Working Paper

IIMK/WPS/293/ITS/2018/37

JUNE 2018

LSPI-CAS: Least-Squares Policy Iteration for Compact Action Set MDPs

Mohammed Shahid Abdulla [1]
Shalabh Bhatnagar [2]

[1]Associate Professor, Information Technology and Systems at the Indian Institute of Management, Kozhikode, Kozhikode, India. IIMK Campus P.O., Kozhikode, Kerala 673570, India; Email: shahid@iimk.ac.in
Phone Number (+91) 495 - 2809254

[2] Professor and Chair, Dept of Computer Science and Automation at the Indian Institute of Science, Bangalore, Email: shalabh@iisc.ac.in , Phone: 91(80) 2293-2987

Abstract

Simulation-based, model-free solutions to Markov Decision Processes (MDPs) using the algorithm Least Squares Policy Iteration (LSPI) have been applied to multiple practical settings and in several variants. An optimal policy in an MDP is that policy, or a description of which action to take in a state of the MDP, which performs best according to a given metric such as infinite-horizon d iscounted c ost. A s imulation-based a lgorithm f or a n M DP o btains the optimal policy for an MDP in a *model-free manner*, i.e. without requiring to know apriori any transition probabilities of the MDP under any policy. This work proposes LSPI-CAS, a version of LSPI for *compact* action-sets, thus avoiding the discretization of the available action set in a state and thereby improving control over the system. Regular LSPI works by repeatedly picking the current best action in a state x from a finite feasible set of actions Ax, requiring finding a minimum over |Ax| values. Our variant uses two kinds of parametrization, a feature vector φ(x) for the state called the actor, and ϕ(x, a) for the state-action pair which is the critic. LSPI-CAS employs a stochastic gradient algorithm called Simultaneous Perturbation Stochastic Approximation (SPSA) to update the actor in each iteration. Regular LSPI has a module named Least-Squares Q-Value (LSQ) which we employ as critic to evaluate perturbed policy iterates, and further update the policy iterate in direction of improving performance. Our algorithm is for infinite-horizon discounted-cost/reward MDPs, the case of finite-horizon compact-action set MDPs having been solved in an earlier work. Numerical results on three settings, a. control of an inverted pendulum, b. Exercise Policy calculation for an American Put option, and c. M/M/1/K queue control problem are provided for the algorithm, alongside comparison with LSPI. Improvements in both performance and run-time to find an optimal policy are demonstrated.

I. Introduction to LSPI

An algorithm called Least-Squares Policy Iteration (LSPI) in [1], [2] presents a novel way of solving Markov Decision Processes (MDPs) for their optimal policy. LSPI uses very low levels of apriori information or memory storage. In particular, the base version of LSPI only requires an offline set of L tuples of the type (s, a, r, s'), where s is the starting state of the MDP, a is the discrete action applied as control, r is the reward accruing due to such action or its resultant transition to next state s'.

Investigation into improving LSPI behaviour continues on account of further applicability to the newer cognitive technique of Deep Reinforcement Learning, cf. [3]. We propose an LSPI algorithm that is suited to deterministic, compact action set MDPs that will be applicable to settings where discretization of the action space is not easily possible, or may result in some loss of optimal control.

Simulation-based solutions to MDPs are considered efficient since the transition probability matrices of a given MDP (one for each policy) are not available, in general, and hence closed-form solutions cannot be calculated. Further, *off-policy* algorithms are considered better since every setting may not permit evaluation of the policy iterate π_k and only a memory of transitions (s, a, r, s') (a replay memory) may need to be used. An optimal policy is a policy that achieves the lowest expected cost in some sense: finite, long-run average cost or infinite-horizon discounted cost. A value function of a state x indicates the long-run cost of taking an action in a state, the action being as dictated by the current policy iterate π_k. Within simulation-based solutions, algorithms like LSPI or LSPI-CAS that use *parametrization* of states or state-action pairs to infer a state's value function (rather than actually store each state's value function in a table) are considered efficient in terms of space complexity. This is because the state space of system increases exponentially once a finer discretization is selected or a new dimension for the existing action is added.

A. Brief introduction to LSPI

Consider an MDP of finite state space S, and a finite action space A_x per state x, currently operating under policy π (i.e. $\pi(x) \in A_x$ for a state $x \in S$). In LSPI, the parameterization to which all computation is restricted is that of a state-action pair (x, a). Thus, only a feature-vector $\varphi(x, a) \in \mathcal{R}^K$, $K << |S| \times |A_x|$, of a state-action pair is available and no other storage (e.g. to store exact Q-value $Q(x, a)$ or value-function $V(x)$) is allowed. A weight vector w_π has to be computed such that $w_\pi^T \varphi(x, a) \approx Q_\pi(x, a)$ where $Q_\pi(x, a)$ is defined as the Q-value of (x, a) under policy π, i.e.

$$Q_\pi(x, a) \quad = \quad E\{R(x, a, y) + \gamma Q_\pi(y, \pi(y))\}, \tag{1}$$

with $y \in S$, a random variable, being the state to which transition of the MDP occurs upon taking action a in state x. As an analogy, note that in the tuple (s, a, r, s') explained earlier, x is s, r stands for $R(x, a, y)$ (is also a random variable given x and a), whilst s' is y. Also, $0 < \gamma < 1$ above is the discount factor for this discounted-reward infinite horizon MDP. In the case of LSPI, the policy π used in the definition (1) happens to be an implicit policy as will be explained in the following.

The inner module of the LSPI algorithm is another algorithm called LSQ (for Least-Squares Q-value) (cf. [1, §4]). LSQ is a method for constructing a linear approximaton $\tilde{Q}_\pi(x, a)$ of the Q-function of a state-action pair (x, a) where the exact value $Q_\pi(x, a)$ is as defined in (1). With respect to earlier algorithms in the literature, the novelty of LSQ is that this approximation is performed *without*:

- using any stepsizes, as in stochastic approximation algorithms such as [4], [5] or,
- following any specific distribution for visiting the states of the MDP during a simulated trajectory, as required in [6].

In LSPI during each iteration k, an episode of L, independent, transitions is simulated by LSQ. For $1 \leq l \leq L$, a single transition from a randomly-picked state x_l of the MDP, using a randomly-picked action a_l, is simulated. The values x_l and a_l are both selected uniformly from state space S and the corresponding finite action space A_{x_l}, respectively. This also makes the entire episode suitable for *experience replay*, i.e. the episode can be repeated as input data into each iteration, without hampering the convergence properties of the algorithm. Assume that transitions (x_l, a_l, y_l) with reward $R(x_l, a_l, y_l)$ have been obtained. Now, an update is performed for the $K \times K$ matrix iterate A and the $K \times 1$ vector iterate b using the following recursion:

$$\begin{aligned} A_l &:= A_{l-1} + \varphi(x_l, a_l) \cdot \left(\varphi(x_l, a_l)^T - \gamma \cdot \varphi(y_l, \pi_{k-1}(y_l))^T\right) \\ b_l &:= b_{l-1} + \varphi(x_l, a_l) \cdot R(x_l, a_l, y_l) \end{aligned} \qquad (2)$$

After a large number of transitions, the episode size L is reached and an inversion $w_k = (A_L)^{-1} b_L$ is performed to obtain the weight vector w_k (this is a short-hand of w_{π_k}). It is important to note how the prescribed action $\pi_{k-1}(y_l)$ in (2) is obtained. This policy iterate π_{k-1} is implicit and depends on w_{k-1},

$$\pi_{k-1}(y_l) \quad := \quad \arg \max_{a \in A_{y_l}} w_{k-1}^T \varphi(y_l, a). \qquad (3)$$

The inversion $w_k = (A_L)^{-1}b_L$, of complexity $O(K^3)$, may have to be supported by routines such as singular value decomposition as A_L may not be full-rank. This possibility is however low if L is large enough. Though it is termed Policy Iteration, LSPI holds no explicit policy iterate π_k, and therefore avoids the requirement to store the policy π_k in its exact form (which in turn would have $O(|S|)$ space complexity, or may even be infeasible altogether). It also avoids the need to parameterize the policy π_k as in the algorithm proposed below, or even the possibility of errors owing to such parametrization. However, it assumes that A_{y_l} is a finite set, since finding the maximum otherwise is non-trivial (in general).

A detailed description of LSPI, apart from the source in [2], is also provided in the more recent [7] with LSPI variants as well as an illustration based on the 'car on the hill' problem. Just as the Online-LSPI proposed in [7], there also have been recent extensions in the form of [8] where LSTDQ (equivalent to LSQ) has been used as the core module. LSPI was further enhanced using 'Options' discovered from processed simulation data in recent work of [9]. A stochastic approximation version of LSPI which eliminates the one-shot $O(K^3)$ inversion is to be found in [10].

B. Brief survey of literature

The convenience of visiting states without adhering to any specific distribution is useful when a *generative*, but often complicated, computational model of the system is available. The transition to the next state y when the current state is x and action

$a \in A_x$ is used, can then be simulated and a memory of such transitions created to be used in each LSPI iteration. Previous work in [5] tackled the problem of obtaining a simulation-based solution of Finite-Horizon MDPs (FHMDPs). Specifically, this was accomplished without the handicap of requiring to visit states according to any specific distribution that depends on current policy iterate π_k (cf. [6] and the description of TD(λ) below). The *exploration tradeoff* affects here: though policy π_k may be known exactly, or approximately due to parametrization, the Transition Probability Matrix (TPM) due to π_k is unknown. As a result, the simulated trajectory representing π_k may not be good at exploring the entire state and action space. Work in [11] sets out to adapt LSPI to the continuous action-space setting. However, it requires the simulation-based algorithm RLSAPI to calculate the maximum of a function over the same continous action-space, for each state, and in each iteration (Figure 2, Step 7 in [11]). Finding such a maximum may not be possible owing to lack of a closed form expression, even for the one-step reward $\max_a R(x, a)$.

The analysis of the TD(λ) algorithm (cf. [6]), as also later techniques like LSTD(λ) (cf. [12]) is relevant to the problem of distribution of visiting states. These analyses indicate that visiting states according to any distribution other than the stationary distribution imposed by iterate π_k does not result in estimating the correct value function V_{π_k} when using linear approximation techniques. In particular, in approximating V_{π_k}, one obtains the projected vector $\Pi_{\Phi,\pi_k} V_{\pi_k}$ where the projection depends on policy iterate π_k (apart from the possibly static feature vector matrix constructed as $\Phi = \{\phi^T(x_l)\}_{l=1}^{L}$). For this reason, the first algorithm in [5] is required to follow a π_k-guided trajectory (and is therefore suited only for finite horizon MDPs). These trajectories are to evaluate the performance of *perturbed* policies $\pi_k^+ = \pi_k + \delta_k \Delta_k$ and $\pi_k^- = \pi_k - \delta_k \Delta_k$. These policies are obtained after perturbing policy iterate π_k with Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm's perturbation terms $\delta_k \Delta_k$ where $\delta_k > 0$ is a scalar diminishing stepsize while Δ_k is a vector of $\{-1, +1\}$ perturbations. The second algorithm in [5], also for a finite horizon T, requires two $|S|-$size storage arrays V_1, V_2 for iterates (as opposed to a larger $T \times |S|$ for 'lookup-table' simulation-based FHMDP algorithms). However, though the algorithm does not require to simulate an entire trajectory like the first algorithm and uses the Dynamic Programming principle instead, a dependence on $O(S)$ makes it unqualified for large state-spaces. In [5], the issue in not following a trajectory as specified by the stationary distribution of π_k is that the gradient estimate required by an SPSA update was not similar to the obtained gradient estimate: i.e.

$$\Pi_\Phi \nabla_{\pi_k} V_{\pi_k}(x) \quad \neq \quad E\left(\frac{(\Pi_{\Phi,\pi_k^+} V_{\pi_k^+})(x) - (\Pi_{\Phi,\pi_k^-} V_{\pi_k^-})(x)}{\delta_k \Delta_k(x)}\right).$$

In the following, we introduce a simulation-based algorithm LSPI-CAS that can be considered a variant of LSPI. Both algorithms adapt LSPI to the infinite-horizon, compact-action setting wherein policy iteration has to be performed using stochastic gradient descent: in particular, using the SPSA method briefly described above. The difference between LSPI and our algorithms LSPI-CAS is the presence of three iterate vectors used for approximation inside the algorithm: policy iterate p_k, positively-perturbed policy's value function v_k^+, and the corresponding negative v_k^-. In contrast, LSPI has only one such vector, the w_k from (3) above. However, LSPI-CAS has the advantage that action from a compact set is likely to yield greater control of the MDP in the form of a higher obtainable reward value or faster convergence. For both algorithms, numerical results in three settings are provided and comparisons with LSPI are favourable.

II. Compact Action Set analog of LSPI: LSPI-CAS

Assume that proposed LSPI-CAS only handles one-dimensional actions, i.e. $\pi(x) \in A_x \equiv [a_x, b_x] \subset \mathcal{R}$. In LSPI-CAS, in each iteration k, it is required to recollect the L source states x_l from which transitions were simulated. One may explicitly store the sequence of states $\{x_l\}$ in a memory and incur $O(L)$ space complexity, which would make space complexity vis-a-vis LSPI unfavourable. An alternative is to use the re-hashing method called 'double hashing'. This method is one of the techniques in the closed hashing scheme employed in a Hash Table data structure (cf. [13, §3.4] for a textbook treatment). To perform hashing into an array of size M, a double hashing method can generate L^2 distinct 'probing sequences' of size L using $L-1$ deterministic functions $f_l(\cdot)$, $l = 2, 3, ..., L$. Using the choice of functional $f_1(X) = X$, the sequence is written as $s_X = \{X, f_2(X), f_3(X), ..., f_L(X)\}$ where each element $f_l(X) \in \{1, 2, ..., L\}$

is a distinct index into the array. The variable X is termed the key or the seed, and the sequence s_Y generated for any other key $Y \neq X$ is distinct.

Other than three parameter vectors p_k, v_k^+ and v_k^-, LSPI-CAS will also require a per-state feature vector $\phi(x_l)$, a column vector of dimension $K \times 1$. This is in addition to a feature vector for every selected state-action pair $\phi(x_l, a_l)$ of $M \times 1$ like in LSPI. Note that the feature vectors are composed only upon observation of the state x_l or the state-action pair (x_l, a_l). The action in state x_l specified by a policy iterate π_k is obtained as a dot product: $\pi_k(x_l) = P_{[a_{x_l}, b_{x_l}]}({p_k}^T \phi(x_l))$ where the function $P_{[a_{x_l}, b_{x_l}]}(\cdot)$ represents truncation into the compact action set $[a_{x_l}, b_{x_l}]$. Similarly, the gradient of the value function V_{π_k} will require to be calculated.

For reference, recall the Bellman equation:

$$V_{\pi_k}(x_l) \quad := \quad E\left\{R(x_l, \pi_k(x_l), \eta(x_l, \pi_k(x_l))) + \gamma \cdot V_{\pi_k}(\eta(x_l, \pi_k(x_l)))\right\}, . \qquad (4)$$

Note the approximation $V_{\pi_k^+}(x_l) \approx (v_k^+)^T \varphi(x_l, \pi_k(x_l))$ where $\pi_k(x_l) = P[a_{x_l}, b_{x_l}](p_k^T \phi(x_l))$. The calculation of v_k^+ will however rely on simulating a transition from x_l using action $\pi_k^+(x_l) = P_{[a_{x_l}, b_{x_l}]}(\pi_k(x_l) + \delta_k \Delta_k(x_l))$ where $\Delta_k(x_l) \in \{-1, +1\}$ is the $l-$th component of the perturbation vector Δ_k. Thus the *policy gradient* $\nabla_\pi V_{\pi_k}(x_l)$, will be calculated as:

$$\nabla_\pi V_{\pi_k}(x_l) \quad \approx \quad \frac{((v_k^+)^T \varphi(x_l, \pi_k(x_l)) - (v_k^-)^T \varphi(x_l, \pi_k(x_l)))}{2\delta \Delta_k(x_l)}.$$

In LSPI-CAS, L is the number of transitions used to calculate v_k^+ and v_k^- using the LSQ module, and thus update the policy iterate p_k.

An infinite horizon discounted reward MDP is considered here, and positive step sizes $1 >> \delta_k > 0$, and a_k are used with conditions: $\sum_{k=1}^{\infty} a_k^2 < \infty$ and $\sum_{k=1}^{\infty} a_k = \infty$. The function $P_\pi(a)$ below is shorthand for $P_{[a_{x_l}, b_{x_l}]}(a)$ earlier. It is used to project the action a into the *feasible* compact action set $A_{x_l} \equiv [a_{x_l}, b_{x_l}]$ for each state x_l. Though not identified separately, we assume that P_π is applicable for each state x_l. The proposed algorithm LSPI-CAS, which uses the LSQ module in steps (b)-(d), can be applied to obtain an approximate optimal policy for a Compact Action Set MDP:

1) For each iteration k, repeat until $|p_{k+1} - p_k| \to 0$ as $k \to \infty$:
    a) Generate a state X_k from S to use as seed for a $L-$size sequence $\{x_l\}$
    b) Do L times with index l:
        i) Compute state $x_l = f_l(X_k)$.
        ii) Generate perturbation: $\Delta_{k,l} = \{-1, +1\}$ with equal probability.
        iii) Store $\Delta_{k,l}$ as $l-$th bit in an $L-$bit binary word Δ_k.
        iv) Calculate action $a_l \triangleq P_\pi({p_k}^T \phi(x_l))$ in x_l to use in (5)-(6) below.
        v) Use action $a_l^+ \triangleq P_\pi({p_k}^T \phi(x_l) + \delta_k \Delta_{k,l})$ in x_l to transition to state $y_l^+ \in S$.
            A) Compute reward $R_l^+ = R(x_l, a_l^+, y_l^+)$ and action $P_\pi({p_k}^T \phi(y_l^+))$ at y_l^+.
            B) Update A_l^+ and b_l^+ as in (5) below.
        vi) Use action $a_l^- \triangleq P_\pi({p_k}^T \phi(x_l) - \delta_k \Delta_{k,l})$ in x_l to transition to state $y_l^- \in S$.

```
        A) Compute reward $R_l^- = R(x_l, a_l^-, y_l^-)$ and action $P_\pi({p_k}^T \phi(y_l^-))$ at $y_l^-$.
        B) Update $A_l^-$ and $b_l^-$ as in (6) below.
    c) Calculate $v_k^+ := (A_L^+)^{-1} b_L^+$.
    d) Calculate $v_k^- := (A_L^-)^{-1} b_L^-$.
    e) do $L$ times with index $l$
        i) Re-compute $x_l = f_l(X_k)$ and perturbation $\Delta_{k,l}$ as the $l-$th bit from $\Delta_k$.
        ii) Compute $l-$th element of vector $r_k$ as in (7)
    f) Perform update $p_{k+1} = (\Phi_k^T \Phi_k)^{-1} \Phi_k^T \cdot r_k$
```

$$\begin{aligned} A_l^+ &:= A_{l-1}^+ + \varphi(x_l, a_l) \cdot \left(\varphi(x_l, a_l)^T - \gamma \cdot \varphi(y_l^+, P_\pi({p_k}^T \phi(y_l^+)))^T \right) \\ b_l^+ &:= b_{l-1}^+ + \varphi(x_l, a_l) \cdot r_l^+ \end{aligned} \tag{5}$$

$$\begin{aligned} A_l^- &:= A_{l-1}^- + \varphi(x_l, a_l) \cdot \left(\varphi(x_l, a_l)^T - \gamma \cdot \varphi(y_l^-, P_\pi({p_k}^T \phi(y_l^-)))^T \right) \\ b_l^- &:= b_{l-1}^- + \varphi(x_l, a_l) \cdot r_l^- \end{aligned} \tag{6}$$

$$r_{k,l} := P_{[a_{x_l}, b_{x_l}]}(P_{[a_{x_l}, b_{x_l}]}({p_k}^T \phi(x_l)) - a_k \{v_k^{+,T} - v_k^{-,T}\} \varphi(x_l, a_l)(2\delta \Delta_{k,l})^{-1}) \tag{7}$$

The matrix Φ_k is an $L \times K$ matrix with the $l-$th row being $\phi^T(x_l)$. In the above algorithm, policy gradient has been approximated using projected SPSA (7) and the value function estimates have been obtained using the LSQ method ((5)-(6)). The specific term approximated in (7) to correct existing policy $P_\pi({p_k}^T\phi(x_l))$ is $E\left\{\frac{\tilde{V}_k^+(x_l)-\tilde{V}_k^-(x_l)}{2\delta_k\Delta_k}\right\}$. Here, $\tilde{V}_k^+(x_l)$ and $\tilde{V}_k^-(x_l)$ are linear approximations to the value functions $V_{\pi_k^+}(x_l)$, $V_{\pi_k^-}(x_l)$ similar to (4) above but when using perturbed policies π_k^+ and π_k^-. The $L-$size column vector r_k is then used in the recursion (1f) as a proxy for current policy π_k updated using gradient $\{\nabla_{\pi_k}V_{\pi_k}(x)\}_{x\in S}$. The L iterations in Step 1b of the algorithm serve the purpose of covering uniformly the entire state space S, to approximate matrix Φ in Φ_k in an expected sense. Here, Φ is an $|S|\times K$ matrix with the $s-$th row being $\phi^T(x_s)$ for the state numbered x_s. The use of x_l drawn from double hashing function $x_t = f_t(X_k)$ aids in recollection, without storage, of the entire sequence of L states visited. The perturbations used are similarly recovered from the $L-$bit word Δ_k composed during the Step 1b. Also note that term $(\Phi_k^T\Phi_k)$, as well as the projection portion of the iteration in Step 1f, can be calculated incrementally without requiring any $O(|S|)$ data structure. This potentially permits continuous state-space, compact action-set MDPs to be solved for an optimal policy via simulation.

The difference with LSPI can be visualized in (5) above. Instead of computed a_l^+ or a_l^-, LSPI would have chosen a random action a_l from the discrete set A_{x_l} to which it applies. LSPI would also have a single update instead of ((5)-(6)) here. The feature vector $\phi(y_l^+, P_\pi({p_k}^T\phi(y_l^+)))$ would be substituted with equivalent $\phi(y_l, a_k(y_l))$ where $a_k(y_l) = \arg\max_{a\in A_{y_l}}\{w_{k-1}^T\phi(y_l,a)\}$, using the earlier notation. It is in this latter step that computational delays occur, for calculating the $\arg\max_{a \in A_{y_l}}$ from a set whose values change due to changing w_k. Note that LSPI-CAS here does not need a Q-value parameter w_k to infer the current best action in y_l^+ or y_l^-.

III. Convergence Proof

Notice the two kinds of matrices, state feature vector matrix Φ_k in each iteration k, and a related state feature vector matrix composed of $\varphi(x, \pi_k(x))$ that encodes information due to current policy π_k. Thus, Φ_k has dimension $L \times K$ while the matrix composed of $\varphi(x, \pi_k(x))$ would be a matrix with dimension $L \times (K + m)$ with the first K columns being same as Φ_k. The effective policy update of $\pi_k = \Phi_k p_k$ is operationalised as $\pi_{k+1} = \Phi_{k+1} p_{k+1}$ after the $k-$th update. The intuition behind this variation is that the difference in the two perturbed actions (calculated using action $p_k^T \phi(x_l)$) prescribed for state x_l under SPSA is approximately $\Delta_k^T \phi(x_l)$. The perturbations in LSPI-CAS satisfy the conditions on SPSA random perturbations given in [14]. Consider notation Φ_{π_k} indicating the feature vector matrix Φ_{k,π_k} above, but for all states $x_l \in S$, hence of dimension $|S| \times (K + m)$. Assume that $\tilde{V}_k^+$ is the $|S|-$size column vector constructed using $\tilde{V}_k^+(x_l)$ above for all x_l. Also assume that P_k^+ is the $|S| \times |S|$ transition probability matrix where $P_k^+(i, j)$ indicates probability of transition from i to j when action $\pi_k^+(i)$ is used. Similarly, let R_k^+ be the $|S| \times 1$ vector where $R_k^+(i)$ is the expected reward in state i when action $\pi_k^+(i)$ is used. The vector v_k^+, such that $\tilde{V}_k^+ = \Phi v_k^+$, is the solution to the balance equation (more detailed algebra available in [2, p. 1117]):

$$\begin{aligned}
\Phi_{\pi_k}(\Phi_{\pi_k}^T\Phi_{\pi_k})^{-1}\Phi_{\pi_k}^T\left(\mathcal{R}_k^+ + \gamma P_k^+\tilde{V}_k^+\right) &= \tilde{V}_k^+ \\
\Phi_{\pi_k}(\Phi_{\pi_k}^T\Phi_{\pi_k})^{-1}\Phi_{\pi_k}^T\left(\mathcal{R}_k^+ + \gamma P_k^+\Phi_{\pi_k}v_k^+\right) &= \Phi_{\pi_k}v_k^+ \\
\Phi_{\pi_k}v_k^+ - \Phi_{\pi_k}(\Phi_{\pi_k}^T\Phi_{\pi_k})^{-1}\Phi_{\pi_k}^T\left(\gamma P_k^+\Phi_{\pi_k}v_k^+\right) &= \Phi_{\pi_k}(\Phi_{\pi_k}^T\Phi_{\pi_k})^{-1}\Phi_{\pi_k}^T\mathcal{R}_k^+ \\
\Phi_{\pi_k}^T\Phi_{\pi_k}v_k^+ - \Phi_{\pi_k}^T\left(\gamma P_k^+\Phi_{\pi_k}v_k^+\right) &= \Phi_{\pi_k}^T\mathcal{R}_k^+ \\
\Phi_{\pi_k}^T\left(\Phi_{\pi_k} - \gamma P_k^+\Phi_{\pi_k}\right)v_k^+ &= \Phi_{\pi_k}^T\mathcal{R}_k^+ \\
v_k^+ &:= \left(\Phi_{\pi_k}^T(\Phi_{\pi_k} - \gamma P_k^+\Phi_{\pi_k})\right)^{-1}\Phi_{\pi_k}^T\mathcal{R}_k^+
\end{aligned}$$

The estimation of the RHS in the latter is being performed in two parts via sampling, $A_k^+ \approx \left(\Phi_{\pi_k}^T(\Phi_{\pi_k} - \gamma P_k^+\Phi_{\pi_k})\right)$, while $b_k^+ \approx \Phi_{\pi_k}^T\mathcal{R}_k^+$, just as in LSPI. Note that a symmetric method applies for calculating A_k^- and b_k^- also. In the above, we could use $\Phi_{\pi_k^+}$ instead of Φ_{π_k}, however the action corresponding to π_k^+ is unknown in next state y_l^+. With the feature vectors that we have chosen, we make a key assumption regarding the approximation architecture:

$$||\Phi_{\pi_k}v_k^+ - V_{\pi_k^+}||_\infty \leq \epsilon_0, \forall k,$$

where $V_{\pi_k^+}$ is the exact value function vector corresponding to policy π_k^+. Such an assumption is also made in the result [2, Theorem 7.1]. Note that the v_k^+ used corresponds to equations above, rather than the equivalent variable in LSPI-CAS. Now consider shorthand for orthogonal projection operators as follows: $\Pi = \Phi(\Phi^T\Delta\Phi)^{-1}\Phi^T$, and $\Pi_{\pi_k} = \Phi_{\pi_k}(\Phi_{\pi_k}^T\Delta_{\pi_k}\Phi_{\pi_k})^{-1}\Phi_{\pi_k}^T$. Here, Δ and Δ_{π_k} are diagonal matrices capturing stationary distribution of the algorithm visiting the states of the MDP. Thus, for example, $\Phi_{\pi_k} v_k^+ \approx \Pi_{\pi_k} V_{\pi_k^+}$. We also use, interchangeably, p and π s.t. $\pi = \Phi p$.

Theorem 1. *π_k converges w.p. 1 to a π^* in A s.t. $A = \{\pi^* | \Pi \nabla_\pi V_{\pi^*} = 0\}$.*

Proof: The steps (e)-(f) in LSPI-CAS above can be rewritten, using $\pi_k = \Phi p_k$ as:

$$\pi_{k+1} \quad := \quad \Pi(P(P(\pi_k) - a_k \Pi_{\pi_k} \nabla_{\pi_k} V_{\pi_k} - a_k \epsilon_k)),$$

where P is the truncation operator applied elementwise to each component of π_k as well as the updated form of π_k. In the above, ϵ_k indicates a term (incorporating ϵ_0) that combines finite-difference approximation error in gradient due to SPSA as well as noise. In particular, the SPSA term in (7) is a finite-difference approximation to $\nabla_\pi \Pi_\pi V_\pi$. Applying [5, Theorem 1], we have convergence of π_k to the set $\{\pi^* | \Pi \Pi_\pi \nabla_\pi V_\pi = 0\}$. Note that $\Pi \Pi_\pi \nabla_\pi V_\pi = \Pi \nabla_\pi V_\pi$ due to the space spanned by columns of Φ being a subset of similar space spanned by columns of Φ_π. ●

If the space of Φ is not a proper subset of Φ_π, then the projection applicable would be in the possibly null intersection of Π and Π_π. This holds true irrespective of how good Φ and Φ_π are in terms of approximation architecture. Similary, working with a case of $\Phi = \Phi_\pi$, $\forall \pi$ is also feasible in the LSPI-CAS algorithm. Note that of the 3 numerical examples we have considered in the next section, the trailing 2 belong to such categories, i.e. $\Phi = \Phi_\pi$ (Options Excercise policy) and space$(\Phi) \cap$space$(\Phi_\pi) \neq \phi$ (M/M/1/K queue control).

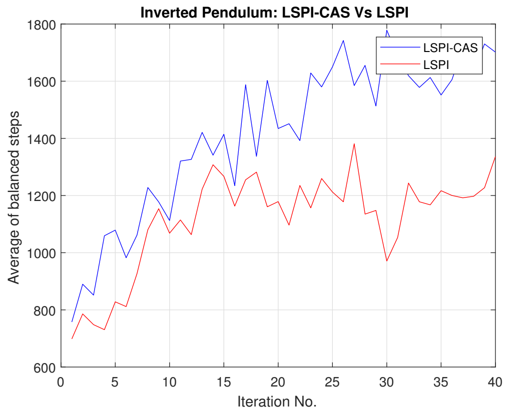
IV. Numerical Results

The inverted pendulum example in [2] has been broadly adapted to demonstrate the suitability of LSPI-CAS. This setting is also found in [15], where the algorithms proposed were different and without proof (albeit for the same setting). The inverted pendulum guided by a differential equation in its parameters θ radians (the angle made with the vertical, clockwise being negative) and angular velocity $\dot{\theta}$ radians/second. A force $F \in [-100, 100]$ Newtons can be applied every 0.01s to the cart bearing the pendulum so as to correct it and bring θ and $\dot{\theta}$ as close to 0, the equilibrium position, as possible. The application interval chosen is 0.01s since the inverted pendulum's equations of motion are suitably linearized and hold only for small Δt intervals. The application of the force, however, is taken to be noisy (e.g. an imprecise push or pull by a less-sophisticated device) due to an additive noise sample picked uniformly from $[-20, 20]$ Newtons.

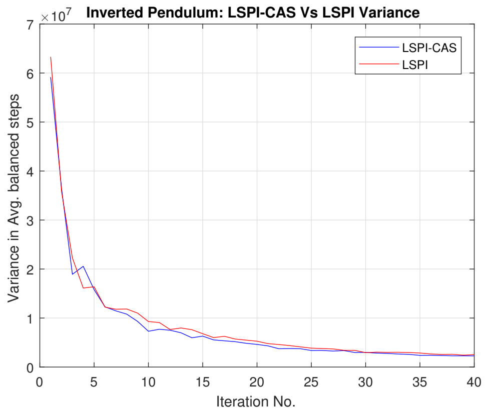
The cost for a transition to a destination state $(\theta_d, \dot{\theta}_d)$ is taken to be $\frac{|\theta_d|+|\dot{\theta}_d|}{\frac{\pi}{2}+2}$. The discount factor used in the relevant infinite-horizon discounted-cost MDP is assumed 0.9. The starting state consists of element θ_s chosen randomly from $[-\frac{\pi}{10}, \frac{\pi}{10}]$ and $\dot{\theta}_s$ in the interval $[-0.2, 0.2]$ to mimic a small disturbance from equilibrium. The 'actor' feature vector for LSPI-CAS took the form $\phi(x) = \{\phi_0(x), \phi_1(x), ..., \phi_K(x)\}$ with $K = 9$, $\phi_0(x) = 1.0$, and a Radial Basis Function (RBF) approximation $\phi_k(x) = \exp(\frac{-1}{2}||x - x_k||_2^2)$ for $k = 1, ..., K$ is used. The points x_k are in the grid format (A_1, A_2) with $A_1 \in \{\frac{-\pi}{4}, 0, \frac{\pi}{4}\}$ and $A_2 \in \{-1, 0, 1\}$. An RBF grid of $K = 27$ points for $\varphi(x, a)$

of regular LSPI (also required for LSQ module of proposed LSPI-CAS) has the format (A_1, A_2, A_3) with $A_1 \in \{\frac{-\pi}{4}, 0, \frac{\pi}{4}\}$, $A_2 \in \{-1, 0, 1\}$ and $A_3 \in \{-100, 0, 100\}$. The feature vector dimension would therefore be 28 for regular LSPI, while it would be 10 for the actor and 28 for critic in LSPI-CAS.

In every trial, the optimal policy is calculated using 150 transition samples per iteration for 40 iterations. This is in order to match with the 6000 samples used per experiment in [2]. Next, the computed optimal policy is tested such that 3000 steps are taken (corresponding to $30s$ of trying to balance the inverted pendulum). The starting state is chosen randomly as before: θ_s from $[-\frac{\pi}{10}, \frac{\pi}{10}]$ and $\dot{\theta}_s$ from $[-0.2, 0.2]$ to mimic a small disturbance. The testing stops if $\theta \geq \frac{\pi}{2}$ – indicating that the pendulum has fallen to ground level. For each $n \leq N$ with $N = 40$, 100 trials are performed and the average number of steps that the algorithm was able to balance are plotted. The stepsize used for the iteration (7) above was $a_k = \frac{1}{(k+1)}$. The discrete actions required by LSPI were 11, $\{-100N, -90N, ..., -10N, 0N, 10N, ..., 90N, 100N\}$, representing a fairly fine discretization of the action space $[-100, 100]$ Newtons. The computational complexity associated with identifying the minimum resulted in approximately 1200s for the 100 runs of LSPI. This is compared to approximately 150s, an order of magnitude lesser, for LSPI-CAS Algorithm, all tests being performed on MATLAB. Figure 2a captures the performance of proposed LSPI-CAS algorithm and compares it with LSPI. Figure 2b compares variance in the average balanced steps' computation indicating both algorithms



(a) Average balanced steps in Inverted Pendulum problem

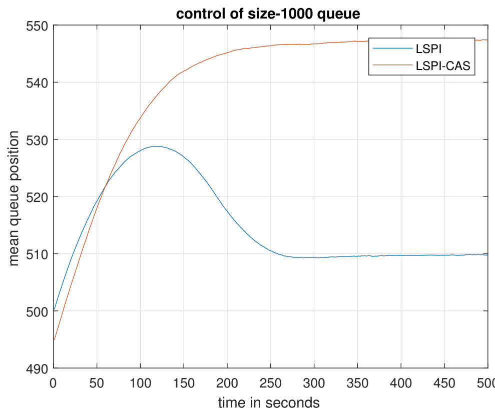


(b) Comparing variance when average of balanced steps was calculated

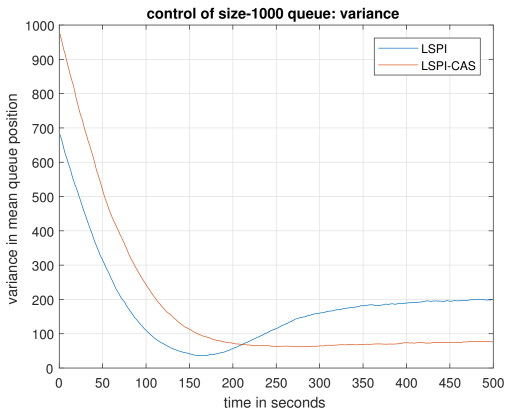
Fig. 1: LSPI-CAS Vs LSPI on Inverted Pendulum

calculate better quality policies as N increases.

A second experiment was run for rate-control of an M/M/1/K queue, in the setting of [4]. The aim is to stabilize the $K-$sized queue near $\frac{K}{2}$ by varying the rate of packet arrivals λ between $[0.05, 4.5]$/second, where $\lambda_{\max} = 4.5$. The mean rate of packets being processed by the server is $\mu = 2.0/s$ while there is also an uncontrolled rate of arrivals (e.g. priority packets) at $0.2/s$. A difference in this implementation of LSPI-CAS is the use of *critic averaging* as suggested in [4], viz. the episode size L above is composed as $L = L' {\cdot} S$. Here, L' is the number of distinct x_l while S is the number of repeat transitions simulated from these x_l. In comparison, for the inverted pendulum example above $S = 1$. We considered a queue of size $K = 1000$, for which a lookup table of size 1000 would



(a) Average queue size using LSPI-CAS (§II)



(b) Comparison of variance in mean queue sizes

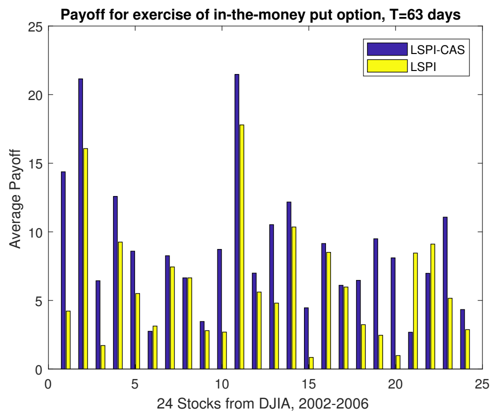
Fig. 2: LSPI Experiments on M/M/1/1000 queue control

be required if the Compact-Action set algorithm of [4] were used. In contrast, LSPI-CAS used a size-4 polynomial feature vector $\phi(x) = (1, \frac{x-K/2}{K/2}, \{\frac{x-K/2}{K/2}\}^2, \{\frac{x-K/2}{K/2}\}^3)$ for the actor. The parameters were $L' = 334$ and $S = 3$, while the critic LSQ feature was $\varphi(x,a) = (1, |x-K/2|, |a-\lambda_{\text{max}}/2|, (x-K/2)*(a-\lambda_{\text{max}}/2))$. The model used for the MDP is an embedded chain with period 1s and transition probability matrices calculated apriori for a dense set of λ (90 actions over $[0.05, 4.5]$) using M/M/1/K formulae. Even though the λ chosen by the policy could be any value in $[0.05, 4.5]$ it is rounded off to the nearest of these 90 values in order to use the associated transition probability.

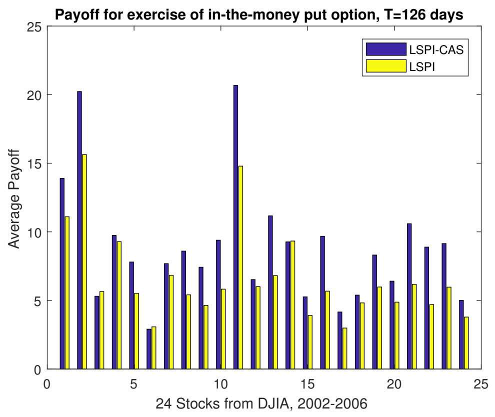
Among applications in literature of LSPI is the American Options exercise policy problem (cf. [16], [17]). Notice in [17, Table 1] 30 stocks of the Dow Jones 30 (DJ30) in the period 2002-2006 to simulate using Geometric Brownian Motion (GBM) the performance of an in-the-money put American option. For reasons such as companies exiting and entering the DJ30 and unusually high estimated σ for some cases, we included only 24 of these stocks to simulate and evaluate between LSPI and LSPI-CAS. For each stock the volatility σ against a risk-free return μ is estimated from daily closing stock prices in 2002-2006. After this, $50,000$ trajectories of GBM simulating the security are generated and both algorithms are trained for $63-$, $126-$ or $252-$days options. In particular, the LSPI variant uses entire trajectories as well as the feature vector architecture in [16], [17]. LSPI-CAS instead uses transitions from uniform-randomly selected (s,t): s being stock price in a range $[0.5S_0, 1.5S_0]$ where S_0 is that stock's starting price. Similary, t is an integer randomly selected from $[1,T]$ (where T is $63, 126$, or 252 as the case may be). A statistical significance indicator of better payoff is also calculated, since the average payoff calculated in $10,000$ trajectories may contain variance. A summary of the performance, in terms of the average payoffs across these stocks, is given as under:

Days T of Option Term	LSPI-CAS	LSPI-Regular	Significant (out of 24)
63	3.90	3.27	21
126	5.19	4.59	20
252	7.31	5.02	21

The significance was measured at the 5% level. The performance of LSPI-CAS in this setting is captured in the following graphs. In each case, about 18-19 stocks from the



(a) Average Payoff when Exercise Policy is calculated using LSPI-CAS Vs LSPI, T=63 days



(b) Average Payoff when Exercise Policy is calculated using LSPI-CAS Vs LSPI, T=126 days

Fig. 3: LSPI Experiments on American Option Exercise Policy

24 have favourable payoffs when LSPI-CAS is used, even if such stocks are excluded wherein significance is not indicated. The average payoff for the entire portfolio is favourable for LSPI-CAS by a minimum of 15% as seen in the table.

V. Conclusions

This article proposed an algorithm for Compact Action Set MDPs to be solved using the basic structure of Least Squares Policy Iteration. A proof with probability 1 was given, which also relies on intersection between the space spanned by two feature vector matrices used in the algorithm. The algorithm and a close variant were tested on settings of inverted pendulum, M/M/1/K queue and American Options problems, all modeled as MDPs. As part of future work, it would be useful to compare LSPI-CAS against stochastic multi-armed bandit algorithms adapted to simulation-based MDPs, an example of the latter being in [18].

REFERENCES

[1] M. Lagoudakis and R. Parr, “Model-Free Least Squares Policy Iteration,” in *Proceedings of NIPS*2001*, 2001, pp. 1547–1554.

[2] M. G. Lagoudakis and R. Parr, “Least-Squares Policy Iteration,” *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[3] W. Bohmer, R. Guo, and K. Obermayer, “Non-Deterministic Policy Improvement Stabilizes Approximate Reinforcement Learning,” in *Proceedings of 13th European Workshop on Reinforcement Learning*, 2016.

[4] S. Bhatnagar and S. Kumar, “A Simultaneous Perturbation Stochastic Approximation–Based Actor–Critic Algorithm for Markov Decision Processes,” *IEEE Transactions on Automatic Control*, vol. 49, no. 4, pp. 592–598, 2004.

[5] M. S. Abdulla and S. Bhatnagar, “Parameterized Actor-Critic Algorithms for Finite Horizon MDPs,” in *Proc. American Control Conference*, 2007.

[6] J. N. Tsitsiklis and B. V. Roy, “An Analysis of Temporal-Difference Learning with Function Approximation,” *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[7] L. Busoniu, A. Lazaric, M. Ghavamzadeh, R. Munos, R. Babuska, and B. D. Schutter, “Least-squares methods for policy iteration,” in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Eds. Springer, 2011, no. 12, pp. 75 – 109.

[8] N. Tziortziotis, C. Dimitrakakis, and K. Blekas, “Cover Tree Bayesian Reinforcement Learning,” *Journal of Machine Learning Research*, vol. 15, pp. 2313–2335, 2014.

[9] C. Daniel, H. van Hoof, J. Peters, and G. Neumann, “Probabilistic Inference for Determining Options in Reinforcement Learning,” *Machine Learning*, vol. 104, no. 2-3, pp. 337–357, 2016.

[10] P. L. A., N. Korda, and R. Munos, “Fast LSTD using stochastic approximation: Finite time analysis and application to traffic control,” in *7th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, 2014.

[11] J. Ma and W. Powell, “A convergent recursive least squares approximate policy iteration algorithm for multi-dimensional Markov decision process with continuous state and action spaces,” in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning ADPRL ’09*, 2009.

[12] J. A. Boyan, “Least Squares Temporal Difference Learning,” in *Proceedings of 16th ICML*, 1999.

[13] Y. Narahari, “Electronic Lecture Notes on Data Structures and Algorithms,” *Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India*, 2000.

[14] J. C. Spall, “Multivariate stochastic approximation using a simultaneous perturbation gradient approximation,” *IEEE Transactions on Automatic Control*, vol. 37, no. 1, pp. 332–341, 1992.

[15] M. S. Abdulla and S. Bhatnagar, “Least-Squares Policy Iteration for Markov Decision Processes with Compact Action Sets,” in *Proc. of 1st Indian Control Conference*, 2015.

[16] Y. Li, C. Szepesvari, and D. Schuurmans, “Learning Exercise Policies for American Options,” in *Proc. 12th International Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[17] Y. Li and D. Schuurmans, “Policy Iteration for Learning an Exercise Policy for American Options,” in *S. Girgin et al. (Eds.): EWRL 2008, LNAI 5323, pp. 165178*, 2008.

[18] S. Padakandla, P. K. J., and S. Bhatnagar, “Energy Sharing for Multiple Sensor Nodes With Finite Buffers,” *IEEE Transactions on Communications*, vol. 63, no. 5, pp. 1811–1823, 2015.

"The unexamined life is not worth living"
-Socrates
Welcome to iimk