algorithms use feature vectors $\phi_r(i), \forall i \in S$ and $0 \leq r \leq T - 1$ where $\phi_r(i) \in \mathbb{R}^K, K \ll |S|$. Accordingly, this algorithm searches in a subset of the space of deterministic policies viz. policies whose $r-$th stage actions lie in the span of the feature vector matrix $\Phi_r = (\phi_r(i), 1 \leq i \leq |S|)^T$. Many FH-MDPs require only a finite-action setting, but since the closed convex-hull of such an action set is compact, the algorithm DP_AFA of §3.1.3 above applies. This observation also holds true for the algorithms that are proposed herein.

We now outline the structure of the current section. The linear parameterization of policy iterate $\pi(n)$ is considered in §3.2.2 and §3.2.3 and certain key differences vis-a-vis relevant algorithms in the literature are also identified. Presented in §3.2.2 is a result that shows convergence w.p. 1 to a set satisfying necessary conditions of constrained optima. Further, §3.2.3 reports another algorithm for FH-MDPs with higher storage requirements, but lesser computation. While §3.2.4 considers extension to the control of stopping-time MDPs, simulation results are presented in §3.3.
3.2 Parametrized Actor-Critic Algorithms for FH-MDPs

3.2.1 Framework and Notation

For an FH-MDP operating under the \( n \)-th policy iterate \( \pi(n) \), the finite-horizon cost-to-go from the \( r \)-th stage onward, \( V_{r,i}(\pi(n)) \), is given by:

\[
E\{ \sum_{m=r}^{T-1} K_m(X_m, \pi_m(X_m(n)) + K_T(X_T)|X_i = i) \}.
\]

This we abbreviate as \( V_{r,i}(n) \). Here \( K_m(\cdot, \cdot) \) denotes the single stage cost at stage \( m \).

Without loss of generality we assume that the terminal cost \( K_T(X_T) = 0 \), \( \forall X_T \in S \). In the compact action setting, the action taken according to policy iterate \( \pi(n) \) when state \( i \) is observed in stage \( r \) is such that \( \pi_{r,i}(k) \in U_{r,i} = \Pi_{j=1}^{N_{r,i}}[a^j_{r,i}, b^j_{r,i}] \) where \( a^j_{r,i} < b^j_{r,i} \in \mathcal{R} \), \( \forall i \in S \). Thus the constraint sets \( U_{r,i} \) are chosen to be compact rectangles in \( \mathcal{R}^{N_{r,i}} \). We operate under a constraint on the action set:

**Assumption 3** The \( U_{r,i} \) are s.t. \( N_{r,i} = N_r \), \( a^j_{r,i} = a^j_r \), \( b^j_{r,i} = b^j_r \), \( \forall i \in S \) and \( 1 \leq j \leq N_r \).

While this does make the set \( U_{r,i} \) same over all \( i \in S \), we claim in \( \S 3.2.2 \) that an analogous restriction is implicit in other parametrization methods proposed in the literature.

Further, \( \pi_r(n) \in U_r = \Pi_{i=1}^{\mid S \mid} U_{r,i} \) and \( \pi(n) \in U^\Delta = \Pi_{r=0}^{T-1} U_r \) where \( \pi(n) \) is the vector \( (\pi_r(n))_{r=0}^{T-1} \) and each \( \pi_r(n) \) is the vector \( (\pi_{r,i}(n))_{i \in S} \), respectively. A projection operator \( P_{r,i}(\pi_{r,i}(n)) \) is used to truncate action \( \pi_{r,i}(n) \) into the above feasible set \( U_{r,i} \), with analogously defined operators \( P_r \) and \( P \) to project into \( U_r \) and \( U \), respectively. The proposed SPSA actor recursion is subject to:

**Assumption 4** For \( i \in S \), \( a \in U_{r,i} \), \( 0 \leq r \leq T - 1 \), and \( j \in S \), costs \( K_r(i,a) \) and transition probabilities \( p_r(i,a,j) \) are continuously differentiable w.r.t. \( a \).

The policy iterate \( \pi(n) \) is of size \( |S| \times T \) (taking \( N_{r,i} = 1 \), \( \forall r, i \)) which is also the size of the look-up table \( V_r(n) = (V_r(n), 1 \leq r \leq T - 1)^T \) where \( V_r(n) = (V_{r,i}(n), 1 \leq i \leq |S|)^T \). Since \( N_{r,i} \) indicates the dimension of the action at stage \( r \) in state \( i \), note that \( N_{r,i} > 1 \) would imply that size of policy \( \pi(n) \) is larger than that of the look-up table \( V(n) \).

In the spirit of [Tsitsiklis and Van Roy, 1997] and related work, the proposed algorithm uses feature vectors \( \phi_r(i), \forall i \in S \) where \( \phi_r(i) \in \mathcal{R}^K, K \ll |S| \). Define the \( |S| \times K \)}
3.2 Parametrized Actor-Critic Algorithms for FH-MDPs

matrix $\Phi_r$ as $(\phi_r(i), \forall i \in S)^T$ and the feature vector $\phi_r(i)$ for each stage-state pair $r, i$ as $(\phi_{r,i}(k), 1 \leq k \leq K)^T$. These features $\phi_r$ - which can be constructed based on metrics to grade the states - become available every time the state is encountered and thus do not need to be stored. We give an example of featurization of the state space $S$ in §3.3.

Feature vectors of states change with the stage $0 \leq r \leq T - 1$, demonstrating an added complication in FH-MDPs when compared to infinite-horizon MDPs. Also, we approximate the gradient-estimate of the cost-to-go $\nabla V_r(n) = (\nabla_{i} V_{r,i}(n), 1 \leq i \leq |S|)^T$ using a linear architecture, where $\nabla_{j} V_{r,i}(n)$ is an estimate of the partial-gradient of $V_{r,i}(n)$ w.r.t. $\pi_{r,j}(n)$. Since $V_{r,i}(n) = K_r(i, \pi_{r,i}(n)) + \sum_{j \in S_{r,i+1}} p_r(i, \pi_{r,i}(n), j)V_{r+1,j}(n)$, terms $\nabla_{j} V_{r,i}(n)$, for $j \neq i$, do not contribute to the update of $\pi_{r,i}(n)$. For a parsimonious representation, $\nabla V_r(n)$ is projected using the $L^2$ norm onto the space spanned by columns of $\Phi_r$. Similar to the treatment in §2.1.7 previously, the orthogonal projection of a vector $x \in \mathcal{R}^s$ is $\Pi_{\Phi_r} x$ where $\Pi_{\Phi_r} = \Phi_r (\Phi_r^T \Phi_r)^{-1} \Phi_r^T$. Thus $\nabla V_r(n)$ is approximated by $\Pi_{\Phi_r} \nabla V_r(n)$. All further treatment makes two simplifying assumptions, $N_{r,i} = 1, \forall r, i$ and that any state can be visited by the process in any stage.

3.2.2 Parametrized FH-MDP Algorithm 1

Certain key differences with the algorithms currently in literature are outlined before our algorithm is proposed, including disadvantages of the proposed scheme:

- The linear parametrization of the cost-to-go by Tsitsiklis and Van Roy (1997) and Konda and Tsitsiklis (2003) uses the fact that the vector $V_r(n) \in \mathcal{R}^{|S|}$ (in the setting of Tsitsiklis and Van Roy (1997), however, the subscript $r$ is irrelevant). Assumption 3 where $U_r \subset \mathcal{R}^{|S|}$, helps in similarly parametrizing the actor, i.e., policy iterate $\pi(n)$.

- In the proposed algorithm, the critic used is an estimate of the gradient of the cost-to-go at policy iterate $\pi(n)$. This gradient estimate in $\mathcal{R}^{|S|}$ is then projected using operator $\Pi_{\Phi_r}$. As seen in Theorem 3 this is central to showing convergence w.p. 1.

- The critic has the same linear parametrization as the actor, i.e. both belong to
the span of $\Phi_r$, although an estimate of the policy-gradient has no intuitive interpretation. We do not need to link the two parameterizations as in the scheme of Konda and Tsitsiklis (2003). The proposed Algorithm 1 is, in effect, an actor-critic implementation with a parametrized actor.

- Due to the compact action setting, two terms of Marbach and Tsitsiklis (2001), a) the likelihood ratio term $L_u(i, \theta)$, and b) gradient of reward term $\nabla K(i, \pi(n))$ are not required. Avoiding b) helps when one-step transition costs $K_r(X_r, \pi_{r,X_r}(n), X_{r+1})$ have the destination state $X_{r+1}$ as an argument. We optimize directly the expected total cost and not the individual cost samples using perturbation-analysis type schemes as Marbach and Tsitsiklis (2001) does. As a result, we do not require constraining regularity conditions. While Marbach and Tsitsiklis (2001) also estimate performance gradient w.r.t. parameter $\pi(n)$ (although for the infinite-horizon average-cost case), we use an explicit stochastic gradient formulation (viz. SPSA).

- No Dynamic Programming (DP) results are used in Algorithm 1. The Poisson equation: $V_{r,X_r}(n) = E\{K_r(X_r, \pi_{r,X_r}(n)) + V_{r+1,X_{r+1}}(n)\}$, is avoided. Use of $\Pi_{r+1} V_{r+1}(n)$ in place of $V_{r+1}(n)$ above renders the equation invalid. This handicap is mitigated, using a different method, in Algorithm 2 of §3.2.3 later.

- The ‘Monte Carlo’ nature of the critic in Algorithm 1 - requiring a $(T-l)$—length simulated trajectory until horizon $T$ is reached - is due to this inability to use the Poisson equation. Regeneration intervals, i.e. finite length simulated trajectories until a certain $i^*$ is hit, are also used in Marbach and Tsitsiklis (2001). However, unlike algorithms of §3.1 we require more than a single transition and this reflects in the computation time.

- In model-free algorithms, experimentation with a computer model of the system grants us liberty to simulate a $(T-l)$—length trajectory from any (stage, state) pair $(r, i)$ even though it is the total cost-to-go $V_{0,i}$ that is to be minimized. We use this to good effect in the proposed algorithm. Note that this assumption is implicit in algorithms of §3.1 and Bhatnagar and Abdulla (2006).
Proposed algorithms only handle $U_{r,i} \equiv U_r$, $\forall i \in S$. An analogous restriction holds in the work of, e.g., Konda and Tsitsiklis (2003) where $U_i$ are s.t. $|U_i| = u$, $\forall i \in S$.

For an $i \in S$ where $N_{r,i} < N_r$, the proposed algorithm can yet be modified by adding dummy intervals, and scaling and translating $U_{r,i}$ that violate $U_{r,i} \equiv U_r$.

We next motivate the proposed algorithm. Similar to the Dynamic Programming algorithm, we update the current policy by moving backwards in horizon with index $l$ from $T - 1$ to 0 (although, crucially, we do not use the dynamic programming principle). We use the compact action set assumption, i.e., $\pi_{r,i}(n) \in U_{r,i}$, and $\pi_{r,i}(n) = P_{r,i}((\phi_r(i), \bar{\pi}_r(n)))$ for the parametrized policy iterate $\bar{\pi}_r(n) \in R^K$. However, the action $\pi_{r,i}(n)$ is not used explicitly. Instead, the two-sided SPSA method perturbs policy iterate $\pi_r(n)$ with $\delta(n)\Delta_r(n)$ to produce policy $\pi_{r,i}^+(n) = P_{r,i}(\pi_{r,i}(n) + \delta(n)\Delta_{r,i}(n))$ and $\pi_{r,i}^-(n) = P_{r,i}(\pi_{r,i}(n) - \delta(n)\Delta_{r,i}(n)) \forall i \in S$ and measures system performance at these two policies. While $\delta(n)$ is a perturbation parameter that diminishes to zero in order that the bias w.r.t. the true policy gradient at $\pi_r(n)$ vanish as $n \to \infty$ (cf. Spall (1992)), $\Delta_r(n)$ is such that $\Delta_{r,i}(n) = \pm 1$ with probability 0.5 each. Policy gradient methods using SPSA estimates applied to solutions of MDPs are proposed by Bhatnagar and Abdulla (2006) and Abdulla and Bhatnagar (2007a).

The evaluation of $\Pi_{\Phi_r} \triangledown_t V_r(n)$ is undertaken in a manner reminiscent of the LSTD(0) algorithm of Boyan (1999). Projecting $V_r(n)$ to the span of $\Phi_r$ is an optimization problem that LSTD(0) does not solve incrementally, unlike TD($\lambda$) of Tsitsiklis and Van Roy (1997). Note that $\Pi_{\Phi_r} = \Phi_r(\Phi^T_r\Phi_r)^{-1}\Phi^T_r$ and hence we estimate $\Phi^T_r \triangledown_t V_r(n)$ first by making $L \gg 0$ starts from states $i$ chosen uniformly out of $S$ and simulating trajectory $\{X_{r+1}^+, X_{r+2}^+, \ldots, X_T^+ | X_r^+ = i\}$ using action $\pi_{r,i}^+(n)$ in stage $r$ and state $i$ and policies $\pi_{r+m}(n + 1), m \geq 1$ for stages $r + m$ upto $T - 1$. We call the accumulated cost in such a trajectory as $K^+$ and observe that since the starting state $i$ is chosen uniformly from $S$, $|S|\phi_r(i) \triangledown_t V_r(n)$ is a vector whose mean is $\Phi^T_r \triangledown_t V_r(n)$. The $K \times K$ matrix $(\Phi^T_r\Phi_r)^{-1}$ is assumed to be pre-stored but can also be estimated using an averaging of the $K \times K$ matrix iterates $\phi_r(i)\phi^T_r(i)$. Analogously, the simulation to compute $K^-$ is performed. The averaging of the ‘critic’ estimates $|S|\phi_r(i) \triangledown_t V_r(n)$ is on a faster timescale $b(n)$ than the
3.2 Parametrized Actor-Critic Algorithms for FH-MDPs

scale $a(n)$ used to update the parametrized ‘actor’ $\bar{\pi}_r(n)$. This relationship results in the actor recursion viewing the critic as having equilibrated to $\Phi^T_r \hat{\nabla} V_r(n)$ at each update $n$. In the trajectory, the next state from $X^{+}_t$ is indicated by the simulation random variable $\eta(X^{+}_t, \pi^{+}_t, X^{+}_{t+1}(n))$. While neither $\pi_r(n)$, $\pi^{+}_r(n)$ nor $\pi_r(n+1)$ may belong to the span of $\Phi_r$, the availability of the projection operator $P_r$ makes it possible to implement such policies during simulation. The number of policy updates $M$ for which the algorithm below is run is typically decided using some convergence criteria.

The requirements on step-sizes $a(n)$ and $b(n)$ are as follows:

$$b(n), a(n) > 0, \forall n \geq 0, \quad \sum_n b(n) = \sum_n a(n) = \infty,$$

$$\sum_n b(n)^2, \sum_n a(n)^2 < \infty, \quad \text{and} \quad a(n) = o(b(n)).$$

Similarly, the perturbation parameter $\delta(n)$ is such that $\sum_n \frac{a(n)}{\delta(n)} = \infty$, $\sum_n \left( \frac{a(n)}{\delta(n)} \right)^2 < \infty$, and $\frac{a(n)}{\delta(n)} = o(b(n))$. The proposed algorithm is described next:

**Algorithm 1**

- for $n = 0, 1, \ldots, M$ do: {
  - for $r = T - 1, \ldots, 0$ do: {
    - Critic: for $m = 0, 1, \ldots, L - 1$ do: {
      - choose $X^{+}_r = X^{-}_r = i$ uniformly from $S$, generate $\Delta_{r,i}(n)$.
      - $K^+ := 0, t := r$.
      - while $t < T$ do: {
        if ($t=r$)
        $$X^{+}_{r+1} := \eta(i, P_{r,i}(\pi_{r,i}(n) + \delta(n)\Delta_{r,i}(n)));$$
        $$K^+ := K^+ + K_r(i, P_{r,i}(\pi_{r,i}(n) + \delta(n)\Delta_{r,i}(n)));$$
        else
        $$X^{+}_{t+1} := \eta(X^{+}_t, P_{t,i}(\pi_{t,X^{+}_t}(n)));$$
      }
    }
  }
}

3.2 Parametrized Actor-Critic Algorithms for FH-MDPs

\[ K^+ := K^+ + K_t(X_t^+, P_{t,i}(\pi_{t,X_t^+}(n))) \]

\[ \text{endif} \]

\[ t := t + 1; \quad \} \]

\[ K^- := 0, \ t := r. \]

\[ \text{while } t < T \text{ do: } \{ \]

\[ \text{if } (t=r) \]

\[ X_{i+1}^- := \eta(i, P_{r,i}(\pi_{r,i}(n) - \delta(n)\Delta_{r,i}(n))); \]

\[ K^- := K^- + K_t(i, P_{r,i}(\pi_{r,i}(n) - \delta(n)\Delta_{r,i}(n))); \]

\[ \text{else} \]

\[ X_{i+1}^- := \eta(X_r^-, P_{t,i}(\pi_{t,X_r^-}(n))); \]

\[ K^- := K^- + K_t(X_r^-, P_{t,i}(\pi_{t,X_r^-}(n))); \]

\[ \text{endif} \]

\[ t := t + 1; \quad \} \]

\[ \nu_{nL+m+1,r} := \nu_{nL+m,r} + b(n) \left( |S| \left( \frac{K^+ - K^-}{2\delta(n)\Delta_{r,i}(n)} \right) \phi_r(i) - \nu_{nL+m,r} \right) \]

\[ \} \]

\[ \cdot \text{Actor:} \]

\[ \bar{\pi}_r(n+1) := (\Phi_r^T \Phi_r)^{-1} \Phi_r^T \cdot P_r \left( \Phi_r \bar{\pi}_r(n) - a(n)\Phi_r(\Phi_r^T \Phi_r)^{-1}\nu_{(n+1)L,r} \right) \quad (3.24) \]

Observe that unlike ACA, the policy \( \pi(n) = (\mu_r(n), 0 \leq r \leq T - 1) \) is not indexed with the state \( i \) chosen as the start-state for the length \( T - r \) trajectory. Similarly, the perturbation \( \Delta_r(nL + m) \) 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 \( \Delta_r(nL + m_1) \) and \( \Delta_r(nL + m_2) \). This phenomenon does not affect the convergence of the
algorithm. Also unlike ACA, we need here 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 above as against one-simulation SPSA in ACA has to do with the size of the look-up table $V_r(nL + m)$ in Step 1 of ACA. This table would be double the size were the two-simulation SPSA to be used. In contrast, such a problem does not arise here and we can afford the smaller variance two-simulation SPSA method.

We write (3.24) as:

$$\bar{\pi}_r(n + 1) := (\Phi^T_r \Phi_r)^{-1} \Phi^T_r P_r(\Phi_r \bar{\pi}_r(n) - \frac{a(n)}{2\delta(n)}(V^+(n) - V^-(n)) \circ \Delta_r^{-1}(n))$$

(3.25)

where $\circ$ stands for component-wise multiplication, in this case with the vector $\Delta_r^{-1}(n)$ (this vector, incidentally, equals $\Delta_r(n)$ due to our choice). The result below shows that, for a given stage $r$, the iterates in (3.24) converge to a candidate constrained optimum $\pi^*_r \in \text{span}(\Phi_r)$, discounting the possibility of spurious minima on the boundary of the feasible action set $U_r$. We adapt the algorithm in §5.1.5 of Kushner and Clark (1978) which treats constrained function minimization using the well known Kiefer-Wolfowitz algorithm.

**Theorem 3** Iterates $\pi_r(n)$ converge w.p. 1 to a $\pi^*_r$ to a set with necessary condition for constrained optima: $\{\pi^*_r : \Pi_r \nabla_r V_r(\pi^*) = 0, \pi^*_r \in U_r; \forall r\}$ where $\pi^* = (\pi^*_0, \pi^*_1, ..., \pi^*_T)^T$.

*Proof:* We reproduce the algorithm in §5.1.5 of Kushner and Clark (1978) for clarity and explain the terms:

$$X_{n+1} := X_n - a_n \pi(X_n)[Df(X_n, c_n) - \beta_n - \xi_n] - ka_n \Phi^T(X_n) \phi(X_n), \quad (3.26)$$

where $X_n \in \mathcal{R}^r$ is the iterate and $Df(X_n, c_n)$ is a finite difference approximation to $\nabla_x f(X_n)$, the gradient at $X_n$ of objective function $f : \mathcal{R}^r \mapsto \mathcal{R}$. The bias in such an estimate is represented by $\beta_n$ whilst $\xi_n$ is noise with certain conditions, Assumptions A5.1.1-A5.1.6 of Kushner and Clark (1978) covering these. The constraint set is denoted
as \( \{ x : \phi(x) = 0 \in \mathcal{R}^s, x \in \mathcal{R}^r \} \) where \( \phi(x) \equiv (\phi_1(x), \phi_2(x), \ldots, \phi_s(x))^T \). Note that the \( \phi \) above are not to be confused with feature vectors \( \phi_r \) that we use. Further, \( \Phi(X_n) \) is a matrix s.t. \( \Phi^T(X_n) = (\nabla_x\phi_1(X_n), \nabla_x\phi_2(X_n), \ldots, \nabla_x\phi_s(X_n)) \) (making \( \Phi^T(X_n) \) an \( r \times s \) matrix). The matrix \( \pi(X_n) \) in (3.26) is obtained from projection \( (I - \pi(X_n)) \) to the span of the columns of \( \Phi^T(X_n) \), whilst \( k \) is an arbitrary but fixed positive number. Here too, matrices \( \pi(X_n) \) are not be confused with policy iterates \( \pi(n) \).

We use the SPSA gradient estimate \( \hat{\nabla}V_r(n) \triangleq E \left( \frac{1}{2\delta(n)}(V_r^+(n) - V_r^-(n)) \circ \Delta_r^{-1}(n)|\pi(n) \right) \) in place of the \([\ldots]\) in (3.26). This is because replacing \( c_n \) with \( \delta(n) \) and using Lemma 1 of Spall (1992), the requirements in A5.1.11 of Kushner and Clark (1978) are satisfied. We now rewrite (3.26) replacing iterates \( X_n \) with \( \pi_r(n) \) as follows:

\[
\pi_r(n + 1) := \pi_r(n) - a_n \left[ \pi_r(n) \hat{\nabla}V_r(n) \right] - ka(n)\Phi^T(\pi_r(n))\phi(\pi_r(n)).
\]

The subscript \( r \) will distinguish our notation from the \( \Phi \) and \( \phi \) of (3.26) as we perform the following substitutions. The constraint set here is \( \{ \pi_r : (I - \Pi_{\Phi_r})\pi_r = 0 \} \), indicating that \( \pi_r \in \text{span}(\Phi_r) \). Similarly \( \Phi^T(\pi_r) \) can be seen to be \( (I - \Pi_{\Phi_r})^T \), \( \forall \pi_r \). Observe that if \( \pi_r(n) \in \text{span}(\Phi_r) \) then \( \phi(\pi_r(n)) = 0 \) canceling the last term in the RHS of (3.27). Further, \( \pi(\pi_r(n)) \) is s.t. \((I - \pi(\pi_r(n))) : \mathcal{R}^{|S|} \rightarrow \mathcal{R}^{|S|} \) is a projection to the span of the columns of \( \Phi^T \), the symmetry of \( I - \Pi_{\Phi_r} \) meaning that \( \pi(\pi_r(n)) = \Pi_{\Phi_r}, \forall \pi_r(n) \). Thus the recursion is now simplified to:

\[
\pi_r(n + 1) = \pi_r(n) - a(n) \left[ \Pi_{\Phi_r} \hat{\nabla}V_r(n) \right].
\]

Neglect operator \( P_r \) in (3.24) and pre-multiply with \( \Phi_r \) to yield the above equation. Now note that ODE evolution is in fact constrained to within \( U_r \) by \( P_r \), via a transformed projection operator handled in, e.g., Theorem 2 earlier.
3.2.3 Parametrized FH-MDP Algorithm 2

As before, the policy element $\pi_r(n)$ is parametrized, i.e., it belongs to the span of $\Phi_r$, $0 \leq r \leq T - 1$ requiring $K \times T$ memory. We will require two $|S|$-sized look-up vectors, $V^1$ and $V^2$. For each index $k$ of the algorithm, stages $r = T - 1, T - 2, \ldots, 0$, and all states $i \in S$ we generate the random variable $\Delta_{r,i}(n)$ from the set $\{+1, -1\}$ w.p. 0.5 each. Compute actions $P_r(\langle \phi_r(i), \pi_r(n) \rangle \pm \delta(n)\Delta_{r,i}(n))$ (call these $\pi^\pm_{r,i}(n)$), and simulate a transition using these actions to states $\eta_r^+$ and $\eta_r^-$, respectively, of the $(r + 1)$-th stage. The array $V^1$ is interpreted as the vector of $(r + 1)$-th stage costs-to-go, therefore

$$\tilde{\nabla}_i V_{r,i}(n) = \frac{1}{2\delta(n)}\Delta_{r,i}^{-1}(n) \left( K_r(i, \pi^+_{r,i}(n), \eta_r^+) + V^1(\eta_r^+) - K_r(i, \pi^-_{r,i}(n), \eta_r^-) - V^1(\eta_r^-) \right),$$

and we proceed further as in the critic recursion of the Algorithm 1, i.e., by scaling the vector $\phi_r(i)$ with the term $|S|\tilde{\nabla}_i V_{r,i}(n)$. We also transit to $\eta_r$ using $\pi_{r,i}(n) = P_r(\langle \phi_r(i), \pi_r(n) \rangle)$ and update $V^2(i)$ using cost-to-go estimate $K_r(i, \pi_{r,i}(n), \eta_r) + V^1(\eta_r)$ in a recursion with stepsize $b(n)$. After the actor update, we assign $V^2$ to $V^1$ and set $V^2$ to the zero vector $0$. This algorithm is described next:

Algorithm 2

- for $n = 0, 1, \ldots, M$ do: 
  - Initialize $V^1 := 0$

- for $r = T - 1, \ldots, 0$ do: 
  - Critic: for $m = 0, 1, \ldots, L - 1$ do: 
    - choose $i \in S$ uniformly, do: 
      - generate $\Delta_{r,i}(n)$
      - $X_{r+1}^+ := \eta(i, P_{r,i}(\pi_{r,i}(n) + \delta(n)\Delta_{r,i}(n))$;
      - $K^+ := K_r(i, P_{r,i}(\pi_{r,i}^+(n))) + V^1(X_{r+1}^+)$;
- $X^r_{t+1} := \eta(i, P_{r,i}(\pi_{r,i}(n)) - \delta(n)\Delta_{r,i}(n));$
  $K^- := K_r(i, P_{r,i}(\pi_{r,i}(n))) + V^1(X^r_{t+1});$

- $X^r_{t+1} := \eta(i, P_{r,i}(\pi_{r,i}(n)));$
  $K := K_r(i, P_{r,i}(\pi_{r,i}(n))) + V^1(X^r_{t+1});$

- $V^2(i) := V^2(i) + b(n)(K - V^2(i));$

- $\nu_{kL+m+1,r} := \nu_{kL+m,r} + b(n)\left(|S|\left(\frac{K^- - K^+}{2\delta(n)\Delta_{r,i}(n)}\right)\phi_r(i) - \nu_{kL+m,r}\right)$

- $\overline{\pi}_{k+1,r} := (\Phi^T_l\Phi_r)^{-1}\Phi^T_rP_r(\Phi_r\overline{\pi}_r(n) - a(n)\Phi_r(\Phi^T_r\Phi_r)^{-1}\nu_{(k+1)L,r})$

- Flip: $V^1 := V^2$ and $V^2 := 0.$

The advantage here is in the reduced simulation load: $O(T)$ instead of $O(T^2)$ in the previous algorithm. There is an increased memory requirement vis-a-vis Algorithm 1, since we need additional $2|S|$ memory. Thus, this method is suitable for FH-MDPs where $T \geq 2$. Also note that the iterated backward flow $r = T - 1, T - 2, \ldots, 0$ of the proposed algorithm prevents parallelization of the algorithm, unlike analogous methods

We describe briefly another algorithm with reduced simulation load conceived of on the same lines. For each stage $r$ moving backwards from $T - 1$ to 0, $|S|$ times we pick a state $i \in S$ uniformly, simulating $L$ transitions each using perturbed actions $\pi_{r,i}(n)$ and
3.2 Parametrized Actor-Critic Algorithms for FH-MDPs

3.2.4 Algorithm 3: Stopping Time Problem

To motivate the algorithm, we introduce the stopping time problem and discuss how current policy evaluation algorithms are unsuitable as critic for the SPSA-based policy-gradient methods of the three algorithms proposed thus far in this section. Consider a finite-state stopping time MDP where the actions applied in each state of the trajectory \( \{X_0, X_1, ..., X_{\tau(n)} | X_0 = i \} \in S \) use policy \( \pi(n) = (\pi_j(n), j \in S)^T \), \( k \geq 0 \), stopping time \( \tau(\pi(n), i) \) being a random variable. The cost-to-go is given by \( V_i(n) = E \left( \sum_{n=0}^{\tau(\pi(n),i)-1} K(X_n, \pi_{X,n}(n)) | X_0 = i \right) \) for all \( i \in S \). Note that \( \pi_i(n) \in U_i \equiv \Pi_{m=1}^{N_i} [a_{i}^{m}, b_{i}^{m}] \) for \( a_{i}^{m} < b_{i}^{m} \) and, similar to the operators \( P_{r,i} \) and \( P_r \) earlier (but without a subscript indicating stage), there exist projection operators \( P_i \) and \( P \) into corresponding \( U_i \) and \( U \). For ease of exposition, assume that the policy iterate \( \pi(n) \) is not parametrized. Policy gradient methods of Bhatnagar and Abdulla (2007), Bhatnagar and Abdulla (2006) and Abdulla and Bhatnagar (2007a) would suggest a policy update iteration:

\[
\pi(n + 1) := P \left( \pi(n) - \frac{a(n)}{2\delta(n)} (V^+(n) - V^-(n)) \circ \Delta^{-1}(n) \right) \quad (3.28)
\]

where \( V^\pm(n) = (V_j(\pi^\pm(n)), \forall j \in S)^T \). Note that, in practice, noisy estimates of \( V^\pm(n) \) will be used.

Now assume that we parameterize the \( V^\pm(n) \) in (3.28): \( V_i(\pi^\pm(n)) = \langle \phi(i), \nu^\pm(n) \rangle \),

where \( \nu^\pm(n) \in R^K \) for \( K \ll |S| \), and \( \phi(i) \) is the feature vector of state \( i \in S \). Algorithms
that apply the methods of TD-(λ) and its variant (cf. Tsitsiklis and Van Roy (1997), and Tsitsiklis and Van Roy (1999a)) or LSTD-(λ) (cf. Bovar (1999)) for policy evaluation are candidates to estimate the critic terms $V^\pm(n)$. Such a policy evaluation has been applied to an autonomous stopping time problem (viz. a Markov Reward Process where only the stopping decision has to be taken in each state) by Tsitsiklis and Van Roy (1999b). In contrast, here we have controls $\pi_i(n)$ in each state $i \in S$. When combined with the SPSA-type policy update (3.28), the above methods of policy evaluation have a potential flaw. Consider the $|S| \times K$ matrix $\Phi = (\phi(i), 1 \leq i \leq |S|)^T$ and $V^+(n) = V(\pi^+(n))$ indicating cost-to-go for perturbed policy iterate $\pi^+(n)$. Policy evaluation methods perform a projection of $V^+(n)$ using a linear operator $\Pi_{\Phi} \Phi^T$ and $V^+(n) = V^+(\pi^+(n))$ indicating cost-to-go for perturbed policy iterate $\pi^+(n)$. Policy evaluation methods perform a projection of $V^+(n)$ using a linear operator $\Pi_{\Phi} \Phi^T$ to a point $W$ in the span of $\Phi$, minimizing a weighted Euclidean norm $\sum_{i=1}^{|S|} p_{\pi^+(n)}(i)(V^+(n) - W(i))^2$, the weights $p_{\pi^+(n)}$ being stationary probabilities of visiting state $i$ under policy $\pi^+(n)$. Tsitsiklis and Van Roy (1997) further have a bounded error w.r.t. $W$ which is presently ignored. Thus, the projection operator depends on the policy $\pi^+(n)$ through the weights $p_{\pi^+(n)}(i)$.

These concerns result in (3.28) being

$$\pi(n + 1) := P(\pi(n) - \frac{a(n)}{2\delta(n)}(\Pi_{\Phi, \pi^+(n)} V^+(n) - \Pi_{\Phi, \pi^-(n)} V^-(n)) \circ \Delta^{-1}(n)),$$

which performs gradient descent to a policy $\pi^*$ that minimizes $\Pi_{\Phi} V(\pi)$, not $V(\pi)$ component-wise. In contrast, the ideal analog of (3.28) would be:

$$\pi(n + 1) := P(\pi(n) - \frac{a(n)}{2\delta(n)}(\Pi_{\Phi} V^+(n) - \Pi_{\Phi} V^-(n)) \circ \Delta^{-1}(n)),$$

where the projection operator $\Pi_{\Phi}$ is independent of the policy iterates $\pi(n)$.

The algorithm of §3.2.2 extends, and we do not provide a detailed description. The key difference is that, per iteration $n$, $L$ trajectories from each $i \in S$ are simulated. Such trajectories - simulated until stopping time is hit - could be long, with differing stop times $\tau(\pi^+(n), i)$ and $\tau(\pi^-(n), i)$. Compare this with the algorithm of §3.2.2 that requires precisely $T - l$ length trajectories for the $l$-th stage, or of §3.2.3 that uses transitions. Also, an analog of Assumption 3 is required: $N_i = N$, $a_i^m = a_i^m$, and $b_i^m = b_i^m$, $\forall i \in S$. 

3.3 Simulation Results

3.3.1 Look-up Table FH-MDP Algorithms

We consider a continuous time queuing model of flow control. The numerical setting here is somewhat similar to the ones of Bhatnagar and Kumar (2004) and Bhatnagar et al. (2001b), respectively. Bhatnagar and Kumar (2004), model this problem in the infinite horizon discounted cost MDP framework, while Bhatnagar et al. (2001b) consider it in the simulation optimization setting. We study this problem in the finite horizon MDP framework here. It is realistic to study flow and congestion control problems in a finite horizon framework, as any user typically holds the network for only a finite time duration. From the user’s perspective, it makes sense to develop control strategies for the duration of time that the user is active in the network. Also, while running (say) high priority applications in communication networks such as the Internet, it is also the time needed to control congestion, while at the same time supporting the applications with enough bandwidth, that is of significance. Such issues cannot be realistically captured while modelling using an infinite horizon, time-stationary framework. Finite-horizon dynamic programming tasks also form natural subproblems in certain kinds of MDPs, e.g. §2 of Chang et al. (2003) 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 problem is also related to Available Bit Rate (ABR) or Unspecified Bit Rate (UBR) explicit-rate flow control in ATM networks. An ATM source permits three broad categories of connections: ABR, UBR and Constant Bit Rate (CBR). ATM sources can infer that certain CBR connections will be established and will commence sending at a specific time-of-day, e.g. business connections begin at 9 AM and close by 6 PM. When a ABR or UBR connection establishment occurs outside these hours, the free bandwidth is available only for a finite horizon of time, i.e. until 9 AM the following day. The ATM source must clear the network by the time the CBR connections are established again. ATM Forum specifications do not require such flow-control, however tuning ABR rates
3.3 Simulation Results

with closed-loop feedback (e.g. using the proprietary Cisco ForeSight algorithm) help in better throughput at the ATM sources.

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 $\tilde{TT}$. 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 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 $\pi = \{\mu_0, \mu_1, ..., \mu_{T-1}\}$ is computed as follows: For $r = 0, 1, ..., T - 2$,

$$K_r(i_r, \mu_{r,i_r}, i_{r+1}) = \left|i_{r+1} - \left(0.8B - \frac{r}{\tilde{T}} \times 0.6B\right)\right|,$$  

while $g_{T-1}(i_{T-1}, \mu_{T-1,i_{T-1}}, i_T) = i_T$. Also, $K_T(i) = 0, \forall i \in S$. One-stage cost functions as above penalize states away from the target states $\hat{T}_{r+1} \triangleq (0.8B - \frac{r}{\tilde{T}} \times 0.6B), r = 0, 1, ..., T - 2$ and $\hat{T}_T = 0$. Since we used $B = 50$ and $T = 10$, the target states (in the above single stage costs) chosen are 40, 37, 34, ..., 16, 0 for stages 1, 2, ..., 10, respectively. The goal thus is 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.

For the case of compact action sets, we select the action set in each state to 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. Note that the single stage cost is bounded and does not depend explicitly on actions. Further, Bhatnagar et al. (2001b) show the (time homogeneous) transition probabilities for this problem to be continuously differentiable functions of the source rate. 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 (3.1)-(3.2) 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. In order to compute the transition probability matrix $P_\tilde{T}$ for the embedded Markov chain under policy $\pi$, we use the approximation method suggested by (Ross, 2000, §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 criteria specified for the method of (Ross, 2000, §6.8). Moreover, such probabilities can only be computed for systems whose dynamics are known, our setting being that of a well-studied $M/M/1/B$ queue.

![Figure 3.1: Using Approximate DP: (a) Optimal Policy and (b) Finite-Horizon Cost](image)

The policy obtained using finite-horizon DP with $P_\tilde{T}$ computed as above is shown in Figure 3.1a. 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 3.1b.
3.3 Simulation Results

![Figure 3.2: Converged Policy using Algorithm ACA for Compact Action setting](image)

Figure 3.2: Converged Policy using Algorithm ACA for Compact Action setting

![Figure 3.3: Converged Policy in Discrete Action setting (a) RPAFA and (b) DPAFA](image)

(a) (b)

Figure 3.3: Converged Policy in Discrete Action setting (a) RPAFA and (b) DPAFA

is the finite-horizon cost for each state in a system operating under the (finite-horizon) policy of Figure 3.1a.

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

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

Here \( \mu_{r,i}(n) \), specifies the feedback source rate when \( i \) is the \( r \)th state to be observed, \( 0 \leq r \leq T - 1 \), at policy update \( n \). The convergence criterion chosen for the algorithm
3.3 Simulation Results

Figure 3.4: Finite-Horizon Cost using Algorithm ACA for Compact Action setting

Figure 3.5: Finite-Horizon Cost in Discrete Action setting (a) RPAFA and (b) DPAFA

RPAFA is similar with \( \text{err}_n \leq 0.01 \), where \( \text{err}_n \) is now defined as

\[
\text{err}_n = \max_{i \in S, k \in \{1,2,...,50\}} \left\{ \sqrt{\sum_{r=0}^{T-1} \sum_{u^j \in U_{r,i}} |\pi^j_{r,i}(n) - \pi^j_{r,i}(n-k)|} \right\}.
\]

Here, \( \pi^j_{r,i}(n) \) denotes the probability of choosing rate \( u^j \) in the \( r \)-th observed state \( i \) for the policy obtained at the \( n \)th update. We choose \( \text{err}_n \) to be much lower here because RPAFA performs a search in the space of probabilities, components of which are all \( \leq 1 \). The policies obtained in all three cases are shown in Figures 3.2, 3.3a and 3.3b while the corresponding finite-horizon costs are plotted in Figures 3.4, 3.5a, and 3.5b, respectively.
The plots shown in Figures 3.1b, 3.4, 3.5a and 3.5b, respectively are obtained from $2 \times 10^5$ independent sample trajectories $\{i_0, i_1, ..., i_{10}\}$, 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 3.3a, for each state the source rate indicated is the rate that has the maximum probability of selection. The costs in Figures 3.1b and 3.5a (also many states in Figure 3.5b) are higher than the compact action case (cf. Figure 3.4) due to the significantly broader range of actions available in the latter setting.

For the algorithms compared over the $B = 50$ setting, tables 3.1, 3.2 and 3.3 show performance comparisons of the proposed algorithms for various metrics with mean and standard deviation taken over the (above mentioned) $2 \times 10^5$ independent sample trajectories. Table 3.1 shows the mean queue length $E(i_r)$ at instants $r = 2, 4, 6, 8,$ and 10, respectively, with the corresponding standard deviation. With $\hat{T}_r$ defined as the ‘target’ state at instant $r$, Table 3.2 shows the probability of the system being in states $\hat{T}_r \pm 1$ at the above values of $r$. Table 3.3 shows the mean one-stage cost $E(K_{r-1}(i_{r-1}, \mu_{r-1,i_{r-1}}, i_r)) = E(|i_r - \hat{T}_r|)$ incurred by the system during transition from $i_{r-1}$ to $i_r$ under action $\mu_{r-1,i_{r-1}}$. From the tables it can be seen that RPFA does not show good performance as it selects an action at each instant based on a randomized policy wherein the ‘optimal’ action, though having the highest probability of selection, may not be selected each time. We observe that DPFA shows better performance than RPFA since it updates deterministic policies in the convex hull of feasible actions and does not require projection into the probability simplex $\bar{S}$. Further, one does not require in the case of DPFA a 4-dimensional Hadamard perturbation as with the latter. One can also see from Table 3.2 that the measured standard deviations match closely the theoretical values obtained in the following manner: Suppose that, for a given $r$ with $1 \leq r \leq T$, $I_{\{i_r = \hat{T}_r \pm 1\}}$ denotes the indicator random variable taking 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 $\sigma_r$ is given by

$$\sigma_r = \sqrt{E(I_{\{v_r = T_r \pm 1\}}^2) - E^2(I_{\{v_r = T_r \pm 1\}})} = \sqrt{p_r(1 - p_r)}.$$ 

We also compare against the simulation-based algorithm termed SAMW (Simulated Annealing with Multiplicative Weights) proposed in Chang et al. (2007b), which is not an actor-critic algorithm. We reproduce the algorithm here:

$$\phi^{i+1}(\pi) := \phi^i(\pi) \frac{V_\pi^i}{Z_i}, \ \forall \pi \in \Pi,$$

where $\phi^i(\pi)$ is the probability of choosing a policy $\pi \in \Pi$ to control the system with, in the $i$-th iteration. By simulating a trajectory using policy $\pi$, we obtain $V_\pi^i$ 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 terminating iteration $T$ is, in general, unknown. Further, $Z^i$ is the zustadsumme, the normalising term $\sum_{\forall \pi \in \Pi} V_\pi^i$.

Though 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 $\phi$ has one entry per feasible policy, making it exponential in the number of stages $r$ and states $x$. The methods DP, A 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 like the err described below. To produce the better convergence properties, we chose $\beta_i = 1.025$ in the SAMW experiments shown here. We infer from Table 3.3 that SAMW does outperform RPAFA, but is bettered by DPAFA.

For the approximate DP algorithm, the transition probabilities using the method of Ross (2000) are easily computed in our setting. However, in many real life scenarios, computing these probabilities may not be as simple, and one may need to rely exclusively on simulation based methods.
### 3.3 Simulation Results

<table>
<thead>
<tr>
<th>Target $T_r$</th>
<th>$r=2$</th>
<th>$r=4$</th>
<th>$r=6$</th>
<th>$r=8$</th>
<th>$r=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP</td>
<td>27.14±6.28</td>
<td>29.36±4.85</td>
<td>22.43±5.10</td>
<td>16.51±4.71</td>
<td>5.73±3.56</td>
</tr>
<tr>
<td>SAMW</td>
<td>27.21±6.29</td>
<td>29.19±4.94</td>
<td>22.42±5.13</td>
<td>16.35±4.69</td>
<td>5.60±4.70</td>
</tr>
<tr>
<td>ACA</td>
<td>26.87±6.22</td>
<td>29.73±4.13</td>
<td>23.32±4.18</td>
<td>17.29±4.05</td>
<td>6.19±3.34</td>
</tr>
<tr>
<td>RPAFA</td>
<td>19.22±6.51</td>
<td>24.26±5.16</td>
<td>19.37±4.79</td>
<td>15.11±4.89</td>
<td>8.86±5.28</td>
</tr>
<tr>
<td>DP AF A</td>
<td>27.09±6.16</td>
<td>28.36±4.52</td>
<td>22.03±4.59</td>
<td>15.93±4.34</td>
<td>5.52±3.34</td>
</tr>
</tbody>
</table>

Table 3.1: observed $E(i_r)$ for the proposed algorithms

<table>
<thead>
<tr>
<th>Target $T_r$</th>
<th>$r=2$</th>
<th>$r=4$</th>
<th>$r=6$</th>
<th>$r=8$</th>
<th>$r=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP</td>
<td>0.07±0.13</td>
<td>0.22±0.34</td>
<td>0.21±0.34</td>
<td>0.22±0.34</td>
<td>0.19±0.31</td>
</tr>
<tr>
<td>SAMW</td>
<td>0.03±0.05</td>
<td>0.14±0.25</td>
<td>0.13±0.23</td>
<td>0.10±0.19</td>
<td>0.10±0.18</td>
</tr>
<tr>
<td>ACA</td>
<td>0.07±0.13</td>
<td>0.25±0.37</td>
<td>0.25±0.38</td>
<td>0.25±0.37</td>
<td>0.13±0.22</td>
</tr>
<tr>
<td>RPAFA</td>
<td>0.01±0.03</td>
<td>0.11±0.20</td>
<td>0.13±0.22</td>
<td>0.14±0.26</td>
<td>0.14±0.24</td>
</tr>
<tr>
<td>DP AF A</td>
<td>0.07±0.13</td>
<td>0.21±0.34</td>
<td>0.20±0.32</td>
<td>0.21±0.33</td>
<td>0.19±0.30</td>
</tr>
</tbody>
</table>

Table 3.2: Probabilities $p_r = P(i_r = \hat{T}_r ± 1)$

<table>
<thead>
<tr>
<th>Target $T_r$</th>
<th>$r=2$</th>
<th>$r=4$</th>
<th>$r=6$</th>
<th>$r=8$</th>
<th>$r=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP</td>
<td>10.48±5.49</td>
<td>4.81±3.23</td>
<td>5.18±3.61</td>
<td>4.98±3.28</td>
<td>5.75±3.59</td>
</tr>
<tr>
<td>SAMW</td>
<td>13.95±6.04</td>
<td>5.16±3.50</td>
<td>5.09±3.31</td>
<td>5.41±3.02</td>
<td>4.56±2.38</td>
</tr>
<tr>
<td>ACA</td>
<td>10.88±5.61</td>
<td>4.18±2.73</td>
<td>4.21±2.83</td>
<td>4.12±2.70</td>
<td>6.13±3.27</td>
</tr>
<tr>
<td>RPAFA</td>
<td>17.94±6.33</td>
<td>7.72±4.43</td>
<td>6.67±3.87</td>
<td>5.89±3.44</td>
<td>8.78±5.28</td>
</tr>
<tr>
<td>DP AF A</td>
<td>10.52±5.49</td>
<td>4.79±3.14</td>
<td>5.02±3.31</td>
<td>4.89±3.14</td>
<td>5.51±3.32</td>
</tr>
</tbody>
</table>

Table 3.3: Mean one-stage costs $E(|i_r – \hat{T}_r|)$
3.3.2 Parametrized FH-MDP Algorithms

We retain the continuous time queuing model of flow control. We use $B = 1000$ and $T = 5$, and designate the ‘target states’ $\hat{T}_l$ as evenly spaced states within the queue i.e., $\hat{T}_l = B - \frac{l+1}{T+1}B$ that shift to 0 as the stage $l$ increases from 0 to 5. Thus, $\hat{T}_1 = 666$, $\hat{T}_2 = 500$, $\hat{T}_3 = 333$, $\hat{T}_4 = 166$ and $\hat{T}_5 = 0$, respectively. The one-step transition cost under a given policy $\pi$ is taken as $K_r(X_r, \pi, X_{r+1}) = |X_{r+1} - \hat{T}_{r+1}|$ with $K_T(i) = 0, \forall i \in S$. This function penalizes deviations from the target states $\hat{T}_{r+1}$, while at the same time maximizing throughput in the early stages ($l$ small) and minimizing queue length and hence the delay as one approaches end-of-horizon $T$.

We used 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 $i$ was:

$$\phi_{r,k}(i) = \left(1.0 + \frac{i - \hat{T}_r}{\hat{T}_r}\right)^{k-1}. \quad (3.30)$$

The averaging parameter $L$ was 50, implying 50 starts from each (state, stage) pair. We used $\delta(n) = n^{-0.167}$, $b(n) = n^{-0.55}$ whilst the actor stepsize $a(n)$ was $(nlmn)^{-1}$. We computed matrices $(\Phi_r \Phi_r)^{-1}$ needed in (3.24) before the algorithm commenced. To apply the DP algorithm, we discretized the interval $[0.05, 4.5]$ to obtain 25 equally spaced actions in each state. The transition matrix $\tilde{P}_T$ was computed using the method of (Ross, 2000, §6.8), which does not scale well in $|S|$ or $|U_{r,i}|$, hence the modest values of $|S| = 1000$ and $|U_{r,i}| = 25$. The computation time incurred in this matrix computation is included in the comparison of Table 3.4. Note, however, that the proposed algorithms require far less memory, as $P_T$ storage is eliminated. A comparison of the cost-to-go from each state upon using the three policies is shown in Figure 3.7, these match each other for the most part.

In Table 3.4, ‘Deviation’ corresponds to maximum deviation from the costs-to-go of Approximate DP observed in Figure 3.7 and ‘State’ indicates the state at which this deviation was observed. All costs-to-go shown are obtained from $10^4$ independent sample trajectories $\{X_0, X_1, ..., X_5\}$ for each initial state $X_0 = i$ and with a different initial seed.
### 3.3 Simulation Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Time in sec.</th>
<th>Deviation</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approximate DP</td>
<td>3090</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Algorithm 1</td>
<td>3923</td>
<td>31.67</td>
<td>532</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>2264</td>
<td>30.94</td>
<td>87</td>
</tr>
</tbody>
</table>

Table 3.4: Performance of FH-MDP algorithms

<table>
<thead>
<tr>
<th>Method</th>
<th>Time in sec.</th>
<th>Deviation</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value Iteration</td>
<td>3095</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td>3500</td>
<td>138.49</td>
<td>956</td>
</tr>
</tbody>
</table>

Table 3.5: Performance of Stopping-time algorithm

As seen in Figures 3.6a and 3.6b, as the stage \( r \) increases, the optimal policy shifts to the left because of the form of the cost function. All code was run on a Pentium III computer and was written in the C programming language.

For the case of optimal control of stopping time processes, the algorithm proposed in §3.2.4 is implemented on a finite buffer of size \( B = 1000 \). After starting from a state \( X_0 = i \), the first time that the set of states \( \{ \lceil \frac{B}{3} \rceil, \lceil \frac{B}{3} \rceil + 1, \ldots, \lceil 2\frac{B}{3} \rceil \} \) of this buffer is hit by the queue-length process \( X_t \) is the stopping time \( \tau(\pi(n), i) \), for a system operating under policy \( \pi(n) \). The cost incurred until \( \tau(\pi(n), i) \) is the sum of the one-step transition costs \( K(X_t, \pi_X(n), X_{t+1}) = \left| X_{t+1} - \frac{B}{2} \right| \) where \( 0 \leq t \leq \tau(\pi(n), i) \). The feature vectors \( \phi_k(i) \) used are just as (3.30), \( \forall i \in S \) and \( 1 \leq k \leq 4 \), with the modification that \( \hat{T}_r = \frac{B}{2} \), \( \forall 1 \leq r \leq T \). To speed up the first hit-time, we used \( \mu = 2.0 \) as a lower bound for states \( i \leq \frac{B}{2} \) during the execution of the algorithm, meaning that \( \lambda_c \in [2.0, 4.5] \). Similarly, we used 2.0 as the upper bound for states \( i > \frac{B}{2} \) resulting in \( U_i = [0.05, 2.0] \) for such states.

Figure 3.8a shows the policy computed using the value iteration (VI) algorithm where knowledge of the transition probabilities is required. The VI algorithm converged to optimal \( V_i(\pi^*) \), \( \forall i \in S \) in 50 iterations, to which we add the time required to compute the \( P_{\pi} \) transition matrices. The total cost is compared in Figure 3.8b, the costs-to-go for both policies being very similar, whilst performance is compared in Table 3.5.
3.3 Simulation Results

Figure 3.6: Optimal Policy computed using (a) Approximate DP (b) Algorithm 1

Figure 3.7: Comparison of the costs-to-go: Algorithm 1 vs. Approximate DP
3.3 Simulation Results

Figure 3.8: Algorithm 3: (a) Optimal Policy Using VI (b) Cost Comparison with VI
Chapter 4

Applications of Stochastic Approximation

Abstract

In §4.1 we propose five algorithms, all variants of Simultaneous Perturbation Stochastic Approximation (SPSA). The original one-measurement SPSA uses an estimate of the gradient of objective function $L$ containing an additional bias term not seen in two-measurement SPSA. As a result, the asymptotic covariance matrix of the iterate convergence process has a bias term. We propose a one-measurement algorithm that eliminates this bias, and has asymptotic convergence properties making for easier comparison with the two-measurement SPSA. The algorithm, under certain conditions, outperforms both forms of SPSA with the only overhead being the storage of a single measurement. We also propose a similar algorithm that uses perturbations obtained from normalized Hadamard matrices. The convergence w.p. 1 of both algorithms is established. We extend measurement reuse to design three second-order SPSA algorithms, sketch the convergence analysis and present simulation results on an illustrative minimization problem in §4.3.1.
4.1 SPSA Algorithms with Measurement Reuse

Next, in §4.2 we propose several stochastic approximation implementations for related algorithms in flow-control of communication networks, beginning with a discrete-time implementation of Kelly’s primal flow-control algorithm. Convergence with probability 1 is shown, even in the presence of communication delays and stochastic effects seen in link congestion indications. This ensues from an analysis of the flow-control algorithm using the asynchronous stochastic approximation (ASA) framework. Two relevant enhancements are then pursued: a) an implementation of the primal algorithm using second-order information, and b) an implementation where edge-routers rectify misbehaving flows. Also, discrete-time implementations of Kelly’s dual algorithm and primal-dual algorithm are proposed. Simulation results a) verifying the proposed algorithms and, b) comparing stability properties with an algorithm in the literature are presented in §4.3.2.

4.1 SPSA Algorithms with Measurement Reuse

Simultaneous Perturbation Stochastic Approximation (SPSA) is an efficient parameter optimization method that operates under the constraint that only noisy measurements of the objective function $L$ are available at each parameter iterate $\theta_k$. First proposed by Spall (1992), it involves making only two measurements of $L$ at each update epoch $k$ that are obtained by perturbing $\theta_k$ along random directions. A plethora of applications and enhancements of this technique can be found at Spall (2001). A variant of SPSA that reduces the number of function measurements made at each iteration $k$ from two to one and establishes the conditions under which a net lower number of observations suffice to attain the same Mean Square Error (MSE) is provided by Spall (1997). However, an impediment in rapid convergence to $\theta^*$ is that the algorithm constructs a gradient estimate of $L$ at $\theta_k$ that contains an additional error term over the scheme of Spall (1992) and that contributes heavily to the bias in the estimate. A solution to this problem was proposed by Bhatnagar et al. (2003) in the simulation optimization setting, where the perturbation to $\theta_k$ in the one-simulation case is generated in a deterministic manner. While this algorithm performs considerably better in practice, the asymptotic convergence properties in the
4.1 SPSA Algorithms with Measurement Reuse

setting of Spall (1992) and Spall (1997) were derived by Xiong et al. (2002) and found to be on par with those of one-measurement SPSA.

Here we first propose two first-order algorithms: one using randomly generated perturbations (cf. §4.1.1) and the other using deterministic perturbations (cf. §4.1.3). We show convergence w.p. 1 for both the algorithms. For the first algorithm, we also derive the asymptotic convergence properties and compare these with Spall (1992) (cf. Section 4.1.2). Further, we design three second-order algorithms based on the measurement-storage concept in Section 4.1.4. A numerical example justifies our findings (cf. §4.3.1).

The general structure of gradient descent algorithms is as follows. Let \( \theta_k = (\theta_{k,1}, \ldots, \theta_{k,p})^T \) where \( \theta_{k,i}, 1 \leq i \leq p \), are the \( p \) components of parameter \( \theta_k \). Let \( G_{k,l} \) be an estimate of the \( l \)-th partial derivative of the cost \( L(\theta_k), l \in \{1, 2, \ldots, p\} \). Then,

\[
\theta_{k+1,l} = \theta_{k,l} - a_k G_{k,l}(\theta_k), 1 \leq l \leq p, k \geq 0,
\]

where \( \{a_k\} \) is a step-size sequence. In the following, we refer to the one-measurement form of SPSA as SPSA2-1R and the two-measurement form as SPSA2-2R following the convention of Bhatnagar et al. (2003). In such a convention, the ‘R’ refers to perturbations which are randomly obtained, in contrast to deterministic perturbations in §3. The trailing ‘1’ in SPSA2-1R refers to the fact that at each iteration, the algorithm makes one measurement. The leading ‘2’ stands for a variant of the algorithm that makes parameter updates at every epoch, in contrast to algorithms like SPSA1-2R which update the parameter after an (increasingly large) number of epochs (cf. Bhatnagar and Borkar (1998)). The current parameter estimate \( \theta_k \) is perturbed with a vector \( \Delta_k = (\Delta_{k,1}, \ldots, \Delta_{k,p})^T \) to produce \( \theta_k^+ = \theta_k + c_k \Delta_k \), where \( c_k \) is a small step-size parameter that satisfies Assumption 5 (below) together with the step-size parameter \( a_k \) in (4.1). The gradient estimates...
4.1 SPSA Algorithms with Measurement Reuse

\[ G_{k,l}(\theta_k) \] used in SPSA2-1R are:

\[ G_{k,l}(\theta_k) = \frac{L(\theta_k^+)+\epsilon_k^+}{c_k\Delta_{k,l}} \]

\[ = \frac{L(\theta_k)}{c_k\Delta_{k,l}} + g_l(\theta_k) + \sum_{i=1, i \neq l}^{p} g_i(\theta_k) \frac{\Delta_{k,l}}{\Delta_{k,l}} \]

\[ + \frac{c_k^2 \Delta_{k,l}^T H(\theta_k) \Delta_{k,l}}{2c_k \Delta_{k,l}} + \frac{\epsilon_k^+}{c_k} + \frac{c_k^3 L^{(3)}(\bar{\theta}_k) \Delta_k \otimes \Delta_k \otimes \Delta_k}{6c_k \Delta_{k,l}}. \] (4.2)

We assume here that \( L \) is twice continuously differentiable with bounded third derivative. Note that \( H(\theta_k) \) is the Hessian evaluated at \( \theta_k \) and \( L^{(3)}(\bar{\theta}_k) \Delta_k \otimes \Delta_k \otimes \Delta_k = \Delta_{k,l}^T(L^{(3)}(\bar{\theta}_k) \Delta_k) \Delta_k \) where \( \bar{\theta}_k = \theta_k + \lambda_k c_k \Delta_k \) for some \( 0 \leq \lambda_k \leq 1 \) and \( L^{(3)} \) is the third derivative of objective function \( L(\cdot) \), where \( \otimes \) denotes the Kronecker product. Also, \( \epsilon_k^+ \) corresponds to a random additive noise component signifying that noisy observations of the cost are available for each parameter update. Thus, \( G_{k,l} \) is a random variable that we assume is measurable w.r.t. the \( \sigma \)-algebra \( \mathcal{F}_k = \sigma(\theta_i, \Delta_i, 0 \leq i \leq k-1, \theta_k) \). In contrast, \( g_l(\theta_k) \) is the \( l \)-th component of the derivative of \( L(\cdot) \) at \( \theta_k \). The gradient of \( L(\cdot) \) at \( \theta_k \) is now defined as \( g(\theta_k) = (g_l(\theta_k), 1 \leq l \leq p)^T \). Although not the current object of study, it is illustrative to observe that the estimate of SPSA2-2R needs two measurements of \( L(\cdot) \) about \( \theta_k \): \( G_{k,l}(\theta_k) = \frac{L(\theta_k^+)+\epsilon_k^+-L(\theta_k^-)}{2c_k \Delta_{k,l}} \). In the above, \( G_{k,l} \) uses function measurements at both \( \theta_k^+ \) and \( \theta_k^- = \theta_k - c_k \Delta_k \) and the measurement noise values at these points are \( \epsilon_k^+ \) and \( \epsilon_k^- \), respectively.

We retain all assumptions of Spall (1997), most of which are carried over from Spall (1992). Like Spall (1997), the key assumption requires the measurement noise \( \epsilon_k^+ \) to be mean 0: \( E(\epsilon_k^+|\theta_k, \Delta_k) = 0, \forall k \), and \( \text{var}(\epsilon_k^+) \to \sigma^2_\epsilon^+ \), where \( \sigma^2_\epsilon^+ \) is some finite constant. The step-size sequences used are of the form \( a_k = ak^{-\alpha} \) and \( c_k = ck^{-\gamma} \), respectively, where \( k \geq 1, a, c > 0 \), are given constants and with constraints on \( 0 < \gamma, \alpha \leq 1 \) such that the following assumption holds:

**Assumption 5** Stepsize \( a_k \) is s.t. \( \sum_{k=0}^{\infty} a_k = \infty \) and \( \sum_{k=0}^{\infty} \frac{a_k^2}{c_k^2} < \infty \).
4.1 SPSA Algorithms with Measurement Reuse

4.1.1 Algorithm SPSA2-1UR

The proposed algorithm also has a similar structure as SPSA2-1R and we call this algorithm SPSA2-1UR, the alphabet ‘U’ indicating ‘unbiased’. We utilize the noisy measurement already made at \( \theta_{k-1}^+ \), the storage of which results in unit space complexity.

Algorithm 1 (SPSA2-1UR)

\[
\theta_{k+1,l} = \theta_{k,l} - a_k \frac{L(\theta_k^+) + \epsilon_k^+ - L(\theta_{k-1}^+) - \epsilon_{k-1}^+}{c_k \Delta_{k,l}}
\]

where \( k \geq 0, 1 \leq l \leq p \). We have in the above:

\[
G_{k,l}(\theta_k) = \frac{L(\theta_k) - L(\theta_{k-1})}{c_k \Delta_{k,l}} + g_l(\theta_k) + \sum_{i=1, i \neq l}^{p} g_i(\theta_k) \frac{\Delta_{k,i}}{\Delta_{k,l}}
\]

\[
- \sum_{i=1}^{p} \frac{c_{k-1}}{c_k} g_i(\theta_{k-1}) \frac{\Delta_{k-1,i}}{\Delta_{k,l}}
\]

\[
+ \frac{c_k^2 \Delta_k^T H(\theta_k) \Delta_k - c_{k-1}^2 \Delta_{k-1}^T H(\theta_{k-1}) \Delta_{k-1}}{2c_k \Delta_{k,l}}
\]

\[
+ \frac{c_k^3 L^{(3)}(\bar{\theta}_k) \Delta_k \otimes \Delta_k \otimes \Delta_k}{6c_k \Delta_{k,l}}
\]

\[
- \frac{c_{k-1}^3 L^{(3)}(\bar{\theta}_{k-1}) \Delta_{k-1} \otimes \Delta_{k-1} \otimes \Delta_{k-1}}{6c_k \Delta_{k,l}}
\]

\[
+ \frac{\epsilon_k^+ - \epsilon_{k-1}^+}{c_k \Delta_{k,l}} \quad \text{(4.3)}
\]

Using a similar analysis as in Proposition 2 of Spall (1992), we identify below the order of convergence of the bias to 0:

**Lemma 7** Suppose for each \( k > K \) for some \( K < \infty \), \( \{\Delta_{k,i}\} \) are i.i.d., symmetrically distributed about 0, with \( \Delta_{k,i} \) independent of \( \theta_j, \epsilon_j^+ \), \( 1 \leq j < k \). Further let \( |\Delta_{k,i}| \leq \beta_0 \) a.s., \( E|\Delta_{k,i}^-| \leq \beta_1 \), and \( L \) be thrice continuously differentiable with \( |L^{(3)}_{i_1,i_2,i_3}| \leq \beta_2, \forall i_1, i_2, i_3 \in \{1,2,...,p\} \), for some constants \( \beta_0, \beta_1, \) and \( \beta_2 \). Then, \( E\{G_{k,l}(\theta_k)\} = g_l(\theta_k) + O(c_k^2) \), \( 1 \leq l \leq p \), a.s.
4.1 SPSA Algorithms with Measurement Reuse

Proof: The bias vector $b_k(\theta) = (b_{k,1}(\theta), b_{k,2}(\theta), \ldots, b_{k,p}(\theta))^T$ is defined as:

$$b_k(\theta_k) = E\{G_k(\theta_k) - g(\theta_k)|\theta_k\}, \quad (4.4)$$

where $G_k(\theta_k) = (G_{k,1}(\theta_k), G_{k,2}(\theta_k), \ldots, G_{k,p}(\theta_k))^T$. Due to the mean-zero assumption on $\epsilon_k^+$ and $\Delta_k^{-1}$ w.r.t $F_k$, we have $E\{\epsilon_k^+ - \epsilon_{k-1}^+|F_k, \Delta_k\} = 0$. It is crucial here to note that, despite the previous equality, $E\{\epsilon_k^+ - \epsilon_{k-1}^+|F_k, \Delta_k\} \neq 0$. Further, using the properties of independence, symmetry and finite inverse moments of perturbation vector elements (viz. $\Delta_k,l$, $\Delta_k,i$, and $\Delta_{k-1,i}$), observe that terms on the RHS of (4.3) have zero mean, with the exception of

$$b_{k,l}(\theta_k) = E\left\{\frac{c_k^2 L(3)(\bar{\theta}_k) \Delta_k \otimes \Delta_k \otimes \Delta_k}{6c_k \Delta_k,l}|F_k\right\}. \quad (4.5)$$

Observe that the bias term here is the same as for SPSA2-1R. The claim is now obtained by the arguments following Equation (3.1) in Lemma 1 of Spall (1992). In particular, note that

$$|b_{k,l}(\theta_k)| \leq \frac{\beta_2 c_k^2}{6} \sum_{i_1} \sum_{i_2} \sum_{i_3} E \left| \frac{\Delta_{k,i_1} \Delta_{k,i_2} \Delta_{k,i_3}}{\Delta_{k,l}} \right|$$

$$\leq \frac{\beta_2 c_k^2}{6} \left((p^3 - (p - 1)^3)\beta_0^2 + (p - 1)^3 \beta_1 \beta_0^3\right)$$

□

The relation in Lemma 7 is of value in establishing a form of asymptotic normality of the scaled iterate convergence process, see §4.1.2. It is interesting to note that Lemma 7 will not hold if normalized Hadamard matrix-based $\{\pm 1\}^p$-valued perturbations $\Delta_k$ that were first introduced by Bhatnagar et al. (2003) (§2.1.2 earlier has explained this deterministic perturbation method) are used. This is because there is no assurance that the term $\sum_{i=1}^p \frac{c_{k-1}}{c_k} g_i(\theta_{k-1}) \Delta_{k-1,i}$ will average to 0 as $k \to \infty$, unlike the previous term $\sum_{i=1,i\neq l}^p g_i(\theta_k) \Delta_{k,i}$. In such a case, a different method for unbiasing that does not use the immediate past measurement, in the spirit of §4.1.3 later, would be appropriate. A consequence of the a.s. convergence of the bias $b_k(\theta_k)$ is the strong convergence of the
iterates \( \theta_k \) to a local minimum \( \theta^* \). We now state Assumption A2 of Spall (1992) (that is also applicable to the setting of Spall (1997)):

**Assumption 6** Assume \( \exists \alpha_0, \alpha_1, \alpha_2 > 0 \) and \( \forall k, E\epsilon_k^+ \leq \alpha_0, EL^2(\theta_k^+) \leq \alpha_1, \Delta_{k,l}^2 \leq \alpha_2 \) a.s., for \( 1 \leq l \leq p \).

While this does not entail any difference, observe that we use \( \Delta_{k,l}^2 \leq \alpha_2 \) a.s. instead of the original \( E\{\Delta_{k,l}^2\} \leq \alpha_2 \) in Spall (1992).

**Lemma 8** Under assumptions of Spall (1997), as \( k \to \infty: \theta_k \to \theta^* \) a.s.

**Proof**: Follows almost verbatim as Proposition 1 of Spall (1992). The only modifications are due to a different error process \( \epsilon_k \), defined as \( \epsilon_k(\theta_k) = G_k(\theta_k) - E(G_k(\theta_k)|\theta_k) \). We can thus rewrite recursion (4.1) as:

\[
\theta_{k+1} = \theta_k - a_k(g(\theta_k) + b_k(\theta_k) + \epsilon_k(\theta_k)).
\]

The claim is obtained if the following conditions are satisfied:

(a) \( \|b_k(\theta_k)\| < \infty, \forall k \) and \( b_k(\theta_k) \to 0 \) a.s.

(b) \( \lim_{k \to \infty} P(\sup_{m \geq k} \|\sum_{i=k}^{m} a_i\epsilon_i(\theta_i)\| \geq \eta) = 0 \), for any \( \eta > 0 \).

where \( \|\cdot\| \) represents the Euclidean norm in parameter space \( \mathcal{R}^p \). Lemma 7 establishes (a) whilst for (b), notice that \( \{\sum_{i=k}^{m} a_i\epsilon_i\}_{m \geq k} \) is a martingale sequence (since \( E(\epsilon_{i+1}|\mathcal{F}_i) = 0 \)) and the martingale inequality gives:

\[
P(\sup_{m \geq k} \|\sum_{i=k}^{m} a_i\epsilon_i(\theta_i)\| \geq \eta) \leq \eta^{-2} E\left\| \sum_{i=k}^{\infty} a_i\epsilon_i \right\|^2 = \eta^{-2} \sum_{i=k}^{\infty} a_i^2 E\|\epsilon_i\|^2,
\]

since \( E(\epsilon_i^T\epsilon_j) = E(\epsilon_i^T E(\epsilon_j|\theta_j)) = 0, \forall j \geq i + 1 \). Further, for \( 1 \leq l \leq p \) using Hölder’s inequality: \( E\left(G_{i,l}^2(\theta_i)\right) \leq E(L(\theta_i^+ - L(\theta_{i-1}^+) + \epsilon_i^+ - e_i) \leq 2(\alpha_1 + \alpha_0)\alpha_2 c_i^{-2} \). Due to the mean-zero property of \( \epsilon_{i,l}(\theta_i) \), we have \( E\left(G_{i,l}^2(\theta_i)\right) = (g_i(\theta_i) + b_{i,l}(\theta_i))^2 + E(\epsilon_{i,l}^2(\theta_i)) \), thus having \( E(\epsilon_{i,l}^2(\theta_i)) \leq E(G_{i,l}^2(\theta_i)) \), and resulting in \( E\|\epsilon_i\|^2 \leq 2p(\alpha_1 + \alpha_0)\alpha_2 c_i^{-2} \). The square summability of \( \frac{e_i}{c_i} \) from Assumption 5 now establishes (b). \(\square\)
4.1 SPSA Algorithms with Measurement Reuse

4.1.2 Asymptotic Normality and Comparison

The results obtained so far aid us in establishing the asymptotic normality of a scaled iterate convergence process. We show that

\[ k^{\beta}(\theta_k - \theta^*) \xrightarrow{D} N(\mu, P\tilde{M}_1P^T) \]

as \( k \to \infty \) where the indicated convergence is in distribution, \( \beta = \alpha - 2\gamma > 0 \) (given \( 3\gamma - \frac{\alpha}{2} \geq 0 \)), and the mean \( \mu \) is the same as in SPSA2-2R (cf. [Spall, 1992, Proposition 2]) and SPSA2-1R. The orthogonal matrix \( P \) above is s.t. \( P^T aH(\theta^*) P = \text{Diag}(\{\lambda_i\}_{i=1}^p) \), \( \lambda_1, ..., \lambda_p \) being the \( p \) eigen values of \( aH(\theta^*) \). Unlike Spall (1997), \( \tilde{M}_1 \) above does not have an \( L^2(\theta^*) \) bias; however, it is scaled by a factor of 2. This factor arises due to the use of the additional noisy measurement \( L(\theta_{k-1} + \epsilon_k) + \delta_k \) in (4.3). In particular, \( \tilde{M}_1 = \frac{2}{\alpha^2c - \rho^2\sigma_r^2}\text{Diag}(\{(2\lambda_i - \beta_+)^{-1}\}_{i=1}^p) \), where \( \beta_+ = \beta \) if \( \alpha = 1 \) and 0 otherwise, and \( E\Delta_{k,\beta}^2 \to \rho^2 \). However, as confirmed by Spall (2005), the \( M_1 \) in (5) of Spall (1997) should be

\[ M_1 = a^2c - \rho^2(\sigma_r^2 + L^2(\theta^*)I)\text{Diag}(\{(2\lambda_i - \beta_+)^{-1}\}_{i=1}^p), \]  

(4.6)

and not \( a^2c - \rho^2(\sigma_r^2\text{Diag}(\{(2\lambda_i - \beta_+)^{-1}\}_{i=1}^p) + L^2(\theta^*)I) \) as printed, see Appendix A for a cursory derivation of \( M_1 \). Similarly, Appendix B establishes the form of \( \tilde{M}_1 \) and mean \( \mu \) that we claim.

We compare the proposed SPSA2-1UR with SPSA2-2R in the number of measurements of cost function \( L \), viz. \( \tilde{n}_1 \) and \( n_2 \), respectively. Like Spall (1997), we consider the case \( \alpha = 1 \) and \( \gamma = \frac{1}{6} \) (giving \( \beta = \frac{2}{3} \)) and \( E((\epsilon_k^+ - \epsilon_k^-)^2|\theta_k, \Delta_k) = 2\sigma_r^2 \), resulting in \( M_2 = \frac{1}{2}a^2c - \rho^2\sigma_r^2\text{Diag}(\{(2\lambda_i - \beta_+)^{-1}\}_{i=1}^p) \) and \( \tilde{M}_1 = 4M_2 \). This gives us:

\[ \frac{\tilde{n}_1}{n_2} \to \frac{1}{2} \left( \frac{4\text{tr}PM_2P^T + \mu^T\mu}{\text{tr}PM_2P^T + \mu^T\mu} \right)^{\frac{3}{2}}, \]  

(4.7)

where tr stands for the trace of the matrix. The ratio above depends upon quantity \( \mu \) and to achieve \( \tilde{n}_1 < n_2 \) we need that \( \mu^T\mu > \left( \frac{4 - \frac{2}{\alpha^2c - \rho^2\sigma_r^2}}{2^2 - 1} \right) \text{tr}PM_2P^T \approx 4.11\text{tr}PM_2P^T \). We use \( n_1 \) to denote the number of measurements of \( L \) made by SPSA2-1R. In the special case
4.1 SPSA Algorithms with Measurement Reuse

$L(\theta^*) = 0$, it is shown by (Spall, 1997, eq. (8)) that $n_1 < n_2$ when $\mu^T \mu > 0.7024 \text{tr} PM_2 P^T$. While our result does not compare favorably, the advantage is that (4.7) holds for all values of $L(\theta^*)$.

The comparison with SPSA2-1R yields an interesting rule of thumb. Using $D_\lambda$ to represent the diagonal matrix $\text{Diag}(\{2\lambda_l - \beta_+\}_{l=1}^p)$, we have:

$$\frac{\tilde{n}_1}{n_1} \rightarrow \left( \frac{\text{tr} PM_1 P^T + \mu^T \mu}{\text{tr} P M_1 P^T + \mu^T \mu} \right)^\frac{3}{2} = \left( \frac{2a^2 c^{-2} \rho^2 \sigma^2 \text{tr} PD_\lambda P^T + \mu^T \mu}{a^2 c^{-2} \rho^2 (\sigma^2 + L^2(\theta^*)) \text{tr} PD_\lambda P^T + \mu^T \mu} \right)^\frac{3}{2}.$$  

Irrespective of $\mu$, $D_\lambda$, and $P$ (quantities that may require substantial knowledge of the system), it suffices to have $L^2(\theta^*) > \sigma^2_i$ to ensure that $\frac{\tilde{n}_1}{n_1} \leq 1$. The experimental results in §4.3.1 provide verification of these claims.

4.1.3 Algorithm SPSA2-1UH

We now propose a fast convergence algorithm by modifying SPSA2-1H of (Bhatnagar et al., 2003, §3). The key departure in SPSA2-1H from gradient estimate (4.2) of SPSA2-1R is that perturbation vectors $\Delta_k$ are now deterministically obtained from normalized Hadamard matrices. For clarity, we reproduce here briefly some of the discussion from §2.1.2 earlier which described these deterministic perturbations. The kind of matrices considered are the following: let $H_2$ be a $2 \times 2$ matrix with elements $H_2(1,1) = H_2(1,2) = H_2(2,1) = 1$ and $H_2(2,2) = -1$. Likewise for any $q > 1$, let the block matrices $H_{2^q}(1,1)$, $H_{2^q}(1,2)$, and $H_{2^q}(2,1)$ equal $H_{2^{q-1}}$. Also, let $H_{2^q}(2,2) = -H_{2^{q-1}}$. For a parameter of dimension $p$, the dimension of the Hadamard matrix needed is $2^q$ where $q = \lceil \log_2 (p + 1) \rceil$.

Next, $p$ columns from the above matrix $H_q$ are arbitrarily chosen from the $q - 1$ columns that remain after the first column is removed. The latter column is removed as it does not satisfy a key property of the perturbation sequence. Each row of the resulting $q \times p$ matrix $\hat{H}$ is now used for the perturbation vector $\Delta_k$ in a cyclic manner, i.e. $\Delta_k = \hat{H}(k\%q + 1)$, where $\%$ indicates the modulo operator. Though not shown here,
the convergence of SPSA2-1H can be shown as a special case of Prop. 2.5 of Xiong et al. (2002). The proposed algorithm, which we call SPSA2-1UH, has two steps:

Algorithm 2 (SPSA2-1UH)

1. For $k \geq 0$, $1 \leq l \leq p$,
   \[ \theta_{k+1,l} := \theta_{k,l} - a_k \frac{L(\theta_k^+) + \epsilon_k^+ - \bar{L}_k}{c_k \Delta_{k,l}} \]

2. if $k \% q = 0$, $\bar{L}_k := L(\theta_k^+) + \epsilon_k^+$ else $\bar{L}_{k+1} := \bar{L}_k$.

In the above, $\bar{L}_k$ changes only periodically in epochs of size $q$ and the algorithm has a unit space requirement. Given index $k$, define $\bar{k} = \max\{m : m < k, m \% q = 0\}$. For SPSA2-1UH, (4.2) is now modified to:

\[
\begin{align*}
G_{k,l}(\theta_k) &= \frac{L(\theta_k) - L(\theta_k)}{c_k \Delta_{k,l}} + g_l(\theta_k) + \sum_{i=1, i \neq l}^{p} \frac{\Delta_{k,i}}{\Delta_{k,l}} g_i(\theta_k) - \sum_{i=1}^{p} \frac{c_k \Delta_{k,i}}{c_k \Delta_{k,l}} g_i(\theta_k) \\
&\quad + O(c_k) + \frac{c_k \epsilon_k^+ - \epsilon_k^+}{c_k \Delta_{k,l}},
\end{align*}
\]

where $O(c_k)$ contains higher order terms. Since $\Delta_{k,l} = 1$, we have $\frac{\Delta_{k,i}}{\Delta_{k,l}} = \frac{1}{\Delta_{k,l}}$, $\forall 1 \leq i, l \leq p$ and $\forall k$. Therefore, as in Lemma 3.5 of Bhatnagar et al. (2003), the fourth term above averages to 0 over $q$ steps as $k \rightarrow \infty$, thus settling the problem posed in §4.1.1. In passing, we also note that step 2 can be written as $k \% q = m$ for any given $m$ s.t. $0 \leq m \leq q - 1$.

Convergence Analysis

We can now formally establish convergence w.p. 1 of $\theta_k$. The original SPSA2-1H algorithm can be expanded as follows

\[
\begin{align*}
\theta_{k+1} &= \theta_k - a_k \Delta_{k,l}^{-1} \Delta_{k,l}^{-1} g(\theta_k + \lambda_k c_k \Delta_k) \\
&\quad - a_k c_k L(\theta_k) \Delta_{k,l}^{-1} - a_k c_k \epsilon_k^+ \Delta_{k,l}^{-1}, \tag{4.8}
\end{align*}
\]
4.1 SPSA Algorithms with Measurement Reuse

where \( 0 \leq \lambda_k \leq 1 \). Here \( \Delta^{-1} \) is the vector \( \Delta^{-1} = (\frac{1}{\Delta_k, \ldots, \frac{1}{\Delta_{k,p}}})^T \). This recursion is now presented in the manner of (Xiong et al., 2002, eq. (6)), with \( r_k, d_k \) and \( e_k^+ \) there replaced by \( \Delta^{-1}, \Delta_k \) and \( \epsilon_k^+ \), respectively:

\[
\theta_{k+1} = \theta_k - a_k g(\theta_k) - a_k \Delta^{-1}_k \Delta^T_k \left\{ g(\theta_k + \lambda_k c_k \Delta_k) - g(\theta_k) \right\} - a_k \left\{ \Delta^{-1} \Delta^T_k - I \right\} g(\theta_k) - \frac{a_k}{c_k} \epsilon_k^+ \Delta^{-1}_k.
\]

In the manner of (4.8), the SPSA2-1UH recursion is written as:

\[
\theta_{k+1} = \theta_k - a_k \Delta^{-1}_k \Delta^T_k g(\theta_k + \lambda_k c_k \Delta_k) - \frac{a_k}{c_k} (L(\theta_k) - L(\theta^+_k)) \Delta^{-1}_k - \frac{a_k}{c_k} (\epsilon_k^+ - \epsilon_k^+) \Delta^{-1}_k.
\]

which can be expanded as

\[
\theta_{k+1} = \theta_k - a_k g(\theta_k)
- a_k \Delta^{-1}_k \Delta^T_k \left\{ g(\theta_k + \lambda_k c_k \Delta_k) - g(\theta_k) \right\}
- a_k \left\{ \Delta^{-1} \Delta^T_k - I \right\} g(\theta_k)
+ a_k \Delta^{-1}_k \Delta^T_k \left\{ g(\theta_k + \lambda_k c_k \Delta_k) - g(\theta_k) \right\}
+ a_k \Delta^{-1}_k \Delta^T_k g(\theta_k)
- \frac{a_k}{c_k} (L(\theta_k) - L(\theta^+_k) - \epsilon_k^+ + \epsilon_k^+) \Delta^{-1}_k.
\]

However, we need to make a non-restrictive assumption:

**Assumption 7** The function \( g \) (cf. A1 of Xiong et al. (2002)) is uniformly continuous.

**Theorem 4** Under Assumption 7 and from Spall (1997), Algorithm 2 produces iterates \( \theta_k \) s.t. \( \theta_k \to \theta^* \) w.p. 1.
Proof: We first show that terms in (1.10) and (1.11) are error terms in the nature of $e_i(\theta)$ in condition (b) of Lemma 8. In particular, we show that these satisfy the conditions (B1) and (B4) of Xiong et al. (2002). We reproduce these two conditions for clarity:

(B1) $\lim_{n \to \infty} \left( \sup_{n \leq k \leq m(n,T)} \left\| \sum_{i=n}^{k} a_i e_i \right\| \right) = 0$, for some $T > 0$, where

$$m(n,T) \triangleq \max \{ k : a_n + \ldots + a_k \leq T \}.$$

(B4) There exist sequences $\{e_{1,n}\}$ and $\{e_{2,n}\}$ with $e_n = e_{1,n} + e_{2,n}$ for all $n$ such that $\sum_{k=1}^{n} a_k e_{1,k}$ converges, and $\lim_{n \to \infty} e_{2,n} = 0$.

Due to $\lim_{k \to \infty} c_k = 0$ and uniform continuity of $g$, $\Delta_k^{-1} \Delta_k^T \{ g(\theta_k + \lambda_k \Delta_k) - g(\theta_k) \}$ satisfies (B4). Since $\lim_{k \to \infty} \frac{\theta_k - \theta_{k-1}}{\Delta_k} = 0$, $\Delta_k^{-1} \Delta_k^T g(\theta_k)$ satisfies (B1). This is shown by applying Lemma 2.2 of Xiong et al. (2002) with the substitution $\{x_n\}$ where $x_n = 1$ for all $n \geq 1$, $\{\Delta_n^{-1} \Delta_n\}$ and $\{g(\theta_n)\}$ for $\{c_n\}$, $\{r_n\}$, and $\{e_n\}$, respectively. Also $\forall k$,

$$\left\| (L(\theta_k) - L(\theta_{k+1})) - (L(\theta_{k+1}) - L(\theta_{k+2})) \right\| \leq \frac{|L(\theta_k) - L(\theta_{k+1})|}{c_k} I_{k\%q=0} + \frac{|L(\theta_{k+1}) - L(\theta_{k+2})|}{c_k} I_{k\%q=0}.$$

We consider the first term on the RHS, the second follows similarly:

$$\frac{|L(\theta_k) - L(\theta_{k+1})|}{c_k} I_{k\%q=0} \leq \left( \frac{M_0}{c_k} \sum_{m=k}^{k-1} \| \theta_{m+1} - \theta_m \| + \frac{M_0}{c_k} \sum_{m=k}^{k-1} M_1 \frac{a_m}{c_m} + M_2 \frac{a_m}{c_m} \| \epsilon_m^+ \| \right) I_{k\%q=0}$$

$$\leq \left( \frac{M_0 M_1 q a_k}{c_k} \frac{\epsilon_k}{c_k} + \frac{M_0 M_2 a_k}{c_k} \sum_{m=k}^{k-1} \| \epsilon_m^+ \| \right) I_{k\%q=0},$$

where $M_0$, $M_1$, and $M_2$ represent appropriate bounds. The summability of $\left\{ \frac{a_k q_k}{c_k} \right\}$ is obtained using Assumption 5 - implying that the LHS satisfies (B1). This fact is used when we apply Lemma 2.2 of Xiong et al. (2002) again (with $\{\Delta_n^{-1}\}$, $\{L(\theta_n) - L(\theta_n)\}$ replacing $\{r_n\}$ and $\{e_n\}$, respectively, and $\{c_n\}$ as is) to see that $\frac{L(\theta_k) - L(\theta_k)}{c_k} \Delta_k^{-1}$ satisfies (B1). We now consider the last term viz. $\frac{\epsilon_k + \epsilon_{k-1}}{c_k} \Delta_k^{-1}$. However, now the noise term $\epsilon_{k,l} = \frac{\epsilon_k + \epsilon_{k-1}}{c_k} \Delta_k^{-1}$. However, now the noise term $\epsilon_{k,l}$ is not mean 0 w.r.t. $F_k$ but letting $k = k + q$, $\forall k$ we see that $E(\epsilon_{k,l} | F_k) = 0$. This results in $\left\{ \sum_{k=n}^{m} \frac{a_k \epsilon_k}{c_k} \right\}_{m \geq n}$ being a martingale sequence w.r.t. $F_n$, where we again
utilize the inequality
\[
P \left( \sup_{m \geq \tilde{n}} \left\| \sum_{k=\tilde{n}}^{m} \frac{a_k}{c_k} \bar{\epsilon}_k \right\| \geq \eta \right) \leq \eta^{-2} \sum_{k=\tilde{n}}^{\infty} \left( \frac{a_k}{c_k} \right)^2 E \| \bar{\epsilon}_k \|^2,
\]
the LHS modified to obtain
\[
\leq \eta^{-2} \sum_{k=\tilde{n}}^{\infty} \left( \frac{a_k}{c_k} \right)^2 E \| \bar{\epsilon}_k \|^2 + P \left( \sup_{h \geq m \geq n} \left\| \sum_{k=n}^{m} \frac{a_k}{c_k} \bar{\epsilon}_k \right\| \geq \eta \right).
\]
The square summability of \( \frac{a_k}{c_k} \) and boundedness of \( \bar{\epsilon}_k \) result in quantities on the RHS vanishing as \( n \to \infty \). The proof of Proposition 2.3 of Xiong et al. (2002) handles the terms in the RHS of (4.9), thus resulting in the claim. \( \square \)

### 4.1.4 Second-Order Algorithms

We now propose three second-order SPSA algorithms, all of which re-use noisy function measurements. The first algorithm - called 2SPSA-3UR since it is a modification of 2SPSA of Spall (2000) - makes three measurements in the vicinity of each iterate \( \theta_k \) and re-uses the current gradient estimate \( G_k(\theta_k) \) to estimate the Hessian matrix \( H_k(\theta_k) \) at \( \theta_k \). The second algorithm 2SPSA-2UR makes two measurements at \( \theta_k \) and reuses the value \( L(\theta_{k-1}^+) \) in the Hessian matrix estimation. The third algorithm (2SPSA-1UR) makes one measurement per iteration. Second-order SPSA algorithms, which are stochastic analogs of the Newton-Raphson algorithm, are also proposed by Spall (2000) and Bhatnagar (2005). In particular, the three algorithms that we propose can be seen as modifications of the algorithms 3SA, 2SA and 1SA of Bhatnagar (2005) with the following properties:

- Unlike the 2SPSA of Spall (2000), all three algorithms 2SPSA-nUR \( n = 1, 2, 3 \) use an additional \( a_k \)-like step-size sequence \( \{b_k\} \) (not to be confused with the bias term \( b_k(\theta_k) \) in Lemma 7) in the recursion to compute \( H_k \). Such an additional step-size \( \{b_k\} \) is employed in all the four second-order SPSA algorithms described by Bhatnagar (2005). The property of \( b_k \) relative to \( a_k \) is the well-known ‘two-timescale’ property: \( \sum_k b_k = \infty \), \( \sum_k b_k^2 < \infty \) and \( a_k = o(b_k) \). Bhatnagar (2005) observes that, in the algorithms, the Hessian should be averaged (and its inverse computed) on a faster timescale. Thus on the timescale on which the parameter
4.1 SPSA Algorithms with Measurement Reuse

$\theta_k$ is updated, the Hessian updates $H_k$ would be seen as having converged. This provides good algorithmic behavior.

- Similar to 2SPSA, we employ an auxiliary perturbation sequence $\{\tilde{\Delta}_k\}$ with the same properties as the original $\{\Delta_k\}$, although independently generated. There is also an associated scaling parameter $\{\tilde{c}_k\}$. We will also require an analog of Assumption 5: replace the pair $(a_k, c_k)$ in Assumption 5 with the pairs $(a_k, \tilde{c}_k)$, $(b_k, \tilde{c}_k)$, $(b_k, c_k)$.

- We use the ‘unbiasing’ concept by storing past or current measurements of $L$ and gradient estimate $G$. However, unlike the unit storage overhead in SPSA2-1UR and SPSA2-1UH, this retention of the current estimate of gradient $G$ arguably costs $O(p)$ in storage. It must be mentioned here that second-order algorithms of Spall (2000) and Bhatnagar (2005) do implicitly assume memory to store, manipulate and multiply Hessian estimates $H_k$ - which are $O(p^2)$ data structures.

- The algorithm 2SPSA-1UR is distinct from the single-simulation algorithm 1SA proposed by Bhatnagar (2005) due to the use of the sum $\Delta_k + \tilde{\Delta}_k$ as perturbation in the gradient estimate $G_k$.

2SPSA-3UR

The proposed algorithm can be viewed as a modification of the 3SA three-measurement algorithm of Bhatnagar (2005). Also used by Bhatnagar (2005), the function $\Gamma$ used below maps from the set of general $p \times p$ matrices to the set of positive definite matrices. There are many possible candidates for such a $\Gamma$, as explained in §II-D of Spall (2000) where the notation $f_k$ is used.

$$\begin{align*}
\theta_{k+1} &= \theta_k - a_k H_k^{-1} G_k(\theta_k) \\
H_k &= \Gamma(\bar{H}_k) \\
\bar{H}_k &= \bar{H}_{k-1} + b_{k-1}(\bar{H}_k - \bar{H}_{k-1})
\end{align*}$$ (4.12)
where

\[
\hat{H}_k = \frac{1}{2} \left[ \frac{\delta G_k^T}{c_k \Delta_k} + \left( \frac{\delta G_k^T}{c_k \Delta_k} \right)^T \right]
\]

\[
\delta G_k = G_k^1(\theta_k^+)-G_k(\theta_k).
\]

There is a re-use of the current gradient estimate \(G_k(\theta_k)\) in the latter recursion. This estimate is computed as in the algorithm SPSA2-2R. In addition to \(\theta_k^+\) and \(\theta_k^-\), we now employ the shorthand notation \(\theta_k^{++} = \theta_k + c_k \Delta_k + \tilde{c}_k \tilde{\Delta}_k\). Similarly, we denote the measurement noise incurred at \(\theta_k^{++}\) as \(\epsilon_k^{++}\).

\[
G_k^1(\theta_k^+) = \frac{\tilde{\Delta}_k^{-1}}{c_k} \left( L(\theta_k^{++}) + \epsilon_k^{++} - L(\theta_k^+ + \epsilon_k^+) \right)
\]

\[
G_k(\theta_k) = \frac{\Delta_k^{-1}}{2c_k} \left( L(\theta_k^+) + \epsilon_k^+ - L(\theta_k^-) - \epsilon_k^- \right)
\]

Appendix C contains the derivation regarding \(E(\hat{H}_k|\mathcal{F}_k) = H(\theta_k) + O(c_k)\). The convergence analysis of \(\theta_k \to \theta^*\) proceeds as Bhatnagar (2005), outlines of which we explain here. We construct a time-axis using the step-size \(b_k\): assume that \(t(n) = \sum_{m=0}^n b_m\) and define a function \(H(\cdot)\) as \(H(t(k)) = H_k\) with linear interpolation between \([t(k), t(k+1))\).

Similarly define function \(\theta(\cdot)\) by setting \(\theta(t(k)) = \theta_k\) and linear interpolation on the interval \([t(k), t(k+1)]\). Let \(T > 0\) be a given scalar and next define a sequence \(\{T_k\}\) as \(T_0 = 0 = H(0)\) and \(T_k = \min\{t(m)|t(m) \geq T_{k-1} + T\}\). Then \(T_k = t(m_k)\) for some \(m_k\) and \(T_k - T_{k-1} = T\). Now define functions \(\bar{H}(\cdot)\) and \(\bar{\theta}(\cdot)\) as \(\bar{H}(T_k) = H(t_{m_k}) = H_k\) and \(\bar{\theta}(T_k) = \theta(t_{m_k}) = \theta_k\), and for \(t \in [T_k, T_{k+1}]\), the evolution is according to the system of ODEs:

\[
\dot{\hat{H}}_{i,j}(t) = \nabla_{i,j}^2 L(\bar{\theta}(t)) - \hat{H}_{i,j}(t)
\]

\[
\dot{\bar{\theta}}(t) = 0,
\]
where $\nabla_{i,j}^2 \mathcal{L}(\theta(t))$ indicates $\frac{\partial^2 \mathcal{L}(\theta)}{\partial \theta_i \partial \theta_j}$. As in Lemma A.8 of Bhatnagar (2005), we have
\[
\lim_{k \to \infty} \sup_{t \in [T_k, T_{k+1}]} \| H(t) - \hat{H}(t) \| = 0 \quad \text{and} \quad \lim_{k \to \infty} \sup_{t \in [T_k, T_{k+1}]} \| \theta(t) - \bar{\theta}(t) \| = 0.
\]
Recursion (4.12) can now be shown to asymptotically track the trajectories of the ODE
\[
\dot{\theta}(t) = -H^{-1}(\theta(t)) \nabla \mathcal{L}(\theta(t))
\]
in a similar manner as above on the slower timescale $\{a_k\}$ as in Theorem 3.1 of Bhatnagar (2005).

**2SPSA-2UR**

The proposed algorithm is a modification of the gradient-free four-measurement algorithm 2SPSA which was described by Spall (2000). In that article (cf. footnote 6), the SPSA-2R analog of 2SPSA was not considered due to the inherent variability of both estimates $G_k$ and $H_k$ if the one-measurement form of SPSA were to be used. We employ the technique of the proposed SPSA-2UR to unbias the estimates, and thus reduce the number of function measurement required from 4 to 2. The family of recursions (4.12) is retained with the differences that

\[
G_k(\theta_k) = \frac{\tilde{\Delta}^{-1}}{c_k} \left( L(\theta_{k+}^+) + \epsilon_{k+}^+ - L(\theta_{k}^+) - \epsilon_{k}^+ \right)
\]

\[
\hat{H}_k = \frac{1}{2} \left[ \frac{\delta G_k^T}{c_k \Delta_k} + \left( \frac{\delta G_k^T}{c_k \Delta_k} \right)^T \right]
\]

\[
\delta G_k = G_k^1(\theta_k^+) - G_k^1(\theta_k),
\]

followed by a correction of only the diagonal terms in $\hat{H}_k$:

\[
\hat{H}_k(i, i) := \hat{H}_k(i, i) + \frac{L(\theta_k^+) + \epsilon_k^+ - L(\theta_{k-1}^+) - \epsilon_{k-1}^+}{c_k^2}
\]

where the measurement at $\theta_{k-1}^+$ is reused. This correction assumes that $\Delta_k$ and $\tilde{\Delta}_k$ are both Bernoulli distributed over $\{+1, -1\}$, however, a similar corrective term can be derived for other classes of perturbations. Appendix D derives the steps leading to this
correction. In the above, \( \tilde{G}_k^1(\theta_k) = \frac{L(\theta_k^+)+\epsilon_k^+ - L(\theta_{k-1}^+)-\epsilon_{k-1}^+}{c_k} \Delta_k^{-1} \) and \( G_k^1(\theta_k^+) \) is as in 2SPSA-3UR. Note that \( \tilde{G}_k^1(\theta_k) \) is precisely the gradient estimate in the algorithm SPSA2-1UR of \( \S4.1.1 \). Also, \( G_k(\theta_k) = G_k^1(\theta_k^+) \) indicating that a re-use of the gradient estimate \( G_k \) is being made to compute the Hessian estimate \( \hat{H}_k \). Here, \( G_k \) is computed using a one-sided difference just as in 2SA of Bhatnagar (2005). Such an estimate still uses 2 measurements, and is different from the one-measurement form of \( G_k \) as in SPSA2-1R or the unbiased \( G_k \) of SPSA2-1UR proposed earlier in \( \S4.1.1 \).

While we do not show a detailed convergence analysis, we observe the following arguments.

\[
E(G_k^1(\theta_k^+)-\tilde{G}_k^1(\theta_k)|\theta_k, \Delta_k) = E(G_k^1(\theta_k^+)|\theta_k, \Delta_k) - E(\tilde{G}_k^1(\theta_k)|\theta_k, \Delta_k)
\]

\[
= g(\theta_k^+)-L(\theta_k^+)-L(\theta_{k-1}^+)-\epsilon_{k-1}^+ \Delta_k^{-1} \frac{c_k}{c_k}.
\]

We make the claim that, modulo the correction \( \text{(4.13)} \) in the diagonal elements,

\[
E \left( g(\theta_k^+)-\frac{L(\theta_k^+)-L(\theta_{k-1}^+)-\epsilon_{k-1}^+ \Delta_k^{-1}}{c_k} (\Delta_k^{-1})^T | \mathcal{F}_k \right) = H(\theta_k) + O(c_k),
\]

the proof being in Appendix D.

Note in the above RHS that \( H(\theta_k) \) is the Hessian at \( \theta_k \) and the trailing term corresponds to a matrix with an induced norm bounded above by \( O(c_k) \). We can also write the above as: \( E \left( \frac{\delta G_k}{c_k \Delta_k | \mathcal{F}_k} \right) = H_{i,j}(\theta_k) + O(c_k), 1 \leq i, j \leq p. \) The convergence analysis uses the ODE technique discussed previously, and since \( G_k \) is the same as algorithm 2SA of Bhatnagar (2005), convergence of \( \theta_k \) is assured using Theorem 3.3 of Bhatnagar (2005).

The proof of convergence of the proposed algorithm can also be obtained in a manner similar to that of Theorems 1a and 2a of Spall (2000). Note that Spall (2000) uses the step-size \( b_{k+1} = \frac{1}{k+1} \). Our algorithm is applicable for more general step-sizes as long as the requirement \( a_k = o(b_k) \) is met.

A possibility in 2SPSA-2UR is to use gradient estimate \( G_{k,l} = \frac{L(\theta_k^+)+\epsilon_k^+ - L(\theta_{k-1}^+)-\epsilon_{k-1}^+}{c_k} \Delta_k^{-1} \) as in \( \text{(4.3)} \). The convergence can be shown by modifying \( \text{(49)} \) of Bhatnagar (2005) using
\[
\frac{L(\theta_k^+) - L(\theta_{k-1}^+)}{\epsilon_k} \Delta_k^{-1} (\text{as in SPSA2-1UR}) \quad \text{instead of the original.} \quad \text{The advantage in such an estimate of } G_k \text{ is that a less noisy set of terms are shown equal to zero in their mean by the property of conditional expectation. In particular, the terms in the } G_k \text{ estimate of 2SA involve both } \Delta_k \text{ and } \tilde{\Delta}_k \text{ whilst the modified } G_k \text{ of 2SPSA-2UR (as in (4.3)) has only } \Delta_k.
\]

### 4.1.5 2SPSA-1UR

Here we propose a one-measurement algorithm which is also a second-order SPSA algorithm as the two previous algorithms. The method has similarities with algorithm 1SA of Bhatnagar (2005), the crucial difference lying in design of the perturbations \(\Delta_k\) and \(\tilde{\Delta}_k\) and consequently, the estimation of \(G_{k,l}\). We will require that \(\Delta_k + \tilde{\Delta}_k \neq 0\) w.p. 1 and that \(c_k = \tilde{c}_k\). The random variable \(\Delta_k + \tilde{\Delta}_k\) would then be symmetric, mean-0 and bounded away from 0. This can be ensured, for example, by using Bernoulli-distributed random variables \(\Delta_k \in \{-1, +1\}\) and \(\tilde{\Delta}_k \in \{-0.5, 0.5\}\). The only measurement made at each iteration \(k\) is at \(\theta_k^{++}\). We retain the algorithm in (4.12), except that \(G_k(\theta_k^+)\) and \(G_k(\theta_k)\) are identified as below:

\[
G_k(\theta_k^+) = \frac{\tilde{\Delta}_k^{-1}}{c_k} \left( L(\theta_k^{++}) + \epsilon_k^{++} - L(\theta_{k-1}^{++}) - \epsilon_{k-1}^{++} \right)
\]

\[
G_k(\theta_k) = \frac{(\Delta_k + \tilde{\Delta}_k)^{-1}}{c_k} \cdot \left( L(\theta_k^{++}) + \epsilon_k^{++} - L(\theta_{k-1}^{++}) - \epsilon_{k-1}^{++} \right)
\]

We identify the changes required using results associated with 1SA of Bhatnagar (2005). First, Proposition A.12 of Bhatnagar (2005) is retained almost verbatim except for corresponding notation changes. In particular, the original statement that shows

\[
\left| E \left[ \left( \frac{L(\theta_k^{++})}{c_k^2} \Delta_k^{-1} \right)^T \Delta_k^{-1} | \mathcal{F}_k \right] - H(\theta_k) \right| \to 0,
\]
is replaced by the analogous
\[
E \left[ \left( \frac{L(\theta_{k+1}^+)}{c_k^2} \Delta_{k-1}^{-1} - \frac{L(\theta_{k-1}^+)}{c_{k-1}^2} \tilde{\Delta}_{k-1}^{-1} \right)^T \Delta_{k-1}^{-1} | \mathcal{F}_k \right] - H(\theta_k) \to 0.
\]

An elementary modification of the proof of Theorem 3.4 of Bhatnagar (2005) suffices to show convergence of the parameter \( \theta_k \). We modify (51) of Bhatnagar (2005) to place the following term as LHS:
\[
\frac{L(\theta_{k+1}^+)}{c_k} - \frac{L(\theta_{k-1}^+)}{c_{k-1}} (\Delta_k + \tilde{\Delta}_k)^{-1}.
\]

A key point to note with regard to the current algorithm is that it is possible to avoid storage of the size-\( p \) vector \( \frac{L(\theta_{k+1}^+)}{c_k} \Delta_{k-1}^{-1} \) above by storing the noisy function measurement \( L(\theta_{k-1}^+)^+ c_k^+ \) (done otherwise to un bias the \( G_{k,l} \) measurement in 2SPSA-1UR) and then generate the vector \( \tilde{\Delta}_{k-1}^{-1} \) afresh at each step \( k \). However, the variability produced using the actual estimate may be lesser. Further, it is interesting to note that the storage of the size-\( p \) vector can be separated into two parts, the first involving unit storage for \( L(\theta_{k-1}^+)^+ c_k^+ \). The second part involves storing the auxiliary perturbation \( \tilde{\Delta}_{k-1} \), which in the Bernoulli case is a mere two-symbol string of size \( p \) that yields to efficient compression. Thus the \( O(p) \) space requirement is conservative.

## 4.2 Network Flow-Control using Stochastic Approximation

### 4.2.1 Problem Description

For a network with finite sets of sources \( R \) and links \( L \), the network utility-maximization problem is given by

\[
\max_{x_r} \sum_{r \in R} U_r(x_r) \ \text{s. t.} \ \sum_{s \in R^l} x_s \leq c_l, \forall l \in L, \quad (4.14)
\]

where \( c_l > 0 \) is the capacity of link \( l \) and \( x_r \) is the rate at which the source \( r \) sends traffic into the network. The utility function \( U_r \) is positive-valued, strictly concave, and
differentiable, and is termed as ‘elastic’ utility due to its concavity. The set \( R^l \subset R \) is the set of sources that transmit over link \( l \). An approximation to this problem is the unconstrained maximization problem with the following objective function:

\[
V(x) = \sum_{r \in R} U_r(x_r) - \sum_{l \in L} \int_0^{\sum_{s \in R^l} x_s} p_l(y) dy, \quad (4.15)
\]

where \( p_l \) are penalty functions that act as proxy for the link capacity constraints in (4.14). Also, the \(|R|\)–dimensional vector \( x \) above is \((x_r, r \in R)^T\).

Studied by Kelly et al. (1998) for a simpler case, the continuous-time primal flow-control algorithm tunes the sending rate \( x_r(t) \) using the ODE:

\[
\begin{align*}
\dot{x}_r(t) &= \kappa_r(x_r)(U'_r(x_r(t)) - q_r(t)), \quad \forall r \in R \\
q_r(t) &\triangleq \sum_{l \in L_r} \sum_{s \in R^l} x_s(t) \quad (4.16)
\end{align*}
\]

where \( \kappa_r(x_r) > 0 \) is a suitable gain and \( L_r \subset L \) is the set of links that carry the traffic of \( r \). The contribution to \( q_r(t) \) from each link \( l \in L_r \) can be interpreted as a price based on the aggregate flow of all sources \( s \in R^l \) being carried over \( l \), whilst \( U'_r(x_r(t)) \) above is the derivative of \( U_r \) at \( x_r(t) \). A crucial assumption is that \( q_r(t) \) in (4.16) is autonomous and is not an explicit function of \( t \), as it depends only on \( x(t) \). However, there are exceptions such as Adaptive Virtual Queue of Kunniyur and Srikant (2002), where \( p_l(t) = p_l(\sum_{s \in R^l} x_s(t), \tilde{c}_l(t)) \) where \( \tilde{c}_l(t) \) is the virtual buffer-size. The timescale along which \( \tilde{c}_l(t) \) evolves can be assumed as faster, and thus we still have the correspondence \( \tilde{c}_l(t) \equiv \tilde{c}_l(\sum_{s \in R^l} x_s(t)) \).

The algorithm (4.16) is implemented in a distributed manner, since each source \( r \) only knows prices charged by links \( l \) carrying its traffic. Under intuitive conditions on \( p_l \) and \( U_r \), there exists a globally asymptotically stable equilibrium point \( x^*_r \) of the above ODE. Due to the penalty function method (4.15), however, \( x^*_r \) is only an ‘approximately fair’ equilibrium.
For a source $r$ that implements (4.16), delays in link congestion indications are experienced due to network congestion or propagation time. The price demanded of $r$ by link $l$ could be $p_l(\sum_{s \in R_l} x_s(t) - \xi^l_s(t) - \psi^s_l(t))$ and not the current value $p_l(\sum_{s \in R_l} x_s(t))$, where $\xi^l_s(t)$ (resp. $\psi^s_l(t)$) is the $\mathbb{R}_+^-\text{-valued}$ feedback (resp. feedforward) delay from link $l$ to source $r$ (resp. from source $s$ to link $l$). Past work assumes propagation delays to dominate queueing delays in large-scale networks, and hence $\xi^l_s(t)$ and $\psi^s_l(t)$ can be held constant in $t$. Using this assumption, and for $U_r(x_r) = w_r \ln x_r$ and $\kappa_r(x_r) = \kappa_r x_r$ where $\kappa_r, w_r > 0$, the impact of such delays is analyzed by [Kelly et al. (1998)] to guarantee local stability alone of (4.16), with the disadvantage that modifications for faster convergence to equilibrium $x_r^*$ may compromise this stability. Past work does not distinguish between links $l$ either, and defines the delay as $T_r = \max_{l \in L_r} (\xi^l_t + \psi^l_t)$ (which is the RTT seen by source $r$). Note that $T_r$ may not be an upper-bound for the cumulative delay $\{\xi^l_t + \psi^l_t\}$, $l \in L_r$, $s \in R^l$ of the autonomous variety we considered above in (4.16). Thus use of $T_r$ does not capture the delays in sending-rate updates, as reflected via the link congestion notifications.

Past investigations, however, probe if (4.16) is stable despite delays $T_r$, and whether this stability is achieved with only minimal restrictions on any of the parameters involved. The work of [Johari and Tan (2001)] assumes homogeneous sources, i.e. $T_r \equiv T$ and $U_r \equiv U$ for all $r \in R$, and a sufficient condition relating gain $\kappa$ and $T$ is proposed. Heterogeneous users are considered by [Vinnicombe (2002)], and a sufficient condition proposed where the product $\kappa_r T_r$ is bounded by a constant and the stability assured about $x_r^*$ is local. This bound assumes a fixed RTT $T_r$ while [Liu et al. (2005)] suggest that such an assumption is not always reasonable, especially upon use of $T_r$ as an independent variable than as an argument. For example, window-size $W_r$ computation is as per the thumb-rule: $x_r = \frac{W_r}{T_r}$ and here RTT variability cannot be disregarded, whilst in RTT’s appearance as argument e.g. in the terms $x_s(t - \xi^l_t(t) - \psi^s_l(t))$ above, where $T_r(t) \approx \xi^l_t(t) + \psi^l_s(t)$ it can be held fixed to $T_r$.

However, not only is a source required to explicitly compute such a $T_r$, the resulting transients due to AQM-induced queueing delays during such a computation would result
in variability and congestion. TCP-Vegas is analyzed by Liu et al. (2005), where the queueing delay itself is taken as the price \( p_i \) demanded by a link. Thus the independent variable \( q_r \) is the total delay over the route, is affected by the individual AQM algorithms at the links \( l \), and could cause stability-impacting transients in (4.16). Another finding of Liu et al. (2005) was that RTT variability would affect only window-based congestion-control schemes. We note that schemes like Vinnicombe (2002) described above require the product \( \kappa_r T_r \) to be upper-bounded for local stability - thus implying that the problem with RTT variability indirectly extends to rate-based controllers also. The advantage with the proposed Asynchronous Stochastic Approximation framework is that there are no restrictions on the controller parameters that relate directly to the delay phenomena. Also note in light of (Liu et al., 2005, Remark 1), the proposed ASA framework permits non-causality in link-price notifications.

In La and Anantharam (2002), two algorithms that solve (4.14) in an exact fashion are proposed. The first of these, Algorithm-I, is an add-on module to the algorithm of Mo and Walrand (2000) that computes target queue sizes \( p_r(n) \) (these are indicative of the user’s contribution to resources’ capacity utilization at iteration \( n \) of the algorithm). At iteration \( n \), \( p_r(n) \) are plugged into a \((p, \alpha)\) proportionally fair algorithm of Mo and Walrand (2000) whose equilibrium point \( w(n) \) corresponds to window-sizes \( w_r(n) \) that a user \( r \) must maintain. While Algorithm-I is in fact discrete-time (we reproduce it here for clarity);

\[
p_r(n) = p_r(n - 1) + \frac{\hat{p}_r(n) - p_r(n - 1)}{M + 1}, \tag{4.17}
\]

we note that the term \( \hat{p}_r(n) \) used here is a solution to an optimization problem which in turn requires knowledge of \( p_s(n) \) of the other sources, \( \forall s \in R \) - data not necessarily known to user \( r \). A guess of the constant \( M \) above also subscribes to the same caveat. While adapted to a more general setting, the algorithm for such flow control (inspired by the Cohen-Grossberg model) in Borkar (2007) makes the same assumption.
Also, Algorithm-II of (La and Anantharam, 2002, eq. (16)-(18)), which proposes direct window-size update rules minus the target queue-size computations $p(n)$ of Algorithm-I may have delay-related issues. As in the $(p, 1)$ algorithm of Mo and Walrand (2000), there is use of round trip delay term $d_i(t)$. While there is no analysis of this algorithm in the face of communication delays, inferences could still be drawn from similar analyses of TCP Vegas. As mentioned previously, due to variations in RTT, a successor of TCP Vegas termed Stabilized Vegas has had its conclusions contested by Liu et al. (2005).

The delay-robustness analysis of Fan et al. (2004), carried out using the passivity framework of feedback-control, requires the source controller to adjust the gain $\kappa$ to suit the delay regime if the delays are larger than a certain threshold. This adjustment of Fan et al. (2004, eq. (32)) requires, among others, a derivative $\zeta$ of the price $p_l$ - a closed form of which, in keeping with the distributed nature of (4.16), may not be available to the sources. Indeed, we employ an indirect, simulation-based, method to obtain derivatives of $p_l$ in §4.2.4 below. The online tuning of $\kappa$ (and the corresponding dual-algorithm gain $\Gamma$) has been mentioned by Fan et al. (2004) as a promising avenue - we adopt this suggestion by making $\kappa$ an Ideal Tapering Stepsize (ITS) in our framework.

Note that we do not propose any new rate controller or utility function as such, only a discrete-time implementation of the primal algorithm whose ITS property makes it robust to (possibly varying) delays. The former approach is adopted by Ranjan et al. (2006), where a delay-independent stability criterion is proposed provided certain relations between $p_l$ and $U_r$ hold. These results characterize invariance, i.e. the continued evolution of (4.16) within a region, and stability (convergence to $x_r^*$) of the non-linear delay differential equations that result. In particular, any family of $p_l$ and $U_r$ that imply a stable market equilibrium are also sufficient for global stability of the system (4.16), when affected by delays $T_r$. A similar regime on $p_l$ and $U_r$ is proposed by Ying et al. (2006), who use the Razumikhin criterion for stability. This latter work also shows that, while only assuring local-stability, product conditions on $\kappa_r T_r$ of Johari and Tan (2001) are also more conservative than the sufficient conditions on $p_l$ and $U_r$ for global stability.

For the algorithm that we propose in §4.2.3, the basic assumptions of Kelly et al.
4.2 Network Flow-Control using Stochastic Approximation

(1998), viz. a) $U_r$ are positive, strictly concave and differentiable, $\forall r$ and b) link price functions $p_l(y)$ are non-negative, continuous and increasing in $y$ are appended with the following.

**Assumption 8** Functions $U'_r$ and link costs $p_l$ are Lipschitz continuous.

We point here to the analogous Assumption 1 of Ranjan et al. (2006) and La and Ranjan (2006), although only a single link is considered in both these investigations.

### 4.2.2 Proposed Implementation

The algorithm (4.16) can be implemented in discrete-time, and three such implementations are identified in the subsection *Time Lags* of Kelly et al. (1998). Stability is also analysed in such a framework, e.g. Johari and Tan (2001) investigate the stability of delayed difference equations arising from one such implementation. The implementations, however, are not strictly asynchronous since a global clock tick $t$ is assumed to index all the recursions. Instead, we propose a controller at source $r$ that uses a private, *local* index $t_r$:

$$x_r[t_r + 1] := x_r[t_r] + \kappa_r(t_r)(U'_r(x_r[t_r])) - q_r[t_r]),$$

followed by $t_r := t_r + 1$. Reminiscent of the ‘self-clocking’ advantage of window-based congestion controllers, sources $r$ (or even links $l$) need not know the global clock tick $t$.

We explain the asynchronous nature of such a scheme, which is captured in $t_r$, where $t_r \in \mathbb{Z}^+$. Assume a counting process where for $t \in \mathcal{R}_+$, $t_r(t)$ indicates the number of updates made at source $r$ until time $t$, the elapsed time between updates $t_r$ and $t_r + 1$ being an estimate of RTT $T_r$. The quantity $T_r$ can be assumed fixed as Vinnicombe (2002), Ranjan et al. (2006) and Deb and Srikant (2004) do but we permit $T_r$ to be varying as discussed previously. Similar to $t_r(t)$, every link $l$ is seen as having made $t^l(t)$ measurements of the (time-varying) aggregate flow $\sum_{s \in R_l} x_s(t)$ over it. These link measurements could also be update schemes, eg exponentially weighted moving averages in RED - which serve as a low-pass filter on the measurement time-series (cf. La and Ranjan (2006)).
In §4.2.5, we provide an example of a link update scheme. Random variable $\xi^l_r$, which differs in interpretation from its $R_+\text{-valued analog}$ in continuous-time, represents the feedback delay. Abusing the notation for $p_l$, the price demanded of source $r$ by link $l$ is the $(t_r - \xi^l_r)$-th iterate $p_l[t_r - \xi^l_r]$. Thus, $\xi^l_r$ takes values in the entire integer set $\mathbb{Z}$.

For instance, Deb and Srikant (2004) consider $p_l[t_r - \xi^l_r] + \epsilon^l_r$ as the number of Explicit Congestion Notification (ECN) marks $M_r[t_r]$ observed in the current epoch $t_r$ of source $r$. Kelly et al. (1998) consider a congestion notification scenario where links $l$ send a Poisson number of indications to sources $r$ at the rate $p_l(\cdot)$ and therefore $M_r[t_r]$ is an RV with mean $p_l[t - \xi^l_r]$. TCP congestion controllers (in their more general form as minimum potential delay controllers) are modelled by Deb and Srikant (2004) in like manner. If considered time-invariant, $\kappa_r[t_r] \equiv \kappa_r$ plays a role in characterizing stability about the equilibrium point $x^*$ (cf. Vinnicombe (2002), (Srikant, 2004, §4), Ying et al. (2006)). Instead, in §4.2.3 we propose a $\kappa_r[t_r]$ that is an Ideal Tapering Stepsize (ITS) of Borkar (1998).

We also consider noise $\epsilon^l_r$ as an additive stochastic effect in the congestion indication scheme, making $p_l[t_r - \xi^l_r] + \epsilon^l_r$ as the price demanded at the $t_r$-th update of $r$. Additive disturbances in the measurement at links $l$ are reflected in $p_l$ and these correspond to bursty mice or uncontrolled traffic: e.g. flows like UDP and CBR (of the ATM framework) that do not adhere to any flow-control algorithm. Such additive noise effects have warranted an analysis separate from delays (cf. Fan et al. (2004)). Further, averaging of such packet-level dynamics using a low-pass filter at the links $l$ (similar in spirit to Random Early Detection) and the impact of such averaging on the delay-robustness of the controller of Ranjan et al. (2006) is studied by La and Ranjan (2006).

In §4.2.4 and §4.2.5 below, we pursue enhancements to the proposed implementation of §4.2.3. Since (4.16) can be viewed as gradient ascent along (4.15), §4.2.4 proposes using the second-order information via an efficient variant of the gradient-descent Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm proposed by Abdulla and Bhatnagar (2006). A modification to the method of Abdulla and Bhatnagar (2006) is needed in order to handle delays $T_r(t)$ above, thus proposing the first asynchronous variant of SPSA to our knowledge. An outline of proof is given.
4.2 Network Flow-Control using Stochastic Approximation

The algorithm proposed in §4.2.5 considers the problem of Fan et al. (2005) where a misbehaving flow $r$ must be rectified by a router $\hat{r}$ placed at the edge of the network using suitably rigged costs $\hat{q}_r$. In §4.2.6 we propose similar discrete-time implementations of the dual and primal-dual algorithms (cf. Srikant, 2004, §2). In particular, the primal-dual algorithm is implemented using a two-timescale ASA method adapted from the constrained Markov Decision Process (MDP) algorithm of Borkar (2005) which updates Lagrange Multipliers on the slowest timescale. A regime of step-sizes (known as the two-timescale condition) at links and sources permits simultaneous updates towards the respective optima. The delay-robustness of such two-timescale ASA has already been established in the analysis of algorithms for Markov Decision Processes (cf. Konda and Borkar (1999)).

In the numerical results of §4.3.2 we perform two experiments. First, we verify the convergence of all the proposed algorithms in a single link-multiple nodes setting with simulated packet arrivals and departures. The rate allocations to which the proposed algorithms converge are only negligibly away from $x^*$. We then consider a system from Ranjan et al. (2006) where the corresponding delayed-differential equation model of the system was shown to be unstable. But when flow-control is implemented using the proposed algorithm, convergence w.p. 1 is assured.

4.2.3 Primal Algorithm using ASA

It is necessary to motivate the use of ASA for at least the simple case of fixed, but heterogeneous, RTTs $T_r$. The proposed algorithm is, however, designed for varying RTTs $T_r(t)$. Suppose that $t \in \mathcal{R}_+$ is the update instant of source $r$ (i.e. $t = n_r T_r$ for some $n_r \in \mathbb{Z}^+$). Then, $t_s(t) \neq t_r(t)$, in general, due to the differing $T_s$s. Moreover, the values of $x_s$ seen by the controller of source $r$ via $p_l$, $l \in L_r$ need not be $x_s[t_s(t)]$ (i.e. the most recent value of $x_s$) and due to propagation delays, could be some past value of $x_s$. In terms of the local indices of $s$, this iterate may be $x_s[t_r - \xi^l_r - \psi^l_s]$ where both $\xi^l_r, \psi^l_s \in \mathcal{Z}$.

In the proposed algorithm, we require that ITS $\kappa_r[t_r]$ be identical for all $r$ (i.e. $\kappa_r[t_r] \equiv$
4.2 Network Flow-Control using Stochastic Approximation

Thus we have the recursion:

\[ x_r[t_r + 1] := x_r[t_r] + \kappa[t_r](U'_r(x_r[t_r]) - q_r[t_r]) \]  

\[ q_r[t_r] := \sum_{l \in L_r} (p_l[t_r - \xi'_r] + \epsilon'_r[l_t]) , \]

with index update \( t_r := t_r + 1 \). Here, the iterate \( q_r[t_r] \) estimates the charge payable by source \( r \) with stochastic effects \( \epsilon'_r[l_t] \) being noise in the estimate. The ITS conditions on the stepsizes \( \kappa \), for all sources \( r \), are:

\[ \kappa[t_r] > 0 \forall t_r \quad , \quad \sum_{t_r} \kappa[t_r] = \infty , \]

\[ \sum_{t_r} \kappa^{1+q}[t_r] < \infty \quad \text{for some} \quad q \in (0,1) . \]

The proposed ITS nature of \( \kappa[t_r] \) is an issue, since conventional flow control algorithms assume stationary behaviour independent of index \( t_r \) and we have not yet come across any algorithms using diminishing stepsizes. However, \cite{Srikant2004, §2}, for instance, assumes that \( \kappa \) is a function of rate \( x_r(t) \) (in particular, a model suited to TCP-Reno assumes \( \kappa \equiv \kappa_r x_r(t - T_r) \)) whilst \cite{Fan2004} suggests an online tuning approach. The use of ITS is crucial to apply the theory of ASA in the current setting.

We introduce a notational simplification in the form of a global event-index \( n \) that denotes events: whether these be measurements of aggregate flow by links \( l \in L \) or rate updates by sources \( r \in R \). For every \( t \in R_+ \), \( n(t) \) will indicate the counting point-process of all events. We use parentheses \( (\cdot) \) to identify a variable according to this global index \( n \) whilst square brackets \( [\cdot] \) will denote ‘with respect to local indices’ \( t_r \). For each value of \( n \geq 0 \) there are corresponding local indices \( t_r \Xi n \) and \( t' \Xi n \) such that \( t_r, t' \leq n \) and \( \sum_{r \in R, l \in L} t_r + t' = n \). The global index \( n \) thus can be constructed from local indices \( t_r \) and \( t' \). Also, if \( x_r(n) \neq x_r(n - 1) \) (implying that an update of \( x_r \) has taken place at \( n - 1 \)), then \( x_r(n + k) = x_r(n), \forall k : t_r(n + k) = t_r(n) \). Analogously define the process \( p_l(n) \) for links \( l \).
Convergence Analysis

For ease of analysis, assume that at a given instant \( t \) if a measurement occurs at \( l \in L \), no rate update occurs at any \( r \in R \). This can be accomplished using a technique to unfold the recursions proposed by Borkar (1998, §3). There can, however, be multiple rate updates at any instant. For each source \( r \), define two vectors 
\[
\bar{x}_r(n) = (x_{s,r}(n), \forall l \in L_r, \forall s \in R^l)^T
\]
and 
\[
\epsilon_r(n) = (\epsilon^l_r(n), \forall l \in L_r)^T.
\]
Here, 
\[
x_{s,r}(n) = x_s(n - \xi^l_r - \psi^l_s) \quad \text{with } Z^+-\text{valued random variables } \xi^l_r, \psi^l_s \text{ being delays in terms of } n.
\]
Though we re-use symbols \( \xi \) and \( \psi \) denoting delays, the parenthesis (\( \cdot \)) makes the context clear. The total delay in ‘sensing’ \( x_s \) by source \( r \), via link \( l \), is given by \( \tau^l_{s,r} = \xi^l_r + \psi^l_s \). Similarly, \( \epsilon^l_r(n) \) is the noise seen at instant \( n \) in measurement \( p_l \) communicated by link \( l \) to source \( r \). Thus, the update can be written as:
\[
x_r(n + 1) := x_r(n) + \kappa(n, r)F_r(\bar{x}_r(n), x_r(n), \epsilon_r(n))I_{\{r \in \varphi_n\}},
\]
where \( \{r \in \varphi_n\} \) corresponds to the event that \( x_r \) is updated at \( n \) and \( \sum_{k=1}^{n} I_{\{r \in \varphi_k\}} = t_r \).

Further, \( \kappa(n, r) = \kappa[t_r] \) and \( F_r \) is the ‘reinforcement’ term \( (U'_r(x_r[t_r]) - q_r[t_r]) \) in the recursion (4.18). It is possible that \( \varphi_n = \phi \), the empty set, if \( n \) corresponds to a link measurement instant.

The analysis follows the pattern and notation of Borkar (1998): (Borkar, 1998, Lemma 3.3) is modified as Lemma to accommodate the delays \( \tau^l_{s,r} \). Treatment of Borkar (1998) handles only a single delay \( \tau^l_{s,r} \) per pair of components \( (s, r) \) whereas here, per source \( r \) we have \( \sum_{l \in L_r} |R^l| \) such delays including ‘self-delays’ of the form \( \tau^l_{s,s} \). We verify assumptions (A1)-(A6) of Borkar (1998) and modify where necessary to accommodate \( \tau^l_{s,s} \). Assumption (A1) follows from the ITS property of \( \kappa[t_r] \). Further, (A2) holds due to \( |R| < \infty \) and the upper-bound on Retransmission Time-out (RTO) (due to which component \( x_r \) is updated regularly). We define the increasing sigma-fields 
\[
\mathcal{F}_n \triangleq \sigma(x_r[\hat{m}], \tau^l_{s,s}(m), \epsilon^l_r(m), \hat{m} \leq n, m < n, r \in R, l \in L_r, s \in R^l)
\]
in order to rephrase (A3):
4.2 Network Flow-Control using Stochastic Approximation

Assumption 9 Delays $\tau^l_{s,r}(n) \in \{0, 1, ..., n\}$, $\forall r \in R$, $l \in L_r$, $s \in R^l$ and $\exists b, C > 0$ s.t.

$$E((\tau^l_{s,r}(n))^b | F_n) \leq C \text{ a.s.}$$

It is sufficient to have the realistic sup$_n \tau^l_{s,r}(n) < \infty$, w.p. 1, to satisfy this assumption. Assumption 8 made above results in (A4) being satisfied. The next assumption is on the stochastic effects $\epsilon^l_{r}(n)$, that we first define:

$$f_r(\bar{x}_r(n), x_r(n)) = \int F_r(\bar{x}_r(n), x_r(n), \epsilon_r) P_r(di_r).$$

where $P_r(\cdot)$ is the law according to which $\epsilon_r$ is distributed. Note that $P_r$ may, in general, depend upon $\bar{x}_r(n)$ although what we require is:

Assumption 10 If $\tau^l_{s,r} = 0$, $\forall r \in R$, $l \in L_r$, $s \in R^l$ then

$$f_r(\bar{x}_r(n), x_r(n)) = U'_r(x_r(n)) - q_r(n).$$

Mean-zero $\epsilon^l_{r}(n)$ suffices to satisfy this assumption, and the Poisson congestion indication of Kelly et al. (1998) is an example. We abuse notation to consider the function $f_r(x(n)) = f_r(\bar{x}_r(n), x_r(n))$ in the absence of delays as in the LHS of the assumption above. Then, the system of ODEs asymptotically tracked by recursion (4.18) is

$$\dot{x}_r(n) = f_r(\bar{x}_r(n), x_r(n)), \forall r \in R.$$ 

Thus (A5) is satisfied since the set $J$ of equilibrium points required by (A5) contains the single element $x^*$, the equilibrium allocation. Further, (A6) is satisfied by the strict Liapunov function $V(x)$ defined in (4.15).

Let $\hat{F}(n) = E(F(n)|F_n)$ and $f^s : R^{[R]|x|R|\times|L|} \mapsto R^{[R]}$ be a function which we describe as follows. Given $s \in R$, let $\tilde{x}_s(n)$ be an element of $R^{[R]|x|R|\times|L|}$ with components $(\tilde{x}_s(n))_{r,l} = x^l_{r,s}(n)$ if $r, s \in R^l$ and all other components zero. Consider $\hat{x}(n) \Delta (x_s(n), \tilde{x}_s(n)), s \in R)$ and thus $f^s$ is such that $f^s_s(\hat{x}(n)) = f_s(x_s(n), x_s(n))\delta_{s,r}$ for $r, s \in R$ where $\delta_{s,r}$ is the Kronecker delta function. Let the analogous $\tilde{f}^s$ be defined when delays $\tau^l_{s,r}$ are zero $\forall r, s$. The stepsize $b(n) \Delta \max_r \kappa(n, r)$ for $n \geq 0$. 


4.2 Network Flow-Control using Stochastic Approximation

Lemma 9 For $\varphi_n \neq \phi$, almost surely, $\exists K_1 > 0$, and a random $N \geq 1$ s.t. for $n \geq N$:

$$\|\bar{f}^{\varphi_n}(\hat{x}(n)) - \hat{F}(n)\| \leq K_1 b^q(n)$$

Proof: Consider the upper-bound $\|f_s(x)\| < K_2, \forall x$. Let $\hat{F}_r = f_{\varphi_n}^{\hat{x}(n)}$, and $c = 1 - q$ (for $q$ from (4.19) above). Modify (3.5) of Borkar (1998) as follows:

$$E\{||\bar{f}^{\varphi_n}(\hat{x}(n)) - \hat{F}_r(n)|| |F_n\}$$

$$\leq E\{||\bar{f}^{\varphi_n}(\hat{x}(n)) - \hat{F}_r(n)|| I_{{\forall (l,s,r): \tau_{ls,r}(n) \leq b-c(n)}} |F_n\}$$

$$+ E\{||\bar{f}^{\varphi_n}(\hat{x}(n)) - \hat{F}_r(n)|| I_{{\exists (l,s,r): \tau_{ls,r}(n) > b-c(n)}} |F_n\}.$$ 

By Assumption 2 (the $b$ and $C$ from which we use) and the modified Chebyshev inequality, the second term is a.s. bounded by $2K_2 C |R|^2 |L|^b c(n)$. The remainder of the proof now follows as in Lemma 3.3 of Borkar (1998). In passing, we mention that the assumption $\tau_{r,r} = 0$ made in (A3) of Borkar (1998) is not required.

Theorem 5 For trajectories with $\|x(n)\|_\infty < \infty, \forall n$, algorithm (4.18) converges to $x^*$ a.s..

Proof: We verify two conditions in the statement of Theorem 3.1 (a) of Borkar (1998). The first condition is to check if $\exists a > 0$ s.t. $\dot{x}(t) = f(x(t))$ is an $a-$robust system. This requires showing that, for a globally asymptotically stable system $\dot{x}(t) = f(x(t))$, the system $\dot{x}(t) = M(t)f(x(t))$ with $M(t) > a$ is also similarly stable. To see this, we note that the strict Liapunov property of $V(x)$ and the distributed form of (4.16) ensures that $\nabla_r V(x)f_r(x) < 0$ for any point $x \neq x^*$. Thus, for any $a > 0$, $a\nabla_r V(x)f_r(x) < 0$, implying the system is $a-$robust.

Next, construct a sequence $t_n$ where $t_n \triangleq \sum_{k=0}^{n} \kappa(k, \varphi_k)$. Here we have assumed that $\varphi_k$ has at most one element of $R$. For a given $t > 0$, $\bar{\mu}^t(s), s \in R$ is $I_{\{\varphi_n = s\}} I_{\{t \in [t_n, t_{n+1})\}}$. Further, $a-$thickness of the continuous-time process $\bar{\mu}^t$ is defined as $\bar{\mu}_r^t(s) > a, \forall s \in R$. We refer to Borkar (1998, §3) for detailed definitions. The second condition of the theorem
4.2 Network Flow-Control using Stochastic Approximation

is to check that all limit points of $\bar{\mu}^t$ in $U$ are $a-$thick a.s.. Note that the limit points of $\bar{\mu}^t$ correspond to the update frequencies of the sources $r$ at time $t \in \mathbb{R}^+$. Upper bounds on RTO mentioned earlier along with bounded delays $\tau_{l,r}$ ensure that as $t \to \infty$, $\bar{\mu}^t(r) > a$, $\forall r \in R$ for some $a > 0$. Thus limit points of $\bar{\mu}^t$ are $a-$thick, and the claim follows. □

The statement of the theorem is qualified, in that it assumes bounded trajectories $x_r(n)$. This is implemented by clipping $x_r(n)$ against an interval $C_r = [x_r,\min, x_r,\max]$ such that $x_r^*$ is contained in this interval, a reasonable guess being $C_r = [0, \min_{l \in L_r} c_l]$. Note similar bounds and justification for using them by Ranjan et al. (2006) and La and Ranjan (2006). Consider $C = \prod_{r \in R} C_r$, then the projected ODE that results will nevertheless be asymptotically stable. This is because the unique asymptotically stable equilibrium point $x^*$ is either contained within $C$ or else if it lies outside of $C$, the ODE will get trapped at a boundary point of $C$ (thus introducing spurious fixed points).

4.2.4 Second-order Primal Algorithm

The primal algorithm can be interpreted as a gradient ascent along the Liapunov function $V(x)$ of (4.15) above. Therefore, the diagonal elements of the Hessian matrix of $V(x)$ can also be computed in a distributed manner: $\nabla_{r,r}^2 V(x) = U_r''(x_r) - \sum_{l \in L_r} \nabla_{r} p_l(\sum_{s \in R_l} x_s)$. Apart from $U_r$, also assume that $p_l(\cdot)$ are $C^2$ (i.e. twice continuously differentiable). This is a pre-requisite since the gradient terms $\nabla_{r} p_l(\sum_{s \in R_l} x_s)$ are estimated using an efficient variant of SPSA proposed by Abdulla and Bhatnagar (2006). Though $\nabla_{r} p_l(\cdot)$ is of identical value $\forall r \in R_l$, each source needs to compute its own estimate. In the description that follows, we shall observe that for a source $r$, estimating $\nabla_{r} p_l(\cdot)$ separately for each $l \in L_r$ will not be required.

The estimate $\nabla_{r} p_l(\sum_{s \in R_l} x_s)$ is constructed by measuring $p_l$ such that the rate of source $r$ is not $x_r[t_r]$ but a perturbed value $x_r^+[t_r] = x_r[t_r] + \delta_r[t_r] \Delta_r[t_r]$. The scalar $\delta_r$ is a positive, diminishing step-size with other properties explained later, while perturbations $\Delta_r[t_r]$ are independent, $\pm 1-$valued with probability 0.5. Both $\delta_r$ and $\Delta_r$ are the local information of source $r$. A crucial departure from SPSA-based gradient search occurs due to the asynchronous nature of the update scheme. The measurement $p_l(n - \xi_r^t)$ seen at
4.2 Network Flow-Control using Stochastic Approximation

update $t_r$ does not reflect the rate $x_r^+[t_r - 1]$ set at the last update $t_r - 1$ since link $l$ has not yet experienced this change in flow (or the measurement of such a change has not reached $r$ yet).

Thus an asynchronous variant of SPSA is proposed here. This algorithm would also be of independent interest in the SPSA framework. Assume that current measurement of $p_l$ has a contribution from $x_r^+[t_r - k_r^l]$ and that $k_r^l \leq K_r, \forall l \in L_r$ with a non-zero probability. In the absence of precise methods to determine $K_r$, it can be taken as 1 if time between updates at $r$ is distributed closely about the fixed RTT $T_r$. Let $\vec{x}_{l,r}(n)$ be the $|R_l|-$sized vector $(x_{s}(n - \tau_{s,r}^l), \forall s \in R^l)^T$ and let $\vec{\delta}_{l,r}(n)$ be $(\delta_{s}(n - \tau_{s,r}^l)\Delta_s(n - \tau_{s,r}^l), \forall s \in R^l)^T$ with $\delta_{l,r}(n, s)$ indicating components. Now we define $\vec{x}_{l,r}^+(n)$ as $\vec{x}_{l,r}(n) + \vec{\delta}_{l,r}(n)$. Note that at $t_r$ the offsets $k^l_r, \forall l \in L_r$, are unknown. We choose from the ‘word’ $A_r(t_r, K_r)$ given below, and our choice is a correspondence for each link $l$ in $L_r$ i.e. $\delta_r[t_r - k_r]\Delta_r[t_r - k_r] = \delta_r(n - \tau_{r,s}^l)\Delta_r(n - \tau_{r,s}^l)$ where $\tau_{r,s}^l \in Z^+$. This we abbreviate as $\vec{\delta}_{l,r}(n)$. We also abbreviate the noisy measurement $p_l(\vec{x}_{l,r}^+(n)) + \epsilon_n^l(n)$ as $\hat{p}_l(\vec{x}_{l,r}^+(n))$. Notation $p_l$ is used loosely here, since $p_l$ depends not only on $\vec{x}_{l,r}^+(n)$ but the sum of flow rates $\sum_{s \in R^l} x_{s}^+(n - \tau_{s,r}^l)$.

Further, the implementation of (4.20) below uses the method of Abdulla and Bhatnagar (2006) which improves upon the single-measurement SPSA technique. Due to advantages in convergence, we unbias the measurement $\sum_{l \in L_r} \hat{p}_l(\vec{x}_{l,r}^+(n))$ by subtracting $p_l^1$. It is important that none of the $K_r$ perturbations $\Delta[\tau_r - k], 1 \leq k \leq K_r$ apply to $p_l^1$ and hence a measurement made at index $\tau_r - K_r$ or before is used. Thus we store two past measurements of prices $q_r$, the most recent in variable $p_2^r$, whilst the measurement made $K_r$ indices past is in $p_1^r$.

The algorithm ‘Asynchronous SPSA’ at each source $r$, when $I_{r(\varphi_n = r)} = 1$, is given in pseudocode:

- **Recompute word by storing** $A_r(t_r, K_r) = \{\Delta_r[t_r - 1], \Delta_r[t_r - 2], \ldots, \Delta_r[t_r - K_r]\}$ (for all $k$, $\delta_r[t_r - k]$ can be computed due to its closed form).
- **For any** $1 \leq k_r \leq K_r$, **pick** $\Delta_r[t_r - k_r], w.p. \frac{1}{K_r}$, **let** $\hat{\delta}_r(n) \hat{\delta}_r[t_r - k_r] \Delta_r[t_r - k_r]$.
- **The Hessian diagonal-element** $H_r(n)$ estimates are obtained from an ‘averaging’
4.2 Network Flow-Control using Stochastic Approximation

recursion over the iterates $h_r(n)$:

$$h_r(n + 1) := h_r(n) + \alpha(n, r) \left( \frac{\sum_{l \in L_r} \tilde{p}_l(\bar{x}^+_l(n)) - p^+_l}{\delta_r(n)} - h_r(n) \right), \quad (4.20)$$

$$H_r(n) := \Gamma \left( U'_r(x_r(n)) - h_r(n) \right),$$

with stepsize $\alpha(n, r)$ and projection operator $\Gamma$ explained below.

- The discrete-time implementation mirroring (4.18) is now:

$$x_r(n + 1) := x_r(n) - \frac{\kappa(n, r)}{H_r(n)} \left( U'_r(x_r(n)) - \sum_{l \in L_r} \tilde{p}_l(\bar{x}^+_l(n)) \right). \quad (4.21)$$

- $t_r := t_r + 1$.

- If $t_r \% K_r = 0$

  - $p^+_1 := p^+_2$

  - $p^+_2 := \sum_{l \in L_r} \tilde{p}_l(\bar{x}^+_l(n))$.

- Generate $\Delta_r[t_r]$, and send at rate $x^+_r(n + 1) = x_r(n + 1) + \delta_r(t_r) \Delta_r[t_r]$ until the next update $I_{\varphi_n = r} = 1$.

The stepsize properties are characteristic of two-timescale stochastic approximation:

$$\alpha(n, r) = \alpha[t_r], \quad \sum_k \alpha[k] = \infty, \quad \sum_k \alpha^2[k] < \infty,$$

$$\kappa[k] = o(\alpha[k]) \quad \text{and} \quad \sum_k \left( \frac{\alpha[k]}{\delta_r[k]} \right)^2 < \infty. \quad (4.22)$$

As the selection of $K_r$ may be imprecise, operator $\Gamma$ truncates estimates $h_r(n)$ of (4.20) against $[-M, -m]$ for some $M > m > 0$ so as to avoid values of small magnitude. Note here that (4.20) does not hinge on knowledge of separate $p_l$. 
4.2 Network Flow-Control using Stochastic Approximation

Convergence Analysis

We give a result which mirrors Proposition 1 of Abdulla and Bhatnagar (2006).

**Lemma 10** Consider re-use term \( p_t^l = \sum_{l \in R^t} p_{1,t}^l \) as sum of contributions from links \( l \), then for \( I_{\{r \neq r\}} = 1 \) we have:

\[
E \left( \frac{\tilde{p}_l(\tilde{x}^{t^+}_{l,r}(n)) - p_{1,t}^l}{\delta_r(n)} \right) = \frac{1}{K_r} \nabla_r p_l(\bar{x}_{l,r}(n)) + O(\delta_r^2(n)).
\]

**Proof:** Using a Taylor series expansion of \( p_l(\bar{x}^{t^+}_{l,r}(n)) \) we have:

\[
\frac{\tilde{p}_l(\tilde{x}^{t^+}_{l,r}(n)) - p_{1,t}^l}{\delta_r(n)} = \frac{p_l(\bar{x}_{l,r}(n)) - p_{1,t}^l + \epsilon_t^l(n)}{\delta_r(n)} + \sum_{s \in R^t} \frac{\nabla_s p_l(\bar{x}_{l,r}(n)) \tilde{\delta}_{l,s}(n)}{\delta_r(n)} + O(\delta_r^2(n)),
\]

where \( H_{l} \) corresponds to the Hessian of \( p_l \). With probability \( \frac{1}{K_r} \) our guess of \( \tilde{\delta}_{l,r}(n) \) will result in the second term in RHS of (4.23) becoming:

\[
\nabla_r p_l(\bar{x}_{l,r}(n)) + \sum_{s \neq r, s \in R^t} \frac{\nabla_s p_l(\bar{x}_{l,r}(n)) \tilde{\delta}_{l,s}(n)}{\delta_r(n)}.
\]

The expected value of the second term w.r.t. \( \Delta \) is 0 due to the i.i.d., mean-0 and symmetric nature of the RVs \( \Delta_{l,r}(n) \). Further, the terms involving \( \tilde{\delta}_{l,r}(n) \) in the first and the third terms of the RHS of (4.23) are also mean-0 when the guess \( \tilde{\delta}_{l,r}(n) \) is wrong. The claim follows. \( \square \)

This estimate, however, corresponds to a delayed measurement and proving convergence would further hinge on the ASA framework of Borkar (1998) as applied in \( \text{(4.23)} \). We show that using \( \sum_{l \in L^r} \tilde{p}_l(\tilde{x}^{t^+}_{l,r}(n)) \) above in place of \( \sum_{l \in L^r} \tilde{p}_l(\tilde{x}_{l,r}(n)) \) in (4.18) does not impact the convergence of the algorithm. Again, a Taylor series expansion of \( \tilde{p}_l(\tilde{x}^{t^+}_{l,r}(n)) \) results in \( \tilde{p}_l(\tilde{x}_{l,r}(n)) + O(\delta_r(n - \tau_{l,r})) \). The latter bias term arises from \( \frac{1}{2} \tilde{\delta}_{l,r}(n) H_l(\bar{x}_{l,r}(n)) \tilde{\delta}_{l,r}(n) \) and cancels due to square-summability of the ratio of \( \kappa[t_r] \) and \( \delta_r[t_r] \) from (4.22).
4.2 Network Flow-Control using Stochastic Approximation

4.2.5 Rectification of misbehaving flows

Recently, Fan et al. (2005) considered a misbehaving source of announced-utility \( U_r(x_r) \) surreptitiously using \( \hat{U}_r(x_r) \) for rate updates. An edge-router \( \hat{r} \) was proposed to police \( r \) subject to certain information constraints. This \( \hat{r} \) knows only current source flow-rate \( x_r \), declared utility \( U_r \) and price \( q_r(n) \) and uses these to produce a reference rate \( \hat{x}_r \). Lacking direct control over \( r \)'s sending rate, however, \( \hat{r} \) computes \( \hat{x}_r \) to demand a price \( \hat{q}_r(n) \) from \( r \), in place of \( q_r(n) \) that \( r \) had assumed would suffice. The convergence (cf. Fan et al. (2005)) is to the (approximate) proportionally-fair equilibrium \( x_r^* \) and not some optima desired by the malicious user. It may even be possible for \( r \) to destabilize the network, by using a \( \hat{U}_r \) that thwarts the convergence of (4.16) - in the nature of denial-of-service attacks.

To illustrate the proposed algorithm, we consider the following synchronous recursions. For the present, take \( \varphi_n \) to be set-valued in \( R \cup L \) and assume that \( r \) and \( \hat{r} \) perform updates at each \( n \), i.e., \( I_{\{r \in \varphi_n\}} = I_{\{\hat{r} \in \varphi_n\}} = 1 \). Say \( t_r \) and \( t_{\hat{r}} \) are the local indices \( t_r, t_{\hat{r}} \leq n \). We propose a method according to which edge-router \( \hat{r} \) can update its reference estimate. Suppose there is an \( n \) s.t. \( I_{\{\varphi_n = r\}} = 1 \) and the local index is \( t_{\hat{r}} \). Then, \( \hat{x}_r(n) \hat{\Delta} x_r[t_{\hat{r}}] \) is available from the last update whereas the rate at which \( r \) is sending currently is \( x_r(n) \). The treatment of Fan et al. (2005) does assume that edge-router \( \hat{r} \) has knowledge of these \( x_r(n) \) for all sources \( r \) in its charge.

The edge-router \( \hat{r} \) takes the view that source \( r \) has changed its transmission rate from \( \hat{x}_r(n) \) to \( x_r(n) \) in a single update, and that \( r \) also has the same update frequency as itself. Thus, \( \hat{r} \) assumes that at \( r \) the local index \( n_r \) is \( n_r + 1 \). It assumes that the following recursion has taken place at source \( r \):

\[
x_r(n) := \hat{x}_r(n) + \kappa(n, \hat{r})(\hat{U}_r'(\hat{x}_r(n)) - \hat{q}_r(n)),
\]

where \( \hat{q}_r(n) \) is the modified cost payable by \( r \) computed (and stored) by \( \hat{r} \) at local index
4.2 Network Flow-Control using Stochastic Approximation

\[ \hat{x}_r(n) := \hat{x}_r(n) + \kappa(n, \hat{r})(U'_r(\hat{x}_r(n)) - \hat{q}_r(n)). \] (4.25)

Thus, an approximation to \( \hat{U}'_r(\hat{x}_r(n)) - U'_r(\hat{x}_r(n)) \) would be
\[ \delta q_r(n+1) := \frac{x_r(n) - \hat{x}_r(n)}{\kappa(n, \hat{r})}, \]
\[ \tilde{q}_r(n+1) := q_r(n) + \delta q_r(n+1), \]
\[ \hat{x}_r(n+1) := x_r(n). \]

Until next update of \( \hat{r} \), say at \( n' > n \) s.t. \( I_{(\hat{r}' = \hat{r})} = 1 \), the price charged to source \( r \) is
\( \tilde{q}_r(n+k) = q_r(n+k) + \delta q_r(n+k) \) for \( n < n + k < n' \) where \( \delta q_r(n+k) = \delta q_r(n+1) \). The value \( \hat{x}_r(n+1) \) is stored at \( \hat{r} \) for the computation at \( n' \).

Thus the price demanded of source \( r \) in epoch \( n+1 \) is
\( \tilde{q}_r(n+1) := q_r(n+1) + \delta q_r(n+1), \)
composed of the cost for the present allocation and the gain made from misrepresenting utility in the last epoch. There is no evolution of reference rate \( \hat{x}_r(n) \) here unlike Fan et al. (2005). But note that even though \( \hat{x}_r(n) \) is not explicitly a stochastic approximation iterate, the reinforcement-term of the recursion (4.25) above uses \( x_r(n) \) which are guided by link-costs \( \tilde{q}_r(k) \), dependant in turn on past iterates \( \tilde{x}_r(k), k < n \).

We justify why asynchronous updates of \( r \) and \( \hat{r} \) must be considered. It has been suggested by Fan et al. (2005) that this possibility is minimal, and that delays can be disregarded, owing to source and edge-router communication not being subject to any serious propagation delays. Note that, even with such a synchronous update assumption, the system in (4.25) does not require a two-timescale interpretation unlike Fan et al. (2005). Consider, however, the possibility that an edge-router - usually an ISP - will have to sample the rates of a large number of sources at each update-slot \( n \). In contrast, the ASA framework requires the edge-router to adjust the costs \( \tilde{x}_r \) only a minimal number of times. It suffices for convergence if sampling of all sources occurs with non-zero relative frequencies, a subsampling dependant on the number of sources. The \( \tilde{x}_r(n) \) computation


4.2 Network Flow-Control using Stochastic Approximation

4.2.6 Dual and Primal-Dual Algorithms

We explain the continuous-time Kelly dual algorithm briefly. The ‘static’ source $r$ infers its sending rate from the price signal $q_r(t)$ as $x_r(t) = \left(U'_r(x_r(t)) - \tilde{q}_r(t)\right)$. Note that the operator $(y_l[t^l] - c_l[t^l])_{p_l}^+$ below signifies truncation to 0 if both $y_l[t^l] - c_l[t^l] < 0$ and $p_l[t^l] \leq 0$ - done in order to prevent negative prices (cf. Fan et al. (2005)). At link $l$, a first-order dynamic price update follows the ODE:

$$\dot{p}_l(t) = \alpha_l(y_l(t) - c_l(p_l(t)))_{p_l}^+, \forall \ l \in \mathcal{L},$$

where $\alpha_l > 0$ is a scale factor, $y_l(t)$ is the flow $\sum_{s \in \mathcal{R}_l} x_s(t)$ through $l$, while $c_l(p_l(t))$ is the flow at $l$ for which $p_l(t)$ is payable. The corresponding ASA algorithm again uses an ITS $\alpha[t^l]$

$$p_l[t^l + 1] := p_l[t^l] + \alpha[t^l](y_l[t^l] - c_l[t^l])_{p_l}^+, \forall \ l \in \mathcal{L},$$

where measurement $y_l[t^l]$ is $\sum_{s \in \mathcal{R}_l} x_s[t^l - \psi_s^l]$. Note that, unlike the penalty-function based primal algorithm, it is the dual algorithm that can perform exact capacity utilization with the terms $p_l$ being termed the ‘shadow prices’ (cf. Low and Srikant (2004)).

The proof techniques of §4.2.3 hold. The Lipschitz continuity of the term $y_l[t^l] - c_l[t^l]$ which corresponds to $\sum_{s \in \mathcal{L}_s} U_s'^{-1}(\sum_{l \in \mathcal{L}_s} p_l[t_r - \xi_s^l]) - c_l[t^l]$ must be satisfied. Thus, we assume $U_s'^{-1}(\cdot)$ and $c_l(\cdot)$ are Lipschitz. The unique, globally stable equilibrium $p^*_l$ (which is the equilibrium price at link $l$) is reached as $t \to \infty$ and has the property $q^*_r = \sum_{l \in \mathcal{L}_r} p^*_l = U'_r(x^*_r)$. The misrepresentation of $U_r$ by misbehaved sources $r$ that we considered previously in §4.2.5 can also occur in this scenario.

Next we propose an ASA algorithm adapting the primal-dual algorithm. In particular, this version of the primal-dual algorithm achieves full utilization by having slower timescale link dynamics over and above the primal algorithm. The inequality constraints of the network-utility maximization problem \((4.14)\) can be folded-in as follows:

\[
\min_{p_l > 0} \max_{x_r \geq 0} \sum_{r=1}^{|R|} U_r(x_r) - \sum_{l=1}^{|L|} p_l \cdot \left\{ \sum_{s \in L_s} x_s - c_l \right\},
\]

where the latter term now involves a multiplication with Lagrange multipliers \(p_l\) (not the functions \(p_l(\cdot)\) used previously). The primal-dual algorithm is capable of ensuring both efficient capacity utilization and fair allocation (cf. \(\text{Low and Srikant (2004)}\)).

Our algorithm resembles the algorithm of \(\text{Borkar (2005)}\) except that the latter is a three time-scale algorithm whilst we propose two timescales. The slower timescale recursion performed at the links \(l\) with stepsizes \(\alpha[t_l]\) guide the Lagrange-multiplier \(p_l\) computation, whilst the faster recursion at sources \(r\) with stepsizes \(\kappa_r[t_r]\) ‘converges’ to a rate-allocation for each \(p_l[t_l]\) iterate:

\[
x_r[t_r + 1] = x_r[t_r] + \kappa_r[t_r] (U_r(x_r[t]) - q_r[t_r])
\]

\[
q_r[t] = \sum_{l \in L_r} p_l[t] - \xi_r + \epsilon_r[t]
\]

\[
p_l[t_l + 1] = p_l[t_l] + \alpha[t_l](y_l[t_l] - c_l)^+
\]

The conditions on the stepsizes, \(\forall r \in R\) and \(l \in L\), are: \(\kappa[t_r], \alpha[t_l] > 0\),

\[
\sum_{t_r} \kappa[t_r] = \sum_{t_l} \alpha[t_l] = \infty,
\]

\[
\sum_{t_r} (\kappa[t_r])^2, \sum_{t_l} (\alpha[t_l])^2 < \infty,
\]

and, the characteristic two-timescale condition:

\[
\alpha[t_l] = o(\kappa[t_l]), \forall l \in L_r.
\]
Note, however, that the onus here is on proving convergence of a two-timescale asynchronous stochastic approximation recursion and therefore requires added conditions on the delays. As pointed out earlier, asynchronous two-timescale stochastic approximation has been handled by Konda and Borkar (1999) in the particular setting of simulation-based algorithms for Markov Decision Processes.

### 4.3 Simulation Results

#### 4.3.1 SPSA Numerical Example

We first compare algorithm SPSA2-2R of Spall (1992) with the proposed SPSA2-1UR using the setting of Spall (1997). In particular, the objective function used is

\[
L_b(\theta) = b + \theta^T \theta + 0.1 \sum_{i=1}^{5} \theta_i^3 + 0.01 \sum_{i=1}^{5} \theta_i^4,
\]

with \( \theta^* = 0 \) and \( L_b(\theta^*) = 0 \) for all \( b \). We keep \( b = 0 \) for comparison with SPSA2-2R and change to \( b = 0.1 \) for comparison with SPSA2-1R. We use \( a = c = 1, \alpha = 6\gamma = 1 \) and \( \theta_0 = 0.11 \) (i.e., the vector with 0.1 in all its components) in all the experiments. Assume that \( \epsilon_k \) are i.i.d., mean-zero, Gaussian random variables with variance \( \sigma_\epsilon^2 \). The formula for asymptotic normality derived previously lets us consider two cases for the observation noise:

1. \( \sigma_\epsilon = 0.1 \) where \( \frac{\bar{a}_1}{n_2} \to 1.30 \), and
2. \( \sigma_\epsilon = 0.07 \) where \( \frac{\bar{a}_4}{n_2} \to 0.93 \), respectively.

Each run of the SPSA2-2R algorithm is for 2000 iterations, thus making 4000 observations of the objective function. Table 4.1 summarizes the results, the mean square error (MSE) obtained being over 100 runs of each algorithm. The MSE values for SPSA2-2R are less when compared to SPSA2-1UR, the proportion being 0.93 and 0.92, respectively for the two cases. However, this ratio improves if we use the SPSA2-1UH algorithm, which we compare with the analogous SPSA2-2H algorithm in Table 4.2.
4.3 Simulation Results

Table 4.1: 2R Vs. 1UR: MSE and No. of Iterations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\sigma_e = 0.1$</th>
<th>$\sigma_e = 0.07$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>Iter.</td>
<td>MSE</td>
</tr>
<tr>
<td>SPSA2-2R</td>
<td>0.0135</td>
<td>2000</td>
</tr>
<tr>
<td>SPSA2-1UR</td>
<td>0.0145</td>
<td>5200</td>
</tr>
</tbody>
</table>

While we have no asymptotic normality results for SPSA2-1UH, the performance obtained is better than that of SPSA2-1UR. We also observe the performance of SPSA2-1UR vis-a-vis SPSA2-1R in Table 4.3. Possibly due to the larger number of iterations required to achieve asymptotic normality, the MSE is always higher. A notable change in the behaviour of SPSA2-1R is the higher MSE when $b = 0.1$. This is due to the $L^2(\theta^*)$ bias term in (4.8). Note that we use $\sigma_e = 0.1$ in both the above comparisons.

We compare the second-order algorithms on the same setting. For algorithms 2SPSA-3UR and 2SPSA-2UR, we use $\hat{\Delta}_{k,i} \in \{+1, -1\}$ whereas for 2SPSA-1UR we used $\hat{\Delta}_{k,i} \in \{+0.5, -0.5\}$. The step-size $\tilde{c}_k$ was the same as $c_k$, whilst $b_k = \frac{1}{\text{max}}$. We used a similar projection operator $\Gamma(\cdot)$ as in the experiments of Bhatnagar (2005), i.e., choose the diagonal elements $\tilde{H}_k(i,i), 1 \leq i \leq p$ of the Hessian estimate and truncate to interval [0, 10.0]. An upper bound of 10 on $H_k(i,i)$ was justified since typically two-timescale algorithms are known to perform better with an additional averaging on the faster timescale, where $L >> 1$ measurements are made. Since recourse to multiple measurements is ruled out in this setting, we chose to prune the fluctuations in these diagonal terms $\tilde{H}_k(i,i)$.

We compare 2SPSA-3UR with the four-measurement 2SPSA of Spall (2000) to obtain the results in Table 4.4. We run both algorithms in such a manner that the number of function evaluations is the same, viz. 4000. The convergence of the bias (of $\hat{H}_k$) in 2SPSA-3UR is $O(c_k)$, resulting in problems establishing any asymptotic normality results. As a consequence, there is no clear set of parameters for which 2SPSA-3UR would outperform 2SPSA. This slower order of convergence may also be responsible for the poor performance of the algorithm. There is an occasional disconnect between finite-time performance of the second-order algorithms vis-a-vis the robust convergence behaviour expected from a Newton-Raphson analog. We chose this numerical setting to compare the proposed algorithms with those in the literature. The work of Zhu and Spall (2002) explores both
finite-time performance and a computationally-efficient second-order SPSA algorithm. The difference with Zhu and Spall (2002) would lie in choosing the $\Gamma$ operator of (4.12). This is an issue identified by Bhatnagar (2005), from where we chose the 3SA, 2SA and 1SA algorithms for modification. Table 4.5 compares performance of 2SPSA-2UR w.r.t. 2SA of Bhatnagar (2005). The algorithms are more or less on par with each other. Table 4.6 compares the proposed 2SPSA-1UR algorithm with 1SA of Bhatnagar (2005), the former being seen to perform better.

Table 4.2: 2H Vs. 1UH: MSE and No. of Iterations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\sigma_e = 0.1$</th>
<th>$\sigma_e = 0.07$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPSA2-2H</td>
<td>0.0133</td>
<td>0.0127</td>
</tr>
<tr>
<td>SPSA2-1UH</td>
<td>0.0109</td>
<td>0.0109</td>
</tr>
</tbody>
</table>

Table 4.3: 1R Vs. 1UR: MSE and No. of Iterations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$b = 0$</th>
<th>$b = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPSA2-1R</td>
<td>0.0443</td>
<td>0.0492</td>
</tr>
<tr>
<td>SPSA2-1UR</td>
<td>0.0132</td>
<td>0.0147</td>
</tr>
</tbody>
</table>

Table 4.4: Comparison of 2SPSA-3UR

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\sigma_e = 0.1$</th>
<th>$\sigma_e = 0.07$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2SPSA</td>
<td>0.037</td>
<td>0.039</td>
</tr>
<tr>
<td>2SPSA-3UR</td>
<td>0.078</td>
<td>0.073</td>
</tr>
</tbody>
</table>

4.3.2 Network Flow Control Experiments

We conduct two sets of experiments with the proposed algorithms. First, we consider a simple setting of a single link into which four sources feed traffic. These sources $r = \{1, 2, 3, 4\}$ send Poisson streams of packets at rates $x_r$. The value $y_1 = \sum_{r=1}^{4} x_r$ is estimated at the end of every slot of 5.0 seconds by the link by dividing the number of packets arrived in the slot with 5.0. This aggregate measurement $y_1$ is then intimated.
4.3 Simulation Results

Table 4.5: Comparison of 2SPSA-2UR

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\sigma_r = 0.1$</th>
<th>MSE</th>
<th>Iter.</th>
<th>$\sigma_r = 0.07$</th>
<th>MSE</th>
<th>Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2SA</td>
<td>0.076</td>
<td>2000</td>
<td></td>
<td>0.077</td>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2SPSA-2UR</td>
<td>0.072</td>
<td>2000</td>
<td></td>
<td>0.078</td>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6: Comparison of 2SPSA-1UR

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\sigma_r = 0.1$</th>
<th>MSE</th>
<th>Iter.</th>
<th>$\sigma_r = 0.07$</th>
<th>MSE</th>
<th>Iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1SA</td>
<td>0.854</td>
<td>4000</td>
<td></td>
<td>0.851</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>2SPSA-1UR</td>
<td>0.655</td>
<td>4000</td>
<td></td>
<td>0.630</td>
<td>4000</td>
<td></td>
</tr>
</tbody>
</table>

to all sources $r$ via the cost-function $p_1(y_1) = y_1^{0.8}$. The sources, however, update only at intervals of 1, 2, 3 and 6 slots, respectively. Therefore, the global index $n$ now corresponds to the most frequent of all updates/measurements, viz. that of link 1 and source 1.

We choose the utility function $U_r(x) = w_r \ln x_r$ (from Kelly et al. (1998)) where $w_r = 1, 2, 3,$ and 6, respectively. First, fixed delays were considered where both feedforward $\psi_1^r(n)$ and feedback delays $\xi_1^r(n)$ were set to 1, 2, 2 and 4, respectively, ensuring that no source $r$ performs an update using the most recent measurements. Next, variable delays were considered as $\hat{\psi}_1^r(n) = \psi_1^r(n) - P_1^r(n)$ where the random variable $P_1^r(n) \in \{0, 1\}$ with probability 0.5 each. Also, feedback delays $\hat{\xi}_1^r(n)$ are defined analogously. This delay structure also helps in the experiment with second-order enhancement of §4.2.4 since a choice of $K_r = 1, \forall r$ is valid. We also truncate against $[1.0, 10.0]$ the iterates $h_r(n)$ of (4.20). Further, to experiment with the rectification algorithm of §4.2.5 we took $\hat{w}_1 = 2.0$, where source 1 has misrepresented its utility $\hat{U}_1(x_1) = \hat{w}_1 \ln x_1$. The edge-router that computes the modified price $\hat{q}_1(n)$ does so every 5 slots, as compared to source 1 which updates rate $x_1(n)$ in every slot $n$.

Our findings are in Table 4.7. The algorithms were terminated at an $\hat{n}$ where

$$\max_{k \in \{1, 2, \ldots, 100\}} \|x(\hat{n} - k) - x(\hat{n})\|_1 \leq 10^{-3}.$$  

We used the stepsizes, $\kappa[t_r] = t_r^{-1}$ and $\alpha[t_r] = t_r^{-0.55}$ (applicable to the second-order
4.3 Simulation Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$E(\hat{n})$</th>
<th>$\sigma_{\hat{n}}$</th>
<th>$|\cdot|_1$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Delay</td>
<td>7365</td>
<td>580</td>
<td>0.070</td>
</tr>
<tr>
<td>Noisy Delay</td>
<td>7234</td>
<td>726</td>
<td>0.072</td>
</tr>
<tr>
<td>Second-Order</td>
<td>4020</td>
<td>385</td>
<td>0.073</td>
</tr>
<tr>
<td>Rectification</td>
<td>7568</td>
<td>625</td>
<td>0.071</td>
</tr>
</tbody>
</table>

Table 4.7: Performance of Primal Algorithm and variants

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$E(\hat{n})$</th>
<th>$\sigma_{\hat{n}}$</th>
<th>$|\cdot|_1$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual</td>
<td>4780</td>
<td>506</td>
<td>0.101</td>
</tr>
<tr>
<td>Primal-Dual</td>
<td>961</td>
<td>153</td>
<td>0.102</td>
</tr>
</tbody>
</table>

Table 4.8: Performance of Dual and Primal-Dual Algorithms

algorithms). The optimum vector $x^*$ is $(0.33, 0.66, 0.99, 1.98)^T$, and the error $\|x(\hat{n}) - x^*\|_1$ w.r.t. $x^*$ is also recorded. The term $\sigma_{\hat{n}}$ denotes the standard deviation of $\hat{n}$ from its mean $E(\hat{n})$ over 100 runs of the algorithm.

We verified the dual and primal-dual algorithms proposed in §4.2.6. The dual algorithm divided the bandwidth $C_1 = 5.0$ packets per second of the single link among the sources $r$. While the utility functions $U_r$ were the same as above, note that the price charged by the link is always linear in the aggregate flow $y_1$. We chose stepsize $\alpha^l[t] = (t)^{-1}$. In the primal-dual algorithm, the same task is accomplished faster. For any system, once the regime $\alpha^l[k] = o(\kappa_r[k])$ is in place, the primal and dual algorithms at sources and links, respectively, can execute in tandem. A source $r$, for example, is unaffected by (and is also unaware of) an algorithm that changes the price-function at some link $l \in L_r$.

In the second set of experiments we chose the unstable system in §IX.A.2 of Ranjan et al. (2006). In this system, three sources $r \in R \equiv \{1, 2, 3\}$ use two links $l_1$ and $l_2$ such that $R^1 = \{1, 3\}$ and $R^2 = \{2, 3\}$. The utility functions are $U_r(x_r) = \frac{1}{a_r x_r}$ (resulting in $U'_r(x_r) = x_r^{-(a_r+1)}$) with $a_1, a_2 = 3$ and $a_3 = 4$. Links charge using the function $p_l(y) = \left(\frac{y}{C_l}\right)^{b_l}$, $b_l = 3.5$ for $l = 1, 2$ where $C_l$ are the link capacities pegged at 5 and 4, respectively. Using the analysis of Ranjan et al. (2006), feedback delays $(\xi_1^1, \xi_1^3)^T = (280, 770)^T$ and $(\xi_2^1, \xi_3^3)^T = (430, 770)^T$ seconds (all feedforward delays $\psi^l$ are 0) result in a delayed differential equation (DDE) that does not possess a globally stable
4.3 Simulation Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$E(\hat{n})$</th>
<th>$\sigma_{\hat{n}}$</th>
<th>$E(x(\hat{n}))$</th>
<th>$| \cdot |_1$-error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>-</td>
<td>-</td>
<td>(1.65, 1.44, 1.22)</td>
<td>0</td>
</tr>
<tr>
<td>Delay</td>
<td>91762</td>
<td>15774</td>
<td>(1.63, 1.43, 1.22)</td>
<td>0.040</td>
</tr>
</tbody>
</table>

Table 4.9: Comparison with system analysed via DDEs

equilibrium point. For smaller delays or a different set of parameters $a$ and $b$, this system converges, as evidenced in §IX.A.1 of Ranjan et al. (2006).

When implemented using the ASA framework of §4.2.3 convergence is assured irrespective of delays, since it is the ODE with a globally stable equilibrium point that the ASA recursion tracks. The global index $n$ represents a slot of 1 second. Thus, source 3, for example updates at \{ $n : n \% 770 = 0$ \}. Since the updates are spaced farther apart, the convergence conditions are stricter, with $\hat{n} = \max_{k \in \{1,2,\ldots,5000\}} \| x(\hat{n} - k) - x(\hat{n}) \|_1 \leq 10^{-3}$.

As regards the experiments we note that discretization per se, for implementing the rate-controller, is not uncommon in the literature, e.g. delay-differential equations are converted to delayed difference equations for simulation via MATLAB by Liu et al. (2005) and Ying et al. (2006). The work of Ranjan et al. (2006) and La and Ranjan (2006) has used the fluid model directly assuming that the flow is the average behaviour of a large number of sub-flows. In comparison, we have both discretized the dynamics as well as considered noise and delay effects in the first set of experiments.
Chapter 5

Conclusions and Future Directions

Abstract

We summarize with conclusions on the material of the preceding chapters and some interesting and immediate future directions to investigate.

5.1 Infinite Horizon MDPs

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

We next proposed a three-timescale simulation-based algorithm with similar coupled stochastic approximation to implement the value iteration algorithm of DP (cf. Bertsekas, 1995, Section 8.2) with the discounted cost criteria. A novel use of the perturbed cost-to-go iterates to act as a proxy for the actual cost-to-go was seen in this latter algorithm. Another advantage of the algorithm was that termination criteria could be specified both in terms of policy and cost-to-go iterates, thereby accommodating problem-specific sensitivity to either. We noted the similarity of the slowest timescale recursion in this algorithm to the small-step version of value iteration in (Bertsekas and Tsitsiklis, 1996, Section 5.5). The future directions we propose here are also of relevance to finite-horizon MDPs:

- Whether convergence is asymptotic or not, it is important to assure the least amount of simulation using sub-optimal actions during the course of computing the optimal policy for an MDP. Thus the possibility of showing ‘logarithmic-regret’, taken from the stochastic multi-armed bandit literature (cf. Auer et al. (2002)), in simulation-based solutions is very attractive. The term ‘logarithmic regret’ arises from the exploration-exploitation dilemma and assures that an algorithm makes only $O(\log n)$ selections of sub-optimal actions among $n$ trials, and has been established as the lower-bound. It would be ideal to extend this to the MDP framework, as has already been done for the trajectory-based, finite sampling-budget, Finite Horizon MDP case in Chang et al. (2005).

Showing ‘convergence’ of an algorithm via logarithmic regret framework is another novelty in that it differs from the current proof techniques of stochastic approximation, although the latter are more general. The obstacle, however, lies in the $N$—sample $Q$—value iterate for action $a \in C_i$ which contains temporary estimates $V_n$, $1 \leq n \leq N$ of the next state’s cost-to-go. In that, we use samples $K(i, a, \eta_n(i, a)) + \alpha V_n(\eta_n(i, a))$ in the mean instead of samples $K(i, a, \eta_n(i, a)) + \alpha V^*(\eta_n(i, a))$. Such a difficulty is natural since $V^*(\eta_n(i, a))$ is unknown to any simulation-based MDP algorithm. Using randomized stationary policies (RSPs) for
the trivial case of a finite-action set 1–state MDP, which is in fact a multi-armed stochastic bandit, has still not yielded true logarithmic regret - the closest achieved is $O((\log n)^2)$ in Cesa-Bianchi and Fisher (1998).

- The work in Abdulla and Bhatnagar (2007b) handled both finite and infinite-horizon MDPs with the key advantage that RSP iterates $\pi_n(i)$ stayed within the probability simplex even upon updates. Note that, due to its execution being an optimization problem, a large amount of computation can be saved by avoiding the projection operator $P_i$ of $\S{2.1.4}$ altogether. However, further analysis on the results in Abdulla and Bhatnagar (2007b) is warranted and this prevents us from including the work in the thesis.

- The fast convergence of the proposed algorithms in this chapter could be theoretically established using rate-of-convergence studies of SPSA-based gradient estimates vis-a-vis actor-critic algorithms that use other policy gradient methods as in Konda and Borkar (1999). In this context, further improvement in performance of the proposed algorithms can be expected by using efficient simulation-based higher order SPSA algorithms that estimate the Hessian in addition to the gradient along the lines of Bhatnagar (2005).

- Also, in (2.31) a strict lexicographic sampling is not necessary. Moreover, the order of states need not be stored by the (distributed) critic, that would result in a promising implementation more in line with the bounded rationality assumption of agents performing policy updates in an asynchronous manner.

- Any combination (or generalization) of the following three tasks in a dynamical system viz. a) solving a system of equations, b) solving a minimization problem, and c) expectation evaluation, are candidates for simulation-based implementation with appropriately chosen stepsizes via the multi-timescale stochastic approximation framework. We are optimistic that particular applications of such a technique would abound.
5.2 Finite Horizon MDPs

We proposed algorithms for finite-horizon MDPs (FH-MDPs) which were simulation based. We first developed look-up table actor-critic reinforcement learning based algorithms under two different settings: (a) finite state and compact action sets and (b) finite state and discrete action sets. We presented a convergence analysis for our algorithms and showed results of numerical experiments on a setting of flow control in communication networks. All our algorithms used a variant of SPSA gradient estimate where the perturbation sequences were derived from normalized Hadamard matrices and show fast convergence.

In the later part, we proposed two parameterized actor-critic algorithms for FH-MDPs. The parameterization requires constant storage: viz. \((2K + K^2) \times T\), where \(K \ll |S|\), although the more efficient second algorithm needs additional \(2|S|\) storage. This contrasted with §3.1 where at least \(2|S| \times T\) is needed, even assuming there are only two feasible actions per state. On the other hand, the proposed algorithms have time requirements \(O(0.5T^2)\) and \(O(3T)\) compared to \(O(2T)\) or \(O(T)\) in §3.1. An extension of the algorithm to the stopping time problem was discussed in §3.2.4, and we identified why current policy evaluation schemes do not produce critic estimates suitable for an SPSA policy-gradient method. Two avenues of future research, the latter relevant to Chapter 2 also, are immediate:

- A solution to the high simulation load of Algorithm 1 of §3.2.2 lies in using one-simulation SPSA estimates, a method adopted in Bhatnagar and Abdulla (2006). Also, algorithms in Abdulla and Bhatnagar (2006) aim to reduce the variance encountered in the one-simulation SPSA gradient estimate by re-using past estimates of \(V^+(n)\). 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 along the lines of Abdulla and Bhatnagar (2006), Bhatnagar (2005).

- The flow control example in Chapters 2 and 3 assumes lack of rate-control at the
senders, whereas the primal algorithm of Chapter 4 assumes an update mechanism at the sources. At the single bottleneck link, the rate allocation for a source $r$ is $x_r = \sum_{i=1}^{[S]} q_{\pi^*}(i) \lambda_r^*(i)$ where $\lambda = (\lambda_r, 1 \leq r \leq |R|)^T$ is the tuned policy and $q_{\pi^*}(i)$ is the stationary probability under policy $\pi^*$. Due to the moving target state, there will be separate rates $x_{r,l}$ for each stage $l$ in the FH-MDP case. If utilities of the sources are also known to the link, an algorithm that performs both cost-minimization and utility maximization - via two timescale stochastic approximation - is conceivable. Network-wide entities like ISPs can be assumed to have access to senders’ utility functions, yet such an algorithm will not be of a distributed nature.

5.3 Stochastic Approximation Applications

We proposed a bias-reducing one-measurement gradient search modifying the one-measurement SPSA and showed improved convergence properties via the covariance matrix expression. We further modified the algorithm to accommodate perturbations drawn from normalized Hadamard matrices, and also proposed second-order algorithms. Next, we considered a generalization of the flow control problem of Chapters 2 and 3 and proposed a robust discrete-time implementation of the Primal flow-control algorithm. We proposed a second-order algorithm (using the above improved one-measurement SPSA) for faster convergence to an equilibrium allocation, and an algorithm for correcting sources that misrepresent their utility. Also proposed were implementations of the ‘Dual’ and ‘Primal-Dual’ algorithm. Two avenues for future research are:

- The asymptotic convergence properties of SPSA2-1H have been theoretically shown to be on par with SPSA2-1R in Proposition 2.5 of Xiong et al. (2002). Yet, it is unclear why SPSA2-1H performs better in practice and this represents an avenue for future investigation. Also of interest is the possibility of reducing the scale factor 2 in the asymptotic covariance matrix $\tilde{M}_1$ using an average of past measurements $L(\theta_{k-j}), j > 1$. Whether online function regression mechanisms will serve as a ‘critic’ to speed up SPSA gradient descent by yielding an approximation of
the objective function remains to be seen. Such an arrangement would place the resulting algorithm in-between the accepted forms of 'gradient-free' and 'gradient-based’ methods. Further, in line with the asymptotic normality results of both first and second order SPSA algorithms, work such as Konda and Tsitsiklis (2004) that identifies rate of convergence of two-timescale recursions would be useful.

- We proposed gradient-ascent along the strict Lyapunov function in §4.2.4 above. The investigation in Ying et al. (2006) (cf. eq. (8) there) proposes a Lyapunov-Razumikhin function $W_r(t)$ to establish stability. Pursuing ascent along $W_r(t)$, due to its simpler form, could also be an avenue for speed-up. Also, the penalty $\delta q_r$ added by the ISP can be perceived as originating from a link. One could investigate whether, with minimal computation, it is possible to extract penalties when a large number of sources use wrong utilities $\hat{U}_r$. 
Appendix A

Derivation of $M_1$

The following is the algebra to show that $M_1 = a^2 e^{-2\rho^2} (\sigma^2 + L^2(\theta^*)) \text{Diag}((2\lambda_l - \beta_+)^{-1})_{i=1}^p$
(cf. (3.5) of Spall (1992) for notation and derivation up to this point):

\[
E(V_k V_k^T | F_k) = k^{-2\gamma} E\left\{ \frac{L(\theta^*_k)}{ck^{-\gamma}} \right\}^{(\Delta_k^{-1}) T} \left( \frac{L(\theta^*_k) + \epsilon_k^+}{ck^{-\gamma}} \right) \left( \frac{L(\theta^*_k) + \epsilon_k^+}{ck^{-\gamma}} \right)^T | F_k \}

= k^{-2\gamma} E\left\{ \Delta_k^{-1} \left( \frac{L(\theta^*_k)}{ck^{-\gamma}} \right)^T L^2(\theta^*_k) \left( \frac{L(\theta^*_k) + \epsilon_k^+}{ck^{-\gamma}} \right)^T | F_k \right\}

- k^{-2\gamma} E\left\{ [g(\theta_k) + b_k(\theta_k)] \left( \frac{L(\theta^*_k)}{ck^{-\gamma}} \right)^T | F_k \right\}

+ k^{-2\gamma} E\left\{ \Delta_k^{-1} \left( \frac{L(\theta^*_k)}{ck^{-\gamma}} \right)^T \frac{\epsilon_k^+}{c^2k^{-2\gamma}} | F_k \right\}

- k^{-2\gamma} E\left\{ [g(\theta_k) + b_k(\theta_k)] \left( \frac{L(\theta^*_k)}{ck^{-\gamma}} \right)^T | F_k \right\}

- k^{-2\gamma} E\left\{ \Delta_k^{-1} [g(\theta_k) + b_k(\theta_k)]^T \frac{L(\theta^*_k)}{ck^{-\gamma}} | F_k \right\}

+ k^{-2\gamma} [g(\theta_k) + b_k(\theta_k)] [g(\theta_k) + b_k(\theta_k)]^T \right.$

(A.1)
Only the first, fourth and eighth terms in (A.1) remain, since the rest cancel as a result of independent, zero mean $\epsilon_k^+$ and $(\Delta_k)^{-1}$. In the expansion (A.1), note that the terms:

$$k^{-2\gamma} E \left\{ \Delta_k^{-1}(\Delta_k^{-1})^T \frac{L^2(\theta_k + \hat{\Delta}_k)}{c^2 k^{-2\gamma}} | F_k \right\}$$

$$k^{-2\gamma} E \left\{ \Delta_k^{-1}(\Delta_k^{-1})^T \frac{\epsilon_k^+}{c^2 k^{-2\gamma}} | F_k \right\}$$

$$k^{-2\gamma} [g(\theta_k) + b_k(\theta)] [g(\theta_k) + b_k(\theta)]^T$$

converge to $c^{-2}\rho^2 L^2(\theta^*)$, $c^{-2}\rho^2 \sigma^2_\epsilon$, and 0, respectively as $k \to \infty$. The mean zero property of $\epsilon_k^+$ is used for the second term. Further, $\Phi = -aI$, resulting in:

$$M_1(i, i) = a^2 c^{-2}\rho^2 (c_\epsilon^2 + L^2(\theta^*))(2\lambda_i - \beta_+)^{-1},$$

and thus validating (4.6).
Appendix B

Derivation of $\tilde{M}_1$

Now to proceed with computation of $\tilde{M}_1$, arranging terms together similar to (A.1), one obtains:

$$E(V_kV_k^T|\mathcal{F}_k) = k^{-2\gamma}E\left\{\frac{L(\theta_k^+) + \epsilon_k^+ - L(\theta_{k-1}^+) - \epsilon_{k-1}^+(\Delta_k)^{-1} - [g(\theta_k) + b_k(\theta_k)]}{ck^{-\gamma}}\right\}$$

$$\cdot \frac{L(\theta_k^+) + \epsilon_k^+ - L(\theta_{k-1}^+) - \epsilon_{k-1}^+(\Delta_k)^{-1} - [g(\theta_k) + b_k(\theta_k)]^T}{ck^{-\gamma}}$$

$$= k^{-2\gamma}E\left\{\Delta_k^{-1}(\Delta_k^{-1})^T \frac{(L(\theta_k^+) - L(\theta_{k-1}^+))^2}{c^2k^{-2\gamma}}|\mathcal{F}_k\right\}$$

$$+ 2k^{-2\gamma}E\left\{\Delta_k^{-1}(\Delta_k^{-1})^T \frac{\epsilon_k^+ - \epsilon_{k-1}^+}{ck^{-\gamma}} \cdot \frac{L(\theta_k^+) - L(\theta_{k-1}^+)}{ck^{-\gamma}}|\mathcal{F}_k\right\}$$

$$+ k^{-2\gamma}E\left\{\Delta_k^{-1}(\Delta_k^{-1})^T \frac{(\epsilon_k^+ - \epsilon_{k-1}^+)^2}{c^2k^{-2\gamma}}|\mathcal{F}_k\right\}$$

$$+ k^{-2\gamma}[g(\theta_k) + b_k(\theta_k)]^T[g(\theta_k) + b_k(\theta_k)]|\mathcal{F}_k\right\}$$

(B.1)

The third term on the RHS of the second equality in (B.1) converges to $2c^{-2}\sigma_i^2\rho^2$. All the other terms converge to 0, giving $\tilde{M}_1$ as claimed. To obtain $\mu$, observe that $b_k(\theta_k)$ is the same as for SPSA-1 of the special case $3\gamma - \frac{\alpha}{2} = 0$ in Proposition 2 of Spall (1992).
Appendix C

Derivation for 2SPSA-3UR

We claim \( \frac{1}{c_k \Delta_{k,j}} \left( g_i(\theta_k^+) - \frac{L(\theta_k^+)-L(\theta_k^-)}{2c_k \Delta_{k,i}} \right) = H_{i,j}(\theta_k) + O(c_k) \) using the knowledge that

\[
g_i(\theta_k^+) = g_i(\theta_k) + c_k \sum_{m=1}^{p} H_{i,m}(\theta_k) \Delta_{k,m} + O(c_k^2)
\]

and

\[
\frac{L(\theta_k^+)-L(\theta_k^-)}{2c_k \Delta_{k,i}} = g_i(\theta_k) + \sum_{m=1,m \neq i}^{p} g_m(\theta_k) \frac{\Delta_{k,m}}{\Delta_{k,i}} + O(c_k^2)
\]

substituting both the above into the original expression, we have

\[
H_{i,j}(\theta_k) + \sum_{m \neq j} H_{i,m}(\theta_k) \frac{\Delta_{k,m}}{\Delta_{k,j}} - \frac{1}{c_k} \sum_{m=1,m \neq i}^{p} g_m(\theta_k) \frac{\Delta_{k,m}}{\Delta_{k,i} \Delta_{k,j}} + O(c_k),
\]

where the expectation of the second and third terms are zero.
Appendix D

Diagonal Correction in 2SPSA-2UR

After the correction of (4.13), we claim

\[ \frac{1}{c_k \Delta_{k,j}} \left( g_i(\theta_k^+) - \frac{L(\theta_k^+,\theta_{k-1}) - e_{k-1}^i}{c_k \Delta_{k,i}} \right) = H_{i,j}(\theta_k) + O(c_k). \]

We use the expression for \( g_i(\theta_k^+) \) from Appendix C and defining \( \tilde{L}(\theta_{k-1}^+) = L(\theta_{k-1}^+) + e_{k-1}^i \), observe that

\[
\frac{L(\theta_k^+) - \tilde{L}(\theta_{k-1}^+)}{c_k \Delta_{k,i}} = \frac{L(\theta_k) - \tilde{L}(\theta_{k-1}^+)}{c_k \Delta_{k,i}} + g_i(\theta_k) + \sum_{m=1, m \neq i}^{p} g_m(\theta_k) \frac{\Delta_{m,k}}{\Delta_{k,i}}
\]

\[
+ \frac{c_k \Delta_{k,j}^T H(\theta_k) \Delta_{k,j}}{2 \Delta_{k,i}} + O(c_k^2).
\]

Substituting which, we have

\[
= - \frac{L(\theta_k) - \tilde{L}(\theta_{k-1}^+)}{c_k \Delta_{k,i} \Delta_{k,j}} + H_{i,j}(\theta_k) + \sum_{m \neq j} H_{i,m}(\theta_k) \frac{\Delta_{k,m}}{\Delta_{k,j}}
\]

\[
- \frac{1}{c_k} \sum_{m=1, m \neq i}^{p} g_m(\theta_k) \frac{\Delta_{m,k}}{\Delta_{k,i} \Delta_{k,j}} - \frac{\Delta_{k,j}^T H(\theta_k) \Delta_{k,j}}{2 \Delta_{k,i} \Delta_{k,j}} + O(c_k).
\]

The first and the second-last terms above are crucial when \( i = j \), whilst all other terms have mean 0.

Assume that \( \Delta_{k,i} \) are Bernoulli RVs and \( \Delta_{k,j}^2 = 1 \), resulting in \( E(\Delta_{k,j}^T H(\theta_k) \Delta_{k}|F_k) = \sum_{i=1}^{p} H_{i,i}(\theta_k) \). Due to the correction added in (4.13), \( E(-\frac{L(\theta_k^+,\theta_{k-1})}{c_k} + \frac{L(\theta_{k-1}^+,\theta_{k-1})}{c_k} | F_k) \) reduces to \( E(\frac{L(\theta_{k}^+,\theta_{k})}{c_k} | F_k) \). This in turn equals \( \frac{1}{2} E(\Delta_{k,j}^T H(\theta_k) \Delta_{k}) + O(c_k) \), canceling the second-last term in the RHS above.
Bibliography


S. Bhatnagar, M. C. Fu, S. I. Marcus, and I-J. Wang. Two–timescale simultaneous


J. C. Spall. personal communication. dated 5 October, 2005.


