IFAC Journal of Systems and Control 13 (2020) 100106

Contents lists available at ScienceDirect

IFAC Journal of Systems and Control

journal homepage: www.elsevier.com/locate/ifacsc

# A Least-Squares Temporal Difference based method for solving resource allocation problems

Ali Forootani [a], Massimo Tipaldi [a,*], Majid Ghaniee Zarch [b], Davide Liuzza [c], Luigi Glielmo [a]

[a] University of Sannio, Department of Engineering, Piazza Roma, 82100 Benevento, Italy
[b] Bu-Ali Sina University, Department of Electrical Engineering, Hamedan, Iran
[c] ENEA, Fusion and Nuclear Safety Department, Frascati, Rome, Italy

ARTICLE INFO

*Article history:*
Received 18 February 2019
Received in revised form 27 June 2020
Accepted 7 August 2020
Available online 22 August 2020

*Keywords:*
Least-squares temporal difference
Approximate dynamic programming
Markov decision process
Birth–death process
Monte Carlo simulations

ABSTRACT

Value function approximation has a central role in Approximate Dynamic Programming (ADP) to overcome the so-called curse of dimensionality associated to real stochastic processes. In this regard, we propose a novel Least-Squares Temporal Difference (LSTD) based method: the "Multi-trajectory Greedy LSTD" (MG-LSTD). It is an exploration-enhanced recursive LSTD algorithm with the policy improvement embedded within the LSTD iterations. It makes use of multi-trajectories Monte Carlo simulations in order to enhance the system state space exploration.

This method is applied for solving resource allocation problems modeled via a constrained Stochastic Dynamic Programming (SDP) based framework. In particular, such problems are formulated as a set of parallel Birth–Death Processes (BDPs). Some operational scenarios are defined and solved to show the effectiveness of the proposed approach. Finally, we provide some experimental evidence on the MG-LSTD algorithm convergence properties in function of its key-parameters.

## 1. Introduction

Stochastic Dynamic Programming (SDP) is a general method for modeling and solving sequential decision problems under uncertainty. The first comprehensive books on such topic were written by Bellman (1957) and Howard (1960). Since then, SDP based approaches have been widely applied in many fields, see Bertsekas (2017) and Powell (2011). In principle, there exist consolidated approaches to solve SDP problems, and calculate (even exactly) the corresponding optimal policy. In particular, we can mention the Value Iteration (VI), the Policy Iteration (PI) and the linear programming methods (Bertsekas, 2012a), chapter 2. However, as for real-world applications, their applicability is limited by severe scalability problems. It is worth highlighting that such issue (also known as the curse of dimensionality) can be caused by both the system state space and the set of alternative decisions at each state (action space). As a consequence, real-world problems become intractable due to computational burden and memory requirements. In this regard, Approximate Dynamic Programming (ADP) proves to offer powerful tools for addressing such scalability issues (Bertsekas, 2012a), chapter 6. ADP is a flourishing research area, emerged through a fruitful cross fertilization of ideas from artificial intelligence, optimal control theory, and operations research. Very recent applications can be found in different fields, such as smart home energy management system (Keerthisinghe, Verbic, & Chapman, 2016), multi-agent robotic systems (Deng, Chen, & Belta, 2017), spacecraft mission operations planning (D'Angelo, Tipaldi, Palmieri, & Glielmo, 2019; Tipaldi & Glielmo, 2018), and power systems (Guo, Liu, Si, He, Harley, & Mei, 2016; Tang, He, Wen, & Liu, 2015).

The ADP methods can be grouped into two main classes, that is to say, approximation in value space and approximation in policy space (Bertsekas, 2012a), chapter 6. In this paper, we focus on the former, and more specifically on the value function approximation. A value function gives the expected cumulative reward when starting from a specific state, and then following a given policy (expectation being done over all possible trajectories). An optimal policy is one that maximizes the value function for each state (Bertsekas, 2012a). In this context, the main technique to address the curse of dimensionality is to approximate such value function via a more compact parametric representation, which is also referred to as the approximation architecture (Bertsekas, 2011a; De Farias & Van Roy, 2003; Geist & Pietquin, 2013). More specifically, in this paper, we approximate the (optimal) value function by adopting a feature extraction mechanism to map the original set of states onto a much smaller set of feature vectors.

Among the different value function approximation methods (Geist & Pietquin, 2013), we consider the least-squares projected fixed point category (Bertsekas, 2011b). In literature, we find two

* Corresponding author.
*E-mail addresses:* aforootani@unisannio.it (A. Forootani), mtipaldi@unisannio.it (M. Tipaldi), m.ghaniee@basu.ac.ir (M.G. Zarch), davide.liuzza@enea.it (D. Liuzza), glielmo@unisannio.it (L. Glielmo).

https://doi.org/10.1016/j.ifacsc.2020.100106
2468-6018/© 2020 Elsevier Ltd. All rights reserved.

main techniques, that is to say, the Least Square Temporal Difference (LSTD) and Least Squares Policy Evaluation (LSPE) (Bertsekas, 2011a; Geist & Pietquin, 2013). They adopt Monte Carlo simulations to compute the approximate value function of a given policy in the lower dimensional feature space. The key idea behind simulations is to express the SDP optimality condition as expected value with respect to some probability distribution, and then to approximate the expected value by sampling according to such distribution. Both LSTD and LSPE are performed as the policy evaluation of the PI algorithm. In particular, an approximate value function of a fixed policy is calculated, and then the policy improvement step is performed separately, by assuming that it can be performed even when we have a large state space (Bertsekas, 2012a), paragraph 6.1.2. Hereafter, we consider that such assumption is not valid, thus we propose to incorporate the policy improvement step within the Monte Carlo simulations.

In this paper, we propose a novel LSTD based approach, which is an exploration-enhanced recursive LSTD algorithm with the PI policy improvement embedded within the LSTD iterations. We call this algorithm "Multi-trajectory Greedy LSTD" (MG-LSTD). Instead of having a predetermined policy, we actually apply a policy derived from a step-by-step optimization process. In other words, we address the curse of dimensionality both in the state and in the action spaces. In particular, we calculate the approximate value function via multi-trajectories Monte Carlo simulations in order to enhance the system state exploration and to generate a richer mixture of state visits. We assume that a good estimation of the underlying MDP state transition probabilities is available. Thus, the MG-LSTD is a model-based approach. However, it could also work in case we have a computer simulator (with the capability of generating samples according to such probability distributions) or the real system, which could make the proposed algorithm closer to model-free approaches, such as the Least Squares Policy Iteration (LSPI) (Lagoudakis & Parr, 2003). In such a case, the MG-LSTD algorithm should be extended to approximate Q-factor functions, rather than value functions (Bertsekas, 2019a).

The MG-LSTD algorithm is applied to solve resource allocation problems, which are modeled by means of parallel Birth–Death processes (BDP) (Bertsekas & Tsitsiklis, 2000; Crawford, Minin, & Suchard, 2014). Dynamic pricing policies can be applied, meaning that price managers can propose different prices for the same resource over time with the aim of maximizing the expected total revenue. In our model, each feasible price corresponds to a specific BDP, while stochastic arrivals and departures of customers are respectively the birth and the death of such BPDs. This way, customers can hold a resource as long as they like. The complete formulation becomes a SDP problem aiming at maximizing the expected total revenue over an infinite time horizon. Its underlying optimization problem is solved by the proposed algorithm with the aim of addressing the curse of dimensionality. Significant operational scenarios are defined and solved with the support of a Matlab based application in order to show the effectiveness of the proposed approach.

The rest of the paper is organized as follows. Section 2 provides some background on ADP and projected fixed point approaches. Section 3 describes the proposed MG-LSTD algorithm, and investigates the properties of the resulting value function approximation as well as some connections with the LSPE. Section 4 analyzes the motivation of the MG-LSTD algorithm, its key-aspects, and its connections with other approaches reported in the literature. Resource allocation problem formulation via a constrained SDP/ADP framework is addressed in Section 5. Section 6 shows numerical results of the MG-LSTD algorithm when used to solve resource allocation problems. Given the empirical nature of the paper, such section shows some experimental evidence on the MG-LSTD algorithm convergence properties in function of its key-parameters and provides some guidance on the proper application of the MG-LSTD algorithm. Finally, Section 7 concludes the paper.

## 2. Preliminaries on ADP and projected Bellman equation

This section provides an essential background on some mathematical models and tools we will exploit in the MG-LSTD algorithm and the resource allocation problem formulation. For a more rigorous and detailed dissertation on the topics here briefly outlined, the reader can refer to the cited literature.

In this paper, we use the value function approximation to cope with the curse of dimensionality, in particular we substitute the original value function with an approximated representation defined over a restricted set of selected features (Bertsekas, 2012a), Section 6.1. We refer to SDP problems structured as Markov Decision Processes (MDPs), defined as follows:

- $\Xi = \{\xi^1, \dots, \xi^\Omega\}$ is the finite set of states, where $\xi^v$, $\xi^w \in \Xi$ are two generic elements of this set (note that, for the sake of generality, we have chosen a symbolic notation for the MDP states). $|\Xi| = \Omega$ is the cardinality of this set. The state variable at time $k$ is denoted with $\xi(k)$, where $\xi(k) \in \Xi$.
- $U = \{u_1, \dots, u_n\}$ is the finite set of actions (also called decisions or inputs), where $u$ is a generic element of this set. We denote the action variable at time $k$ with $u(k)$, where $u(k) \in U$. In case the set of admissible actions depends on the state $\xi^v$, we write $U(\xi^v)$.
- $\mathcal{T} \subseteq \Xi \times U \times \Xi$ is the state transition probability matrix, whose elements are

  $$p_{\xi^v \xi^w}(u) := P\left[\xi(k+1) = \xi^w \mid \xi(k) = \xi^v,\ u(k) = u\right].$$

  It represents the probability that an action $u \in U$, performed in the state $\xi^v \in \Xi$ at the time slot $k$, leads the system into the state $\xi^w \in \Xi$ at the time slot $k+1$.
- $R : \Xi \times U \rightarrow [0, +\infty)$ is the reward function, obtained when taking an action $u_i$ at any state $\xi^v \in \Xi$. In this paper, we consider that reward function only depends on the state, so we can simply write $R(\xi^v)$.

By using SDP based frameworks, it is possible to formulate and solve sequential decision problems under stochastic uncertainty modeled as MDPs. At the core of such frameworks, there is the resolution of a stochastic optimization problem.

More specifically, let us define $\mu_k(\cdot) : \Xi \rightarrow U$ as the control function mapping states into controls (i.e., $\mu_k(\cdot)$ represents the decisions selected at time $k$ on the whole state space $\Xi$). Let $\pi = \{\mu_0, \dots, \mu_{\mathcal{N}-1}\}$ denote the policy, that is to say, the sequence of control functions $\mu_k(\cdot)$ applied on the whole state space $\Xi$ over the finite time horizon $\mathcal{N}$. The value function of the policy $\pi$ starting from the initial state $\xi(0)$ over the infinite time horizon can be written as follows

$$J_\pi\big(\xi(0)\big) = \lim_{\mathcal{N}\rightarrow\infty} E\left\{\sum_{k=0}^{\mathcal{N}-1} \alpha^k R\big(\xi(k)\big)\right\}, \tag{1}$$

where $E\{\cdot\}$ is the expectation operator (calculated over the sequence of states $\xi(k)$ visited by applying the policy $\pi$), and $0 < \alpha < 1$ is the discount factor. The optimal value function can be expressed as follows

$$J^*\big(\xi(0)\big) = \max_{\pi} \lim_{\mathcal{N}\rightarrow\infty} E\left[\sum_{k=0}^{\mathcal{N}-1} \alpha^k R\big(\xi(k)\big)\right]. \tag{2}$$

With a slight abuse of notation, we can define Eqs. (1) and (2) for a generic state $\xi \in \Xi$ as $J_\pi(\xi)$ and $J^*\big(\xi\big)$, respectively. Hereafter, we consider only stationary policies over an infinite time horizon (i.e., $\pi = \{\mu, \mu, \dots\}$ and $\mathcal{N} \rightarrow \infty$). We can denote the value function of stationary policies with $J_\mu(\xi)$.

For any value function $J : \Xi \rightarrow \Re$ ($\Re$ is the set of real numbers), we define the Bellman operator for policies and the Bellman optimality operator (Bertsekas, 2012a), section 1.1

$$(T_\mu J)(\xi^v) = R(\xi^v) + \alpha \sum_{\xi^w \in \Xi} p_{\xi^v \xi^w}\big(\mu(\xi^v)\big) J(\xi^w), \tag{3}$$

$$(TJ)(\xi^v) = \max_{u \in U(\xi^v)} \Big[R(\xi^v) + \alpha \sum_{\xi^w \in \Xi} p_{\xi^v \xi^w}(u) J(\xi^w)\Big]. \tag{4}$$

Both operators can be viewed as mappings operating on the function $J$ to produce other functions of states, respectively $T_\mu J$ and $TJ$. $J_\mu$ and $J^*$ are also the fixed points of such mappings, that is to say, $J_\mu = T_\mu J_\mu$ and $J^* = TJ^*$ (Bertsekas, 2012a), section 1.2 Note that the action $u$ to be chosen in (4) does not depend on the reward $R(\xi^v)$. This is not true when the reward function also depends on the action selected in the current state. However, for the sake of generality, we prefer to keep the Bellman optimality operator in its more general form in the remainder of the paper.

There exist consolidated approaches to solve SDP problems, and calculate the corresponding optimal policy. In this context, we can mention the PI algorithm. It is an iterative algorithm, in which each iteration is composed of two steps: policy evaluation and policy improvement. Such approaches solve exactly the SDP problems, but can be hardly used in real-world applications because of the above-mentioned curse of dimensionality.

Approximate Dynamic Programming (ADP) proves to offer powerful tools for addressing such scalability issues. In this paper, we focus on approximation in value space (Bertsekas, 2012a; Powell, 2011; Tsitsiklis & Van Roy, 1997). The key idea is to approximate the value function $J$ with a more compact parametric representation, that is to say, $\tilde{J}(\xi, r)$, $\tilde{J} : \Xi \times \Re^q \rightarrow \Re$, where $r \in \Re^q$ is an adjustable parameter vector with all the components $r_i$ to be "trained". The choice of the architecture is very significant for the success of the approximation approach. One possibility is to use the following linear form

$$\tilde{J}(\xi, r) = \sum_{i=1}^{q} r_i \phi_i(\xi), \tag{5}$$

where $\phi_i(\xi)$ are some known scalars that depend on the state $\xi$. For each state $\xi$, the approximate value $\tilde{J}(\xi, r)$ is the inner product $\phi(\xi)' r$ with $\phi(\xi) = \big(\phi_1(\xi), \ldots, \phi_q(\xi)\big)'$ (note that the prime sign denotes the transpose operator). We refer to $\phi(\xi)$ as the feature vector of $\xi$, and its components $\phi_i(\xi)$ as features. In other words, we approximate the original value function $J$ by adopting a feature extraction mechanism to map the original set of states onto a much smaller set of features. Such features can be regarded as functions encoding some state properties, and are defined based on the problem to be solved. The value function vector $J$ ($J \in \Re^\Omega$ with components $J(\xi^v)$) is approximated by a vector in the subspace

$$\Delta = \{\Phi r | r \in \Re^q\}, \tag{6}$$

where

$$\Phi = \begin{bmatrix} \phi_1(\xi^1) & \ldots & \phi_q(\xi^1) \\ \vdots & \vdots & \vdots \\ \phi_1(\xi^\Omega) & \ldots & \phi_q(\xi^\Omega) \end{bmatrix}$$

is the feature matrix. Note that, in general we have $q \ll \Omega$. The $q$ columns of $\Phi$ are viewed as basis functions, and $\Phi r$ as a linear combination of basis functions. The value function of a policy $J_\mu(\xi)$ and the optimal value function $J^*(\xi)$ can be approximated respectively as $\tilde{J}_\mu(\xi, r) = \phi(\xi)' r_\mu$ and $\tilde{J}^*(\xi, r) = \phi(\xi)' r^*$.

Now we recall some definitions, assumptions, and results providing the mathematical background for the LSTD and the MG-LSTD. For further investigation and deep insight regarding the importance of these assumptions (along with the convergence result of temporal difference learning approaches with linear function approximation), we refer the reader to the original work reported in Tsitsiklis and Van Roy (1997).

**Definition 2.1.** Let us denote the MDP state transition probability matrix for a specific stationary policy $\mu$ with $P \in \Re^{\Omega \times \Omega}$, its value function vector with $J_\mu \in \Re^\Omega$ (with components $J_\mu(\xi^v)$), and the reward vector with $R \in \Re^\Omega$ (with components $R(\xi^v)$). The Bellman operator for the policy $\mu$ can be written in the following matrix form

$$T_\mu J = R + \alpha P J.$$

**Assumption 1.** For each admissible stationary policy $\mu$, the underlying Markov chain is irreducible and regular. The related stochastic matrix $P$ has a steady state probability vector $\varepsilon \in \Re^\Omega_+$ with components $\varepsilon_{\xi^v} > 0$ ($v = 1, \ldots, \Omega$).

**Assumption 2.** The matrix $\Phi$ has rank $q$.

Assumption 1 implies that the underlying Markov chain has a single recurrent class and no transient states. Assumption 2 means that each vector $J$ in the subspace $\Delta$ can be represented in the form $\Phi r$ with a unique vector $r$.

In order to introduce the projected form of the Bellman operators, we use the weighted Euclidean norm of any value function vector $J \in \Re^\Omega$ with respect to the vector of positive weights $\varepsilon$

$$\|J\|_\varepsilon = \big(J' \Theta J\big)^{\frac{1}{2}}, \tag{7}$$

where $\Theta$ is the diagonal matrix with the steady state probabilities $\varepsilon_{\xi^1}, \ldots, \varepsilon_{\xi^\Omega}$ along the diagonal. Let $\Pi$ denote the projection operation of any $J \in \Re^\Omega$ onto $\Delta$ with respect to this norm. In other words, $\Pi J$ implies computing the unique vector in $\Delta$ that minimizes the following

$$\hat{r} = \arg\min_{r \in \Re^q} \|J - \Phi r\|^2_\varepsilon. \tag{8}$$

It can also be written as

$$\Pi J = \Phi \hat{r}, \tag{9}$$

with

$$\hat{r} = (\Phi' \Theta \Phi)^{-1} \Phi' \Theta J, \tag{10}$$

where $\Pi = \Phi(\Phi'\Theta\Phi)^{-1}\Phi'\Theta$. Thanks to Assumption 2, the inverse $(\Phi'\Theta\Phi)^{-1}$ exists.

By using the projection operator, we can introduce the projected Bellman operator $\Pi T_\mu J = \Pi(R + \alpha P J)$. It can be seen that, under the above-mentioned assumptions, the corresponding projected Bellman equation $\Phi r = \Pi T_\mu(\Phi r)$ has a unique fixed point (Bertsekas, 2011a). In particular, the mappings $T_\mu$ and $\Pi T_\mu$ are contractions of modulus $\alpha$ with respect to the weighted Euclidean norm (7). We denote with $\tilde{J}_\mu = \Phi r_\mu$ the fixed point of $\Pi T_\mu$, i.e. $\Phi r_\mu = \Pi T_\mu(\Phi r_\mu)$. It is worth highlighting that the projection equation has to be solved in the subspace $\Delta$ by setting a proper combination of the basis functions.

Solving the projection equation implies performing matrix–vector multiplications and inner products of size $\Omega$ when calculating $r_\mu$ (Bertsekas, 2011b). We can address such issue by using iterative approaches based on temporal difference (TD) methods and Monte Carlo simulations.

### 2.1. The matrix form for the projected Bellman equation

Let us write the projected Bellman equation $\Phi r = \Pi T_\mu(\Phi r)$ in a more explicit form. It can be shown that its solution is the vector

$\tilde{J}_\mu = \Phi r_\mu$, where the parameter vector $r_\mu$ satisfies the following orthogonality condition (Bertsekas, 2012a), section 6.3

$$\Phi'\Theta\big(\Phi r_\mu - (R + \alpha P \Phi r_\mu)\big) = 0. \tag{11}$$

Thus, the optimality condition of the projected Bellman equation can be written in the following equivalent matrix form

$$C r_\mu = y, \tag{12}$$

where

$$C = \Phi'\Theta(I - \alpha P)\Phi, \quad \text{and} \quad y = \Phi'\Theta R. \tag{13}$$

It can be potentially solved by matrix inversion, that is to say, $r_\mu = C^{-1}y$ ($C$ is invertible and positive definite in the sense that $r'Cr > 0, \forall r \neq 0$ Bertsekas, 2011b). Unlike the Bellman equation $J_\mu = T_\mu J_\mu$, the optimality condition of the projected Bellman equation has a smaller dimension ($q$ rather than $\Omega$). However, computing $C$ and $y$ requires performing high-dimensional linear algebra, e.g., inner products of size $\Omega$. Consequently, as for problems where $\Omega$ is very large, the explicit computation of $C$ and $y$ is infeasible. Furthermore, it requires the knowledge of the steady state probabilities $\varepsilon_{\xi^1}, \ldots, \varepsilon_{\xi^\Omega}$. Both these issues can be solved by using the Monte Carlo simulations and LSTD based algorithms.

## 3. The multi-trajectory greedy least-squares temporal difference algorithm

Starting from the recursive LSTD, this section describes the proposed MG-LSTD algorithm, and investigates the properties of the resulting value function approximation.

The LSTD algorithm adopts Monte Carlo simulations to train the approximate value function of a given policy in the feature space with low-dimensional linear algebra. The key idea of simulation is to express the optimality condition (see (11) for the matrix form of the projected Bellman equation) as expected value with respect to some probability distribution, and then to approximate the expected value by sampling according to such distribution (Bertsekas, 2011b). The LSTD is performed as the policy evaluation step of the Policy Iteration algorithm (Bertsekas, 2012a), section 6.3. As a result, we compute a simulation-based approximation of $r_\mu$, which is denoted with $\hat{r}_\mu$.

In the recursive LSTD approach (see Algorithm 1), $M$ samples from one long trajectory are collected by applying a fixed policy. The parameter vector $\hat{r}_s$ is renewed at each new sample along such trajectory by using the sampled version of the matrix $C$ and the vector $y$, that is to say, $C_s$ and $y_s$ (note that $s$ is the index of the $s$th sample generated by the Monte Carlo simulation, starting from $\xi(0)$). The definitions of $C_s$ and $y_s$ are given in the Algorithm 1. At the end of the long trajectory, the approximate parameter vector $\hat{r}_\mu$ is set to $\hat{r}_M$. Moreover, $\Sigma \in \mathfrak{R}_+^{q\times q}$ is selected as a symmetric positive definite matrix and $\sigma$ is a positive scalar. For more details, we refer the reader to Nedić and Bertsekas (2003) and Yu and Bertsekas (2009), where the LSTD method and its convergence proof have been deeply investigated.

The MG-LSTD (see Algorithm 2) is an exploration-enhanced recursive LSTD algorithm with the policy improvement step embedded within the LSTD iterations. It aims at calculating the optimal value function $J^*(\xi)$ approximated via the linear feature-based architecture given by (5). In other words, it calculates the vector $\hat{r}^*$ via the LSTD successive approximations of the parameter vector $\hat{r}_s$. Two main elements differentiate the MG-LSTD from the recursive LSTD:

**Algorithm 1:** Recursive Least-Squares Temporal Difference

```
• Set $s$ to 0
while $s \le M$, $s \in \{0, 1, \ldots, M\}$ do
    • Generate the next state sample $\xi(s+1)$ from the current
      state $\xi(s)$ by Monte Carlo simulation and by applying the
      policy $\mu$ under evaluation
    • Compute the corresponding feature vector $\phi\big(\xi(s+1)\big)$
    • Compute the reward of the current state $R\big(\xi(s)\big)$
    if $s = 0$ then
        Set $\hat{r}_{-1} \leftarrow \bar{r}$ (with $\bar{r}$ being a heuristic guess based on
        some intuition about the problem), $C_{-1} = 0$, and $y_{-1} = 0$
    end
    • Calculate the matrix $C_s$
      $C_s = (1 - \frac{1}{s+1})C_{s-1} + (\frac{1}{s+1})\phi\big(\xi(s)\big)\Big(\phi\big(\xi(s)\big) - \alpha\phi\big(\xi(s+1)\big)\Big)'$
    • Calculate the vector $y_s$
      $y_s = (1 - \frac{1}{s+1})y_{s-1} + \frac{1}{s+1}\phi\big(\xi(s)\big)R\big(\xi(s)\big)$
    • Having $C_s$ and $y_s$, compute the parameter vector update $\hat{r}_s$
      as follows $\hat{r}_s = \big(C_s'\Sigma^{-1}C_s + \sigma I\big)^{-1}\big(C_s'\Sigma^{-1}y_s + \sigma\hat{r}_{s-1}\big)$
    • Set $s \leftarrow s+1$, $C_{s-1} \leftarrow C_s$, and $y_{s-1} \leftarrow y_s$
end
• Set $\hat{r}_\mu = \hat{r}_M$, and $\tilde{J}_\mu = \Phi\hat{r}_\mu$ with the components
  $\tilde{J}_\mu(\xi, r) = \phi(\xi)'\hat{r}_\mu$
```

- We use multiple $M$-length simulation trajectories, each of them having an initial state chosen according to some probability distribution. In particular, we consider $Q$ trajectories, where $j \in \{1, 2, \ldots, Q\}$ represents the $j$th trajectory. With a slight abuse of notation, $\xi^j(s)$ is the state at the $s$th iteration along the $j$th trajectory, with $s \in \{0, \ldots, M\}$. We denote with $\hat{r}_s^j$ the approximate parameter vector computed up to the $s$th iteration within the $j$th trajectory. Once the $j$th trajectory is terminated, the calculated $\hat{r}_M^j$ is replaced as the initial guess for the next one. In the end, we set $\hat{r}^*$ equal to $\hat{r}_M^Q$.
- The policy improvement is embedded into the LSTD steps: given a generic current state $\xi^j(s)$ along the current Monte Carlo $j$th trajectory, we generate a set of possible next states $\xi_{u_i}^j(s+1)$, one for each admissible action $u_i \in U(\xi^j(s))$. This set is created from the state transition probabilities $p_{\xi^j(s)\xi^w}(u_i)$. For each candidate next state, the corresponding parameter vector update $\hat{r}_s^j(u_i)$ is calculated (note that the same definitions used in the recursive LSTD are applied to $C_s^j$ and $y_s^j$ (see Algorithm 2)). Then, we choose the pair (candidate next state, candidate parameter vector update) by maximizing the sampled approximate value function

$$\underset{\hat{r}_s^j(u_i),\, \xi_{u_i}^j(s+1)}{\arg\max} \Big( R\big(\xi^j(s)\big) + \alpha\phi\big(\xi_{u_i}(s+1)\big)'\hat{r}_s^j(u_i) \Big). \tag{14}$$

Note that we denote with $u^j(s)$ the control action selected in the $s$th iteration within the $j$th Monte Carlo trajectory by Eq. (14), leading the system from $\xi^j(s)$ into $\xi^j(s+1)$.

In the MG-LSTD, we have adopted a sampled version of the matrix form for the projected Bellman equation, whose matrix $C$ is invertible (Bertsekas, 2011b). However, this property may not hold for $C_s^j$ until a sufficient number of samples in the Monte Carlo simulation are acquired for its calculation. To solve such issue, a regularization term is introduced. More specifically, in each iteration along the $j$th Monte Carlo trajectory, we compute

**Algorithm 2:** Multi-trajectory Greedy LSTD (MG-LSTD)

```
• Choose the number of trajectories $Q$ and their length $M$, set j = 1 and s = 0
• Initialize the vector $\hat{r}^1_{-1} = \bar{r}$ and the initial state $\xi^1(0) = \xi^1_0$
while $j \le Q, j \in \{1, \dots, Q\}$ do
    while $s \le M$, $s \in \{0, 1, \dots, M\}$ do
        if $s = 0$ & $j = 1$ then
            • $C^1_{-1} = 0, \quad y^1_{-1} = 0, \quad \hat{r}^1_{-1} = \bar{r}$
        end
        foreach $u_i \in U(\xi^j(s))$ do
            • From the current state $\xi^j(s)$, generate a candidate next state $\xi^j_{u_i}(s+1)$ by Monte Carlo simulation and by applying the admissible control action $u_i$
            • Compute the corresponding feature vector $\phi\big(\xi^j_{u_i}(s+1)\big)$ and the reward of the current state $R\big(\xi^j(s)\big)$
            • Calculate the matrix $C^j_s(u_i)$
              $C^j_s(u_i) = (1 - \frac{1}{s+1})C^j_{s-1} + (\frac{1}{s+1})\phi\big(\xi^j(s)\big)\Big(\phi\big(\xi^j(s)\big) - \alpha\phi\big(\xi^j_{u_i}(s+1)\big)\Big)'$
            • Calculate the vector $y^j_s$ (necessary only for the first admissible control action)
              $y^j_s = (1 - \frac{1}{s+1})y^j_{s-1} + \frac{1}{s+1}\phi\big(\xi^j(s)\big)R\big(\xi^j(s)\big)$
            • Compute the candidate parameter $\hat{r}^j_s(u_i)$ as follows
              $\hat{r}^j_s(u_i) = \big(C'^j_s(u_i)\Sigma^{-1}C^j_s(u_i) + \sigma I\big)^{-1}\big(C'^j_s(u_i)\Sigma^{-1}y^j_s + \sigma\hat{r}^j_{s-1}\big)$
        end
        • Choose the pair $(\hat{r}^j_s(u_i), \xi^j(s+1))$ and the related greedy control action $u^j(s)$ by
          $\underset{\hat{r}^j_s(u_i), \xi^j_{u_i}(s+1)}{\operatorname{argmax}} \Big(R\big(\xi^j(s)\big) + \alpha\phi\big(\xi_{u_i}(s+1)\big)'\hat{r}^j_s(u_i)\Big)$
        • Set $C^j_s \leftarrow C^j_s(u^j(s))$, $\xi^j(s+1) \leftarrow \xi^j_{u^j(s)}(s+1)$, $\hat{r}^j_s \leftarrow r_s(u^j(s))$, $s \leftarrow s+1$
    end
    • Set $j \leftarrow j+1$, $C^j_{-1} = C^{j-1}_M, y^j_{-1} = y^{j-1}_M, \hat{r}^j_{-1} = \hat{r}^{j-1}_M$
    • Select the initial state $\xi^j(0) = \xi^j_0$ for the next trajectory and set $s$ to 0
end
• Set $\hat{r}^* = r^Q_M$, and $\tilde{J}^* = \Phi\hat{r}^*$ with the components $\tilde{J}^*(\xi, r) = \phi(\xi)'\hat{r}^*$
```

$\hat{r}^j_s(u)$ by solving the following least square problem

$$\min_r \Big\{ \big(y^j_s(u) - C^j_s(u)r\big)'\Sigma^{-1}\big(y^j_s(u) - C^j_s(u)r\big) + \sigma\|r - \hat{r}^j_{s-1}\|^2 \Big\}.$$

By setting the objective function gradient to 0, we have

$$\hat{r}^j_s(u) = \big(C'^j_s(u)\Sigma^{-1}C^j_s(u) + \sigma I\big)^{-1}\big(C'^j_s(u)\Sigma^{-1}y^j_s(u) + \sigma\hat{r}^j_{s-1}\big), \quad (15)$$

where the quadratic term $\sigma\|r - \hat{r}^j_{s-1}\|^2$ is known as a regularization term (here, $\|\cdot\|$ denotes the $L_2$-norm), and has the effect of biasing the estimate $\hat{r}^j_s(u)$ towards the previous parameter vector estimation $\hat{r}^j_{s-1}$. As for the first iteration, we consider the heuristic guess $\bar{r}$ for the parameter vector $\hat{r}^1_{-1}$. It is based on some intuition about the problem at hand. Moreover, the matrix $\Sigma$ and the coefficient $\sigma$ are respectively positive definite and positive (Hoffman, Lazaric, Ghavamzadeh, & Munos, 2012).

Instead of having one single long trajectory, we use multiple shorter trajectories starting from different initial states $\xi^j(0)$. Within a specific trajectory, the selection of the states to be visited is determined by the state transition probabilities $p_{\xi^j(s)\xi^w}(u_i)$ and the greedy control action maximizing Eq. (14). Fig. 1 provides a schematic representation of the next state selection process of the MG-LSTD algorithm. This way, the MG-LSTD algorithm can experience a variety of situations and progressively favor those that appear to be the best. As a result of the overall approach, a better approximation of the optimal value function (and its computed parameter vector $\hat{r}^*$) is expected.

Once all the Monte Carlo simulation trajectories have been processed, we can set $\hat{r}^*$ equal to $\hat{r}^Q_M$. Then, we can calculate the approximate optimal policy $\tilde{\mu}^*(\cdot)$ by replacing $\tilde{J}^* = \Phi\hat{r}^*$ as the terminal value function in the Bellman optimality operator

$$\tilde{\mu}^*(\xi^v) = \underset{u_i \in U(\xi^v)}{\arg\max} \Bigg[ R(\xi^v) + \alpha \sum_{\xi^w \in \Xi} p_{\xi^v\xi^w}(u_i)\phi(\xi^w)'\hat{r}^* \Bigg]. \quad (16)$$

The MG-LSTD convergence properties have been experimentally verified when solving the optimization problems for the operational scenarios presented in Section 6. It is worthwhile highlighting that we always use admissible policies when embedding the policy improvement step into its iterations, see Eq. (14). More importantly, we have to remind that using projected fixed-point approaches (such as LSTD or LSPE) for the policy evaluation within the PI algorithm iterations, and combining them with the policy improvement step may lead to oscillations. This is due to the fact that, though the projected Bellman operator can be a contraction, it lacks of monotonicity (Bertsekas, 2012a), paragraph 6.4.3. In this regard, the MG-LSTD approach can provide an alternative since a sampled version of the policy improvement has been incorporated into its iterations, and the full policy improvement is only performed at the end by using (16).

### 3.1. Properties of the MG-LSTD value function approximation

In this paragraph, we derive a relevant accuracy implication for the value function calculated by the MG-LSTD algorithm. In particular, we have exploited the results of the following theorem reported in De Farias and Van Roy (2003), Section 3.

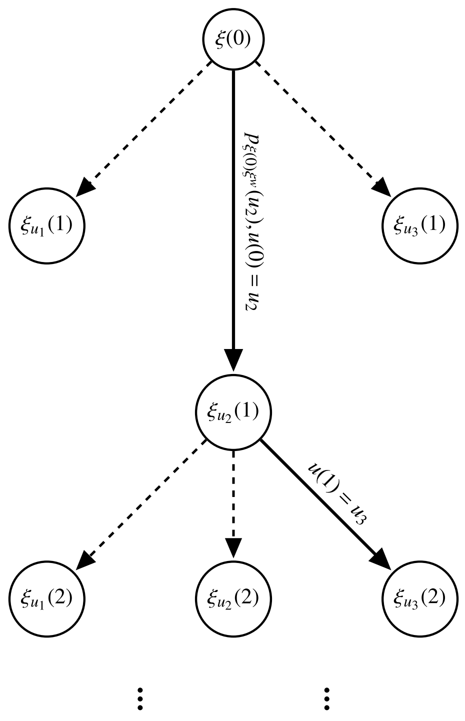

**Fig. 1.** Example of next state selection by the MG-LSTD algorithm (note that for the sake of simplicity we have removed the trajectory index $j$): starting from the initial state $\xi(0)$ with three admissible actions (i.e., $U(\xi(0)) = \{u_1, u_2, u_3\}$), we apply all the admissible actions $u_i \in U(\xi(0))$ and generate three next state candidates $\xi_{u_i}(1)$ based on $p_{\xi(0)\xi^w}(u_i)$. We calculate the corresponding $C_0(u_i)$, $y_0(u_i)$ and $\hat{r}_0(u_i)$, and then we select the state according to Eq. (14). As shown in the figure, the control action $u_2$ is chosen in the state $\xi(0)$, which brings the system into the state $\xi(1) = \xi_{u_2}(1)$.

**Theorem 3.1.** *Consider the following weighted $L_1$-norm $\| \cdot \|_{1,z}$ defined by*

$$\|J\|_{1,z} = \sum_{v=1}^{\Omega} z(\xi^v)|J(\xi^v)|,$$

*where $z \in \Re^{\Omega}$ is called state relevance weights, with positive components $z(\xi)$ and $\sum_{v=1}^{\Omega} z(\xi^v) = 1$. A vector $r$ solves*

$$\max_{r \in \Re^q} \quad z'\Phi r$$
$$s.t. \quad T\Phi r \geq \Phi r$$

*if and only if it solves*

$$\min_{r \in \Re^q} \|J^* - \Phi r\|_{1,z}$$
$$s.t. \quad T\Phi r \geq \Phi r.$$

**Proof.** It is well known that the Bellman optimality operator $T$ is monotonic and has $J^*$ as unique fixed point. Thus, it follows that, for any bounded function satisfying $J \leq TJ$, we have

$$J \leq TJ \leq T^2 J \leq \ldots \leq J^*.$$

For any parameter vector $r$ satisfying the constraint $T\Phi r \geq \Phi r$, we have that $\Phi r \leq J^*$. It follows that

$$\|J^* - \Phi r\|_{1,z} = \sum_{v=1}^{\Omega} z(\xi^v)|J^*(\xi^v) - \phi(\xi^v)'r| = z'J^* - z'\Phi r.$$

This means that maximizing $z'\Phi r$ is equivalent to minimizing $\|J^* - \Phi r\|_{1,z}$. Note that a similar conclusion can be derived in case $T\Phi r \leq \Phi r$. □

Theorem 3.1 proves a connection between the maximization of $\Phi r$ and the minimization of the term $\|J^* - \Phi r\|_{1,z}$. In the MG-LSTD algorithm, we choose the control action at each iteration according to Eq. (14), which is a sampled version of the maximization problem of Theorem 3.1. This way, we can reduce the distance to $J^*$ as the MG-LSTD algorithm collects more and more samples along the different trajectories and Eq. (14) is applied. Moreover, the choice of the state relevance weights bears a significant impact on the quality of the value function approximation. In the LSTD approach, the parameter vector $r$ is computed by Monte Carlo simulation over a long trajectory by applying a specific policy. In this regard, the state relevant weights correspond to the probability of being at a specific state when time goes to infinity and the policy under evaluation is applied (Bertsekas, 2012a), section 6.3. As for the MG-LSTD algorithm, the definition of the Monte Carlo simulation paths (and the explored states) is based on Eq. (14), thus the proposed algorithm tends to visit the states that have more value compared to the others (on a sampled basis). Therefore, the state relevance weights can be linked to the MDP steady state probability vector under the policy calculated by Eq. (14), which can converge to a specific policy as time goes by and a steady approximation $\hat{r}^*$ to the parameter vector $r^*$ is achieved.

## 3.2. Connection of the MG-LSTD with the LSPE

Now we explore some interesting links of the MG-LSTD with the other relevant method of least-squares value function approximation, that is to say, the LSPE. Without loss of generality, we apply the MG-LSTD with the number of trajectories $Q = 1$, and thus we remove the dependency from the current Monte Carlo trajectory $j$.

To start with, we define $\xi(s+1)$ as the state generated along the Monte Carlo trajectory from the current state $\xi(s)$ by applying the $s$th iteration of the MG-LSTD algorithm (see Fig. 1 and Eq. (14)). In addition, $u(s)$ is the corresponding action and $\mu(s)$ is the resulting policy. The latter can be derived as follows. We start with an initial policy $\mu(0)$, for instance a base policy (Bertsekas, 2012a), paragraph 2.3.4. As the simulation goes on, we replace the control actions for the visited states by the ones calculated by the MG-LSTD algorithm. The existence of the steady state probability vector $\varepsilon(s)$ of the resulting $\mu(s)$ (if it is applied as a stationary policy) is guaranteed by Assumption 1.

If we knew $\varepsilon(s)$, we could write the following least squares formula to calculate the parameter vector $r_\mu(s)$ of the policy $\mu(s)$ as follows (note that with $\varepsilon_{\xi^v}(s)$ is the component of $\varepsilon(s)$ corresponding to the state $\xi^v$)

$$r_\mu(s) = \arg\min_{r \in \Re^q} \sum_{v=1}^{\Omega} \varepsilon_{\xi^v}(s) \Bigg( \phi(\xi^v)'r - \max_{u \in U(\xi^v)} \Big( R(\xi^v) + \alpha \sum_{w=1}^{\Omega} p_{\xi^v \xi^w}(u)\phi(\xi^w)'r_\mu(s-1) \Big) \Bigg)^2.$$

Note that the definition of parameter vector $r_\mu(s)$ is different from the one given in the paragraph 2.1. It results from a weighted norm minimization problem over the complete state space, where for each state we have considered the temporal difference with the Bellman optimality operator, whose terminal value function is calculated from $r_\mu(s-1)$. By setting the objective function gradient to 0, we have

$$r_\mu(s) = \left( \sum_{v=1}^{\Omega} \varepsilon_{\xi^v}(s)\phi(\xi^v)\phi(\xi^v)' \right)^{-1}$$

$$\times\left(\sum_{v=1}^{\Omega}\varepsilon_{\xi^v}(s)\phi(\xi^v)\max_{u\in U(\xi^v)}\left(R(\xi^v)+\alpha\sum_{w=1}^{\Omega}p_{\xi^v\xi^w}(u)\phi(\xi^w)'r_\mu(s-1)\right)\right).$$

Since we do not know the steady state probability vector $\varepsilon(s)$, we have to use the experimental values calculated as the Monte Carlo simulation goes on. If we consider $\hat{\varepsilon}_{\xi^v}(s)$ and $\hat{p}_{\xi^v\xi^w}(s)$ as the empirical frequencies of being in state $\xi^v$ and of the state transition $(\xi^v, \xi^w)$ respectively observed by applying the MG-LSTD algorithm, we have the following

$$\hat{\varepsilon}_{\xi^v}(s)=\frac{\sum_{l=0}^{s}\delta\big(\xi(l)=\xi^v\big)}{s+1},$$
$$\hat{p}_{\xi^v\xi^w}(s)=\frac{\sum_{l=0}^{s}\delta\big(\xi(l)=\xi^v,\xi(l+1)=\xi^w\big)}{\sum_{l=0}^{s}\delta\big(\xi(l)=\xi^v\big)},$$

where $\delta(\cdot)$ is the indicator function, that is to say, $\delta(EV)=1$ if the boolean predicate $EV$ is true, and if not, $\delta(EV)=0$.

As a consequence, we can write

$$r_\mu(s)\approx\left(\sum_{v=1}^{\Omega}\hat{\varepsilon}_{\xi^v}(s)\phi(\xi^v)\phi(\xi^v)'\right)^{-1}$$
$$\times\left(\sum_{v=1}^{\Omega}\hat{\varepsilon}_{\xi^v}(s)\phi(\xi^v)\max_{u\in U(\xi^v)}\left(R(\xi^v)+\alpha\sum_{w=1}^{\Omega}\hat{p}_{\xi^v\xi^w}(u)\phi(\xi^w)'r_\mu(s-1)\right)\right).$$

By considering only the actual visited states, we have

$$\hat{r}_\mu(s)=\left(\sum_{l=0}^{s}\phi\big(\xi(l)\big)\phi\big(\xi(l)\big)'\right)^{-1}$$
$$\times\left(\sum_{l=0}^{s}\phi\big(\xi(l)\big)\Big(R\big(\xi(l)\big)+\alpha\phi\big(\xi(l+1)\big)'\hat{r}_\mu(s-1)\Big)\right). \quad (17)$$

Note that the *max* operator has been replaced by the sampled state transition derived from Eq. (14). This formula is exactly the one used in the LSPE, embedding the above-mentioned policy improvement (Bertsekas, 2012a), paragraph 6.3.4. In other words, the MG-LSTD algorithm can be easily converted into the MG-LSPE algorithm.

## 4. Further considerations on the MG-LSTD algorithm

In this section, we analyze the key-aspects of the MG-LSTD algorithm, its motivation, and its connections with other approaches reported in the literature. In this regard, we examine the MG-LSTD algorithm convergence properties and the role of its input parameters (e.g., the feature matrix $\Phi$) from a qualitative point of view by using the convergence results of LSTD based approaches available in the scientific literature. The study on the MG-LSTD convergence properties from an analytical point of view is regarded as future work.

### *4.1. MG-LSTD key-aspects, motivation and connection with other approaches*

The main objectives of proposing the MG-LSTD algorithm can be summarized as follows: to address real-world systems with a large state space; to learn proper policies over the complete state space; to improve the system behavior exploration capabilities.

Like all the ADP methods, the MG-LSTD algorithm can be used to address the scalability issue of real-world stochastic problems. ADP methods can be grouped into two main classes, that is to say, approximation in value space and approximation in policy space (Bertsekas, 2012a), Section 6. Among the different value function approximation methods reported in the literature (Geist & Pietquin, 2013), we can consider the least-squares projected fixed point category and its two main techniques, that is to say, the LSTD and the LSPE (Bertsekas, 2011b). Such techniques are used to evaluate the state value function of a specific policy by using Monte Carlo simulations and the sampled version of the projected Bellman equation (Bertsekas, 2011b). However, they are not suitable for stochastic optimal control applications, where we are interested in finding suitable policies to achieve specific tasks in a proper way. As a consequence, the policy improvement step of the PI algorithm becomes necessary for such applications.

In this regard, a difficulty can arise when the number of actions at each stage is very large, and thus the policy improvement step can also be computationally unfeasible. In the literature, there exist different methods to learn good decision policies from samples. They can be categorized into model-based (e.g., the $\lambda$-Policy Iteration Bertsekas, 2012b) and model-free approaches (e.g., Least Square Policy Iteration (Lagoudakis & Parr, 2003) and Monte Carlo Tree Search (MCTS) methods Browne, et al., 2012). For instance, in Silver, et al. (2017), we can find a successful application of the MCTS to the program *AlphaGo*. The MCTS algorithm can be applied to any finite length decision making processes, such as computer games. Such algorithm explores the most promising moves, by expanding the search tree based on the random sampling of the state space and the estimated value function. In Silver, et al. (2017), the value function is the probability of winning, and moves are selected by using deep neural networks. The latter are trained on a combination of supervised learning (i.e., moves from games played by human experts) and reinforcement learning (i.e., gaining experience from playing against its own clone).

There are three main differences between the MCTS approach presented in Silver, et al. (2017) and our proposed MG-LSTD algorithm. In particular, the latter can be regarded as a model-based approach and operates over an infinite time horizon. Moreover, it addresses the PI algorithm by embedding its policy improvement step within the recursive LSTD iterations in order to compute an approximation of the optimal value function. In the MG-LSTD, the policy under evaluation is renewed at each iteration by exploiting the information from the already-visited states and by using a greedy approach for the selection of the next control action along the current Monte Carlo trajectory, see Eq. (14). Unlike the recursive LSTD algorithm (see Algorithm 1), the matrix $C_s^j(u_i)$ is calculated for all the control actions admissible in the current state $\xi^j(s)$, namely, $u_i \in U(\xi^j(s))$. Then, the MG-LSTD algorithm sets the matrix $C_s^j$ to the one corresponding to the greedy control action $u^j(s)$ (see Algorithm 2).

The MG-LSTD algorithm also addresses the dilemma of exploration versus exploitation, i.e., trying out non-optimal actions to explore the system behavior versus exploiting the (so-far) best perceived actions to obtain rewards (see Arulkumaran, Deisenroth, Brundage, & Bharath, 2017, paragraph VI.B). In particular, the MG-LSTD algorithm makes useful progress when it applies Eq. (14) and exploits the (so-far) best perceived actions, while defining the path along the current Monte Carlo trajectory. On the other hand, for each admissible action $u_i$ in the current state $\xi^j(s)$ (with $u_i \in U(\xi^j(s))$), the candidate next state $\xi^j_{u_i}(s+1)$ is generated according to the MDP state transition probabilities $p_{\xi^j(s)\xi^w}(u_i)$, thus contributing to the system state space exploration (note that the candidate next state $\xi^j_{u_i}(s+1)$ is an input for the greedy action selection). Moreover, we adopt a multi-trajectory approach, which implies starting the Monte Carlo simulation from a different initial state $\xi^j(0)$ at the beginning of each $j$th trajectory. Such initial state can be chosen according to a fixed probability distribution $\gamma(0)=\big(\gamma_{\xi^1}(0),\ldots,\gamma_{\xi^v}(0),\ldots,\gamma_{\xi^\Omega}(0)\big)$, with $\gamma_{\xi^v}(0)=Pr\big(\xi^j(0)=\xi^v\big)$ and $Pr(\cdot)$ denoting any probability mass function over the state space.

It is also worth highlighting that, along each MG-LSTD trajectory, we preserve the information from the already-visited states via the matrix $C_s^j$, the vector $y_s^j$, and the regularization term (see Algorithm 2). In addition, the initial conditions of the next trajectory are set according to the results calculated in the previous one (e.g., $\hat{r}_{-1}^j = \hat{r}_M^{j-1}$).

## 4.2. Literature review on the convergence property of the LSTD based approaches

Temporal Difference (TD) learning is a widely applied and highly successful approach for approximating the value function associated to a certain policy $\pi$ on a Markov Decision Process $\langle \Xi, U, \mathcal{T}, R \rangle$. It combines Monte Carlo simulations and Dynamic Programming (Sutton & Barto, 2018), chapter 6. LSTD based approaches belong to the family of TD learning methods (Bertsekas, 2012a), section 6.3.

As proven in Tsitsiklis and Van Roy (1997), for linear function approximation, the convergence of TD learning based methods is guaranteed if the states are sampled according to the frequencies natural to the underlying Markov Chain, e.g., the steady-state probability distribution of Assumption 1. In other words, samples consist in a sequence of the actual visited states obtained either through simulation of the underlying Markov chain or observation of the actual physical system (Bertsekas, 2012a), section 6.3. In this case, an on-policy reinforcement learning scheme is considered (Sutton & Barto, 2018), section 5.5.

Conversely, TD learning based methods can diverge when states are sampled according to some probability distributions different from the underlying Markov chain dynamics (Tsitsiklis & Van Roy, 1997). Such approach is called off-policy in the reinforcement learning. This means modifying the state transition probability matrix of a given policy $\mu$ by occasionally generating transitions other than the ones dictated by $\mu$ (Bertsekas, 2012a), paragraph 6.4.2. The distribution of updates in the off-policy case is not according to the on-policy distribution (Sutton & Barto, 2018), section 5.7. As a result, the convergence of the LSTD based approaches cannot be guaranteed in case we do not sample states with the frequencies natural to the underlying Markov chain (see the examples provided by the papers referenced in Tsitsiklis and Van Roy (1997), section IX). In such a case, the convergence depends on the selection of the feature matrix $\Phi$, the stationary probability distribution of the policy under evaluation, the discount factor as well as the used offline probability distribution (see Theorem 3 reported in Tsitsiklis & Van Roy, 1997).

The MG-LSTD algorithm starts each episode (or trajectory) from a feasible initial state, while holding the previous values of $C_s^j$, $\hat{r}_s^j$ and $y_s^j$ as the initial setting of the upcoming simulation. This approach does not violate the convergence result presented in Tsitsiklis and Van Roy (1997) since, in the LSTD based methods, there is no assumption about the choice of a specific initial setting at the beginning of the Monte Carlo simulation (Bertsekas, 2019b), section 4.10. The MG-LSTD algorithm should be regarded as an on-policy approach since, during each episode, the state visits are selected according to the frequencies natural to the underlying Markov chain: thus, its convergence should be guaranteed. In this regard, an analysis of LSTD method convergence under general conditions can be found in Yu (2012), where the convergence properties of enhanced-exploration methods are investigated. In particular, for the discounted cost criterion, the convergence can be guaranteed for the enhanced-exploration approaches under mild and minimal conditions.

Moreover, as for multi-trajectory PI approaches using LSTD for the policy evaluation step, it is important to mention the convergence study of the multi-trajectory $\lambda-$ PI algorithm reported in Bertsekas (2012b) (in particular, see Section 4.3). The main common aspects between such algorithm and the MG-LSTD approach is the usage of multiple short trajectories with an exploration-enhanced restart, rather than a single infinitely long trajectory. However, there are some important differences. Besides the greedy approach for selection of the next control action within the LSTD iterations, the MG-LSTD algorithm uses trajectories with a fixed length (rather than trajectories with random geometrically distributed length) and single step simulations (rather than multi-step simulations).

Like all the LSTD methods, based on the results presented in Bertsekas (2011a), its asymptotic convergence rate should not be affected by the defined input parameters (e.g., the feature matrix), but instead it should only depend on the choice of the state sampling mechanisms. It means that the convergence rate of the LSTD algorithm (where we evaluate a specific policy) is expected to be different from the one of the MG-LSTD algorithm (where the policy under evaluation is renewed at each iteration, see Eq. (14)): the two state sampling mechanisms, even though based on the underlying system dynamics, are different. Thus, the MG-LSTD convergence rate is expected to be slower than the one of the single trajectory LSTD, implemented according to the Algorithm 1.

## 4.3. Feature selection, computational cost, and input parameter definition

The selection of a suitable feature matrix $\Phi$ for the problem at hand is an active area of research, and plays an important role in the computational complexity of LSTD based methods and the quality of the computed parameter vector (Bertsekas, 2011a; Bertsekas & Yub, 2009). For instance, in Keller, Mannor, and Precup (2006), a method that uses Neighborhood Component Analysis (NCA) to select new features to be added into the feature matrix is proposed. More specifically, sampled trajectories are used to generate the approximate Bellman error for each state. By using NCA, a transformation is learned that maps states with similar Bellman errors together, which is then used to select features for the new basis functions to be used in the next iteration. The computational cost of LSTD based algorithms increases with the number of features. Some approaches can be applied, such as the incremental LSTD methods as reported in Gehring, Pan, and White (2016) and Geramifard, Bowling, and Sutton (2006). In this manuscript, the MG-LSTD algorithm is adopted to solve resource allocation problems formulated via a constrained SDP based framework. The feature matrix $\Phi$ is a given input parameter to the problem to be solved, and the investigation on how the feature selection process can affect the MG-LSTD algorithm solutions goes beyond the scope of this paper.

It is also worth noting that the sample computational complexity of both the recursive LSTD and MG-LSTD algorithms is mainly driven by a matrix operation inversion. Unlike the recursive LSTD, in the MG-LSTD, we need to execute a $q \times q$ matrix inversion ($q$ is the number of features) in order to calculate the candidate parameter vector $\hat{r}_s^j(u_i)$ for each admissible action $u_i$ in the current state $\xi^j(s)$. This computational cost can be reduced from $O(q^3)$ to $O(q^2)$ by using the Sherman–Morrison formula (Dann, Neumann, & Peters, 2014).

As discussed in the previous paragraph, as long as we sample according to the underlying system dynamics, the asymptotic convergence rate of LSTD based approaches is not affected by the defined input parameters (e.g., the feature matrix), but instead it depends on the choice of the state sampling mechanisms (Bertsekas, 2011a). The feature matrix definition can affect the quality of the value function approximation (Bertsekas, 2011a). This is not the case of the other input parameters, such as the matrix $\Sigma$ (as for the latter, provided that it is positive definite and symmetric Bertsekas, 2011a; Nedić & Bertsekas, 2003; Yu & Bertsekas, 2009). Such matrix is used to reduce the sensitivity to the amount of sampling required for reliable simulation-based matrix inversion: in particular, it can happen that the sampled version $C_s^j$ of the matrix form for the projected Bellman equation is not invertible at the beginning of the MG-LSTD algorithm (Bertsekas, 2011a). The parameter $\sigma$ has the effect of biasing the estimate $\hat{r}_s^j$ towards the heuristic initial guess $\bar{r}$. An inappropriate initial guess should not be an issue since the resulting bias tends to vanish as the Monte Carlo simulations over the different trajectories go on. In the end, both $\sigma$ and $\Sigma$ can be defined according to the specific needs of the problem at hand (e.g., $\Sigma$ can be set to the identity matrix if the matrix $C_s^j$ is always invertible) and should not affect the convergence properties of the algorithm (Bertsekas, 2011a; Nedić & Bertsekas, 2003; Yu & Bertsekas, 2009).

As for the MG-LSTD, another relevant aspect is the choice of the number of trajectories $Q$ and their length $M$. A trade-off among the MG-LSTD computational cost, its system state exploration capability for systems with a large state space, and the quality of the computed parameter vector has to be considered. Having a low number of trajectories with a high value of $M$ is not a good choice. Indeed, $M$ should be set in a way to make a significant progress in the calculation of the parameter vector over a specific trajectory, and $Q$ has to be chosen in order to guarantee the convergence of the final computed parameter vector.

The MG-LSTD algorithm could also be applied for continuous state/action space problems (Hasselt & Wiering, 2007). Often, the related systems can be converted into discrete systems, either because the application itself is characterized by discrete state values and actions (in such a case, only the time variable has to be discretized) or because the overall continuous decision problem can be approximated via quantizing also the state and the action spaces.

In the remaining part of the paper, we show how to apply the MG-LSTD algorithm to resource allocation problems modeled via an SDP based framework. This way, we perform a numerical investigation of its convergence properties, analyze the effect of the MG-LSTD algorithm input parameters on the computed parameter vector, and compare the proposed MG-LSTD algorithm with the recursive LSTD approach.

## 5. Formulating resource allocation problems as a constrained SDP framework

This section shows how to structure resource allocation problems as a set of parallel BPDs. The resulting constrained SDP formulation allows us to calculate a proper pricing policy by applying ADP techniques, that is to say, the MG-LSTD algorithm. Such formulation was presented and analyzed by the authors in Forootani, Tipaldi, Ghaniee Zarch, Liuzza, and Glielmo (2019). More specifically, a discretization process was applied to model resource allocation problems as a set of discrete-time BDPs, and then integrated into one Markov decision process. Since the application itself is characterized by discrete state values and actions, it was only necessary to perform the discretization of the time variable. Hereafter, the main aspects of such constrained SDP framework are outlined. For more details, the reader can refer to Forootani, Iervolino, and Tipaldi (2019) and Forootani, Tipaldi, Ghaniee Zarch, Liuzza, and Glielmo (2019).

We consider the problem of dynamically pricing $N$ resources in order to address resource requests from customers over time. We have a set of $m$ hourly prices (or prices per unit of time), and price managers can select one of them in order to maximize the expected total revenue. There is a dedicated BDP for each feasible price in order to model the unpredictable behavior of customers in requesting and releasing resources. The state of each BDP corresponds to the number of customers holding resources at that price. Price managers can charge different prices for the same resource over time depending on the resource availability and expected profit. In our model, we assume that all the resources are equivalent and that customers always accept the proposed price, whereas price managers can reject a resource request if not convenient from a profit standpoint. Therefore, the system evolves as a set of parallel BDPs. By assigning a specific price at each time slot, the price manager defines which BDP is active for one (possible) birth and one (possible) death, whereas all the others are active only for one (possible) death (we assume that only one customer associated to each BDP may leave at any time). This way, we can establish the decision making process by integrating all the BDPs into one MDP.

Having said that, we can proceed by providing the mathematical formulation of the resulting SDP framework.

**Definition 5.1.** The constrained MDP *associated to each price* $c_i$ is defined by a tuple $\mathcal{C}_i = \langle \Xi_i, U_i, \mathcal{T}_i, R_i \rangle$ where

- $\Xi_i$ is the state space, $\Xi_i = \{\xi_i^0, \xi_i^1, \ldots, \xi_i^N\}$. The state variable at time $k$ is denoted with $\xi_i(k)$, where $\xi_i(k) \in \Xi_i$. In particular, $\xi_i(k) = \xi_i^{h_i}$ means that $h_i$ resources are allocated at current time $k$ at price $c_i$. Moreover, we denote by $\xi_i^{v_i}$ and $\xi_i^{w_i}$ two generic states of the process.
- The maximum number of allocable resources is assumed to be finite and equal to $N$, and we define $|\xi_i^{h_i}| = h_i \le N$.
- $U_i$ is a finite set of control actions (also called decisions or inputs), defined as $U_i = \{c_i, \psi\}$, where $c_i$ represents "allocation" with price $c_i$ and $\psi$ denotes the action of "rejection". It is the action space of the MDP $\mathcal{C}_i$. With a slight abuse of notation, we denote with $U_i(\xi_i)$ the set of inputs admissible at state $\xi_i$. Hence, for each state $\xi_i^{h_i}$, with $h_i = 0, 1, \ldots, N$, we have
$$\begin{cases} U_i(\xi_i) = \{c_i, \psi\}, & \text{if } |\xi_i| < N, \\ U_i(\xi_i) = \{\psi\}, & \text{if } |\xi_i| = N. \end{cases} \tag{18}$$
The input variable at time $k$ is denoted as $u_i(k) \in U_i(\xi_i(k))$.
- $\mathcal{T}_i \subseteq \Xi_i \times U_i \times \Xi_i$ is the state transition probability matrix with elements
$$p_{\xi_i^{v_i}\xi_i^{w_i}}(u_i(k)) := P\big(\xi_i(k+1) = \xi_i^{w_i} | \xi_i(k) = \xi_i^{v_i}, u_i(k)\big), \quad s.t.\ v_i - 1 \le w_i \le v_i + 1. \tag{19}$$
Here, the state transition probabilities can be calculated by means of the birth and death rates. In the other cases, the transition probabilities are simply set to zero.
- $R_i : \Xi_i \to [0, +\infty)$ is the reward function. In our case
$$R_i(\xi_i) = c_i |\xi_i^{h_i}| = c_i h_i. \tag{20}$$

Note that, for the problem considered, the reward depends only on the state. We can denote $\xi_i := \xi_i^{h_i}$ for a specific price, and we can drop the norm notation and write directly $\xi_i$ when it is clear from the context.

Fig. 2 shows the state transition probabilities of the MDP $\mathcal{C}_i$ according to the admissible control actions. By applying the decision $u_i = c_i$, the resulting Markov chain $\mathcal{C}_i$ allows a birth transition from the current state with probability $\lambda_i$ (it is the BDP birth rate, and represents the probability of a customer requiring a resource at hourly price $c_i$), a death transition with probability $\mu_i$ (it is the BDP death rate, and represents the probability of a customer releasing a resource previously purchased at hourly price $c_i$) and a self-transition with probability $1 - \lambda_i - \mu_i$ (representing the fact that no customer releases or asks for a resource). On the other hand, when the decision $u_i = \psi$ is taken, no customer can

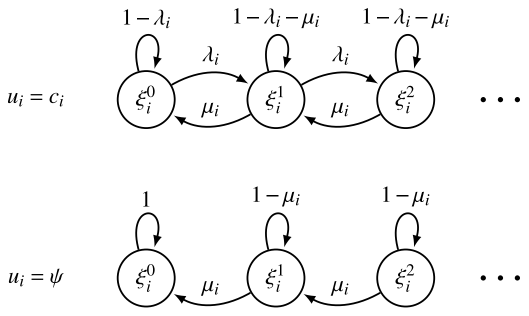

**Fig. 2.** The MDP $\mathcal{C}_i$. State transitions allowed with the input $u_i = c_i$ are depicted in the top. State transitions allowed with the input $u_i = \psi$ are depicted in the bottom.

purchase the resource at price $c_i$, and only a death transition or a self-transition from the current state are allowed.

The "composition" of $m$ different $\mathcal{C}_i$ corresponding to each price and the common constraint of having $N$ available resources give rise to the overall system $\mathcal{C}$.

**Definition 5.2.** The constrained MDP of the overall system is defined by a tuple $\mathcal{C} = \langle \Xi, U, \mathcal{T}, R\rangle$ where

- $\Xi$ is the entire state space, $\Xi = \left\{\xi^h = (\xi_1^{h_1}, \dots, \xi_m^{h_m}) \in \times_{i=1}^m \Xi_i \;:\; \|\xi^h\|_1 \leq N,\; \|\xi^h\|_1 = \sum_{i=1}^m h_i\right\}$ (here, $\|\cdot\|_1$ denotes the $L_1$-norm). We show with $\xi(k)$ the state at time $k$. Moreover, we denote by $\xi^v = (\xi_1^{v_1}, \xi_2^{v_2}, \dots, \xi_m^{v_m})$ and $\xi^w = (\xi_1^{w_1}, \xi_2^{w_2}, \dots, \xi_m^{w_m})$ two generic state of the MDP. To simplify the complexity in the notation, we denote $\xi := \xi^h$ as a generic state of the process. We recall that $\Omega$ is the cardinality of the set $\Xi$.
- $U = \{c_1, \dots, c_m, \psi\}$ is the overall input set. With a slight abuse of notation, we denote with $U(\xi)$ the set of admissible inputs at any state $\xi \in \Xi$. We have
$$\begin{cases} U(\xi) = \{c_1, \dots, c_m, \psi\}, & \text{if } \|\xi\|_1 < N, \\ U(\xi) = \{\psi\}, & \text{if } \|\xi\|_1 = N. \end{cases} \tag{21}$$
- $\mathcal{T} \subseteq \Xi \times U \times \Xi$ is the state transition probability matrix, whose elements are
$$p_{\xi^v\xi^w}\big(u(k)\big) := p\big[\xi(k+1) = \xi^w | \xi(k) = \xi^v, u(k)\big]. \tag{22}$$
In case $u(k) = c_i$ we have
$$p_{\xi^v\xi^w}(c_i) = p_{\xi_i^{v_i}\xi_i^{w_i}}(c_i) \cdot \prod_{\substack{j=1\\ j\neq i}}^m p_{\xi_j^{v_j}\xi_j^{w_j}}(\psi), \tag{23}$$
$$\begin{aligned} s.t.\ & v_i - 1 \leq w_i \leq v_i + 1, \\ & v_j - 1 \leq w_j \leq v_j, \end{aligned}$$
while if $u(k) = \psi$
$$p_{\xi^v\xi^w}(\psi) = \prod_{j=1}^m p_{\xi_j^{v_j}\xi_j^{w_j}}(\psi), \tag{24}$$
$$s.t.\ v_j - 1 \leq w_j \leq v_j.$$
We can also highlight the following aspects for the transition from the state $\xi^v$ into the state $\xi^w$
$$\max \|\xi^w\|_1 = \min\left(\|\xi^v\|_1 + 1, N\right), \tag{25}$$
$$\min \|\xi^w\|_1 = \max\left(0, \|\xi^v\|_1 - m\right). \tag{26}$$
- $R : \Xi \to [0, +\infty)$ is the reward function, defined in this case as the summation of hourly prices paid for the various resources
$$R(\xi) = \sum_{i=1}^m c_i|\xi_i| = \sum_{i=1}^m c_i h_i. \tag{27}$$

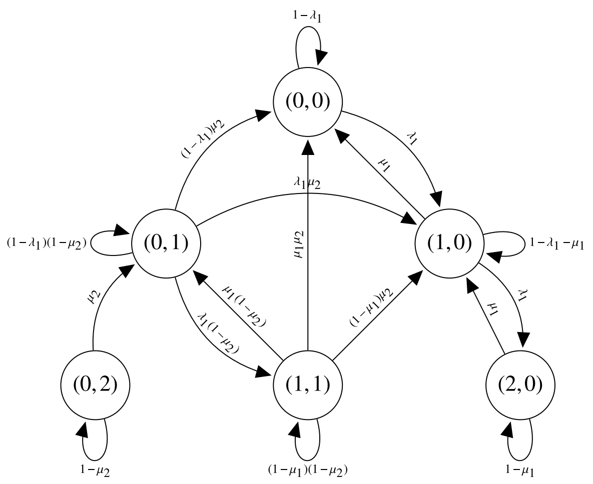

**Fig. 3.** The graph shows the MDP state space and the corresponding state transition probabilities for a resource allocation problem with $N = 2$, $m = 2$, and for the control input $u(k) = c_1$. The system states are $(\xi_1, \xi_2) \in \{(0,0), (0,1), (1,0), (1,1), (0,2), (2,0)\}$, where $\xi_1$ and $\xi_2$ are associated to $c_1$ and $c_2$, respectively. As for the states where $\{(2,0), (1,1), (0,2)\}$, the only admissible control input is $u = \psi$.

Note that, as for the case of the single $\mathcal{C}_i$, the overall reward depends only on the state. Fig. 3 shows the MDP state space and the state transition probabilities for a resource allocation problem with $N = 2$ and the action space $U = \{c_1, c_2, \psi\}$. It is worth highlighting that for the states where $\|\xi\|_1 = N$, the only admissible control input is $\psi$, while for the other states the control input $c_1$ is applied.

### 5.1. Resource allocation problem resolution via an ADP approach

The presented model allows us to structure resource allocation problems via a constrained SDP formulation. In particular, we aim at finding the optimal control that maximizes the expected total revenue over an infinite time horizon. More formally, we address the following objective function ($\pi$ is a stationary policy)

$$J^*\big(\xi(0)\big) = \max_{\pi} \lim_{\mathcal{N}\to\infty} E\left[\sum_{k=0}^{\mathcal{N}-1} \alpha^k \sum_{i=1}^m c_i\xi_i(k)\right]. \tag{28}$$

We adopt a linear feature-based architecture to approximate the value function and calculate a sub-optimal stationary policy. In particular, the value function $J(\xi)$ can be approximated as $\tilde{J}(\xi, r) = \phi(\xi)'r$, where $\phi(\xi) = [\phi_0(\xi), \phi_1(\xi), \dots, \phi_q(\xi)]'$ is the vector of feature (or basis) functions evaluated over $\xi$, and $r = (r_0, r_1, \dots, r_q)'$ is a vector of $q+1$ parameters to be tuned by training the selected architecture.

In this paper, for each state $\xi = (\xi_1, \xi_2, \dots, \xi_m)$, we define the following feature functions

$$\phi_0(\xi) = 1, \quad \phi_i(\xi) = h_i, \quad i = 1, \dots, m. \tag{29}$$

In other words, the number of features is equal to $m+1$ and the features are defined as a bias constant plus the number of resources allocated at each price $c_i$. Such feature selection arises from the reward function definition (27). In the next section, we train the selected architecture by means of the MG-LSTD algorithm to compute a suitable $\hat{r}^*$ to approximate $J^*(\xi)$ as $\tilde{J}^*(\xi, \hat{r}^*) = \phi(\xi)'\hat{r}^*$.

**Table 1**
State space cardinality with different number of resources $N$ and prices $m$.

| m | N | Number of states |
|---|---|---|
| 2 | 10 | 66 |
| 3 | 20 | 1771 |
| 5 | 20 | 53 130 |
| 6 | 20 | 230 230 |
| 4 | 50 | 316 251 |
| 5 | 50 | 347 8761 |

## 6. Simulation results

In this section, we define some operational scenarios for resource allocation problems. Sub-optimal stationary value functions and policies are calculated over an infinite time horizon by means of the MG-LSTD algorithm. We also provide some experimental evidence on the MG-LSTD algorithm convergence properties in function of its key-parameters and some guidance on its proper usage.

A Matlab based application has been developed to construct the system state space, to define the state transition probability matrix concerning the parallel BDPs, to solve the optimization problem via the MG-LSTD algorithm, to analyze the related results, as well as to perform Monte Carlo simulations for evaluating the calculated pricing policies. In this respect, such application takes as input parameters all the data necessary to set up the MDP formulation and the related SDP framework, e.g., the birth and death rates associated to each price, the number of resources, and the discount factor.

Table 1 shows how the number of allocable resources $N$ and admissible prices $m$ can affect exponentially the cardinality $\Omega$ of the MDP state space $\Xi$. The usage of exact DP algorithms, such as PI, is likely to become infeasible in case of realistic problems, thus we have to apply ADP based techniques.

**Example 6.1.** Number of prices $m = 3$, number of resources $N = 4$, $\alpha = 0.996$, prices $c = [0.9\ 1\ 1.1]$, birth rates $\lambda = [0.6\ 0.5\ 0.3]$, death rates $\mu = [0.2\ 0.2\ 0.4]$, $\sigma = 0.01$, and $\Sigma$ is the identity matrix.

Even though the SDP problem of this example is computationally tractable (the state space cardinality is 35), we have applied the MG-LSTD algorithm with $Q = 200$ trajectories of $M = 1000$ iterations in length. Linear basis functions of the form (29) have been considered to approximate the value function. Hence, $r$ is a $4 \times 1$ dimensional vector. The calculated parameter vector of the MG-LSTD algorithm is $\hat{r}^* = [725.63\ \ 3.84\ \ 5.02\ \ 2.55]'$.

In this example, by applying a Monte Carlo simulation based approach for policy evaluation (see Forootani, Tipaldi, Ghaniee Zarch, Liuzza, & Glielmo, 2019 for more details), we compare the MG-LSTD pricing policy calculated by means of (16) against the following non-optimal policies:

- Non-optimal policy 1: if $\mathrm{mod}(k, 3) = 2$ choose price $c_1$, if the time instant $k$ is divisible by 3 choose price $c_2$, if $\mathrm{mod}(k, 3) = 1$ choose price $c_3$.
- Non-optimal policy 2: choose a price randomly.
- Non-optimal policy 3: choose price $c_1$.
- Non-optimal policy 4: if $\|\xi(k)\|_1 \leq 2$ then choose price $c_1$, if $\|\xi(k)\|_1 = 3$ then choose price $c_2$.

In particular, 100 randomly generated experiments are carried out by running all the addressed policies. Their long-term revenues, calculated by summing the revenues of all the visited states (plus the terminal value function for the last visited state), are shown in Fig. 4. Note that we have used $\tilde{J}^*(\xi, r) = \phi(\xi)'\hat{r}^*$

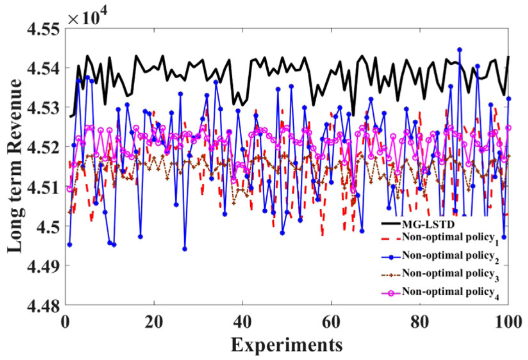

**Fig. 4.** Example 6.1: long-term revenues of 100 experiments when applying the MG-LSTD pricing policy and the non-optimal ones.

**Table 2**
Example 6.1: MG-LSTD vs. non-optimal policies.

| Policy | Revenue mean value |
|---|---|
| MG-LSTD | 45 378 |
| Non-optimal policy 1 | 45 138 |
| Non-optimal policy 2 | 45 176 |
| Non-optimal policy 3 | 45 144 |
| Non-optimal policy 4 | 45 210 |

as terminal value function in all the cases. As shown in Table 2, the long-term revenue mean value of the MG-LSTD policy is slightly greater than the ones obtained by applying the non-optimal policies. Fig. 4 would have displayed a more marked difference in favor of the MG-LSTD pricing policy if we had applied, as terminal value function, the approximate value function of the non-optimal policies in the computation of their long-term revenue. This aspect is further investigated in the next example.

**Example 6.2.** The same set-up of Example 6.1.

In this example, we have computed the parameter vector $\hat{r}_\pi$ of the non-optimal policies defined in Example 6.1 by using the recursive LSTD approach. By applying the Algorithm 1, the following parameter vectors have been calculated:

- $\hat{r}_{\pi_1} = [661.47\ \ 6.70\ \ 6.30\ \ 2.86]'$,
- $\hat{r}_{\pi_2} = [682.23\ \ 6.39\ \ 6.80\ \ 2.28]'$,
- $\hat{r}_{\pi_3} = [678.30\ \ 6.02\ \ 3.11\ \ 2.07]'$,
- $\hat{r}_{\pi_4} = [606.29\ \ 7.95\ \ 2.44\ \ 1.27]'$,

where, e.g., $\hat{r}_{\pi_1}$ denotes the parameter vector associated to the non-optimal policy 1.

Contrary to the previous example, the approximate value functions $\tilde{J}_\mu(\xi, r) = \phi(\xi)'\hat{r}_\mu$ of the non-optimal policies have been regarded as terminal value functions in the calculation of their long-term revenues for 100 randomly generated experiments. As for the MG-LSTD pricing policy, the approach described in the previous example is applied. In this example, we compare the proposed MG-LSTD approach with the recursive LSTD algorithm in terms of long term revenues. As shown in Fig. 5, the computed MG-LSTD policy outperforms significantly the other non-optimal policies.

**Example 6.3.** Number of prices $m = 4$, number of resources $N = 50$, $\alpha = 0.996$, prices $c = [0.9\ \ 1\ \ 1.1\ \ 1.2]$, birth rates $\lambda = [0.6\ 0.5\ 0.3\ 0.2]$, death rates $\mu = [0.2\ 0.2\ 0.4\ 0.4]$, $\sigma = 0.01$, and $\Sigma$ is the identity matrix.

The cardinality of the state space is 316 251, thus exact DP methods cannot be applied. In this example, we have used $Q =$

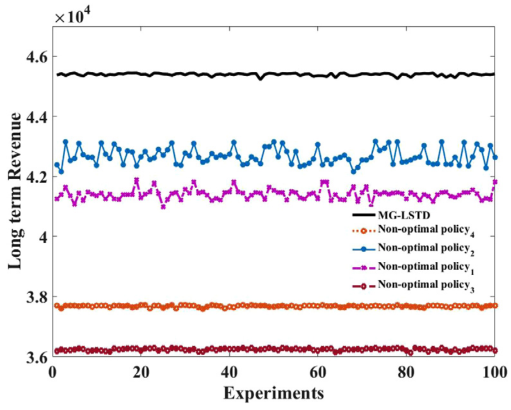

**Fig. 5.** Example 6.2: MG-LSTD algorithm vs. the recursive LSTD approach.

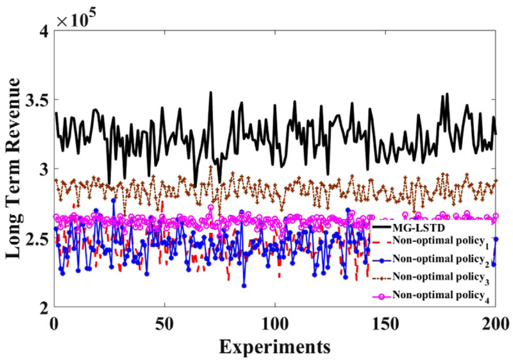

**Fig. 6.** Example 6.3: long-term revenues of 200 experiments when applying the MG-LSTD pricing policy and the non-optimal ones.

1000 trajectories of $M = 20\,000$ iterations in length to apply the MG-LSTD algorithm. As before, linear basis functions of the form (29) have been considered to represent the approximation subspace. The calculated parameter vector is $\hat{r}^* = [3735.7 \quad 12.52 \quad 2.64 \quad 1.6 \quad 8.36]'$.

As done before, we have compared the MG-LSTD pricing policy calculated via (16) against the following non-optimal policies:

- Non-optimal policy 1: if $\text{mod}(k, 4) = 2$ choose price $c_1$, if the time instant $k$ is divisible by 4 choose price $c_2$, if $\text{mod}(k, 4) = 1$ choose price $c_3$, if $\text{mod}(k, 4) = 3$ choose price $c_4$.
- Non-optimal policy 2: choose a price randomly.
- Non-optimal policy 3: choose price $c_1$.
- Non-optimal policy 4: if $\|\xi(k)\|_1 \le 25$ then choose price $c_1$, if $\|\xi(k)\|_1 = 35$ then choose price $c_2$, otherwise choose price $c_3$.

We have carried out 200 experiments. We have used $\tilde{J}^*(\xi, r) = \phi(\xi)'\hat{r}^*$ and $\tilde{J}_\mu(\xi, r) = \phi(\xi)'\hat{r}_\mu$ as the terminal value functions for the MG-LSTD pricing policy and the non-optimal ones, respectively. Their long-term revenues have been calculated and are shown in Fig. 6. The long-term revenue mean value achieved by the MG-LSTD is greater than the ones achieved by the other non-optimal policies (see Table 3).

In Fig. 7 we show the number of customers holding resources at price $c_2$ over the temporal horizon of 100 time slots for a specific execution of the MG-LSTD pricing policy and the non-optimal ones. As we can see, the MG-LSTD policy tends to choose the price $c_2$ more frequently than the other policies. Fig. 8 shows the number of customers holding resources at price $c_3$. As we can see, the MG-LSTD pricing policy never selects the policy $c_3$ over the horizon (this is due to the fact that $\lambda_3 < \mu_3$, thus customers are very likely to release resources purchased at price $c_3$). Notice also that the non-optimal policy 4 tends to choose the price $c_3$ more frequently than the other policies. This can be simply explained by the actual number of resources allocated over time (see the definition of the non-optimal policy 4).

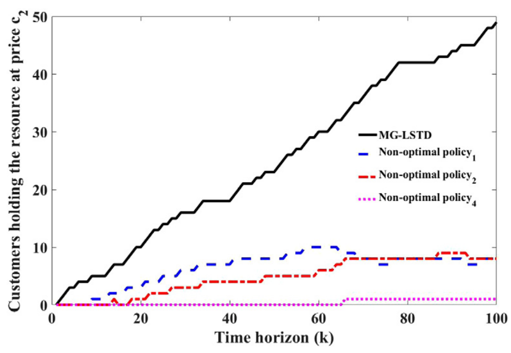

**Fig. 7.** Example 6.3: number of customers holding resources at price $c_2$ when applying the MG-LSTD and the non-optimal policies.

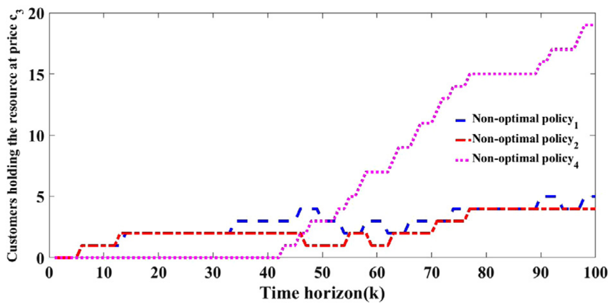

**Fig. 8.** Example 6.3: number of customers holding resources at price $c_3$ when applying the MG-LSTD and the non-optimal policies.

**Table 3**
Example 6.3: MG-LSTD vs. non-optimal policies.

| Policy | Revenue mean value |
|---|---|
| MG-LSTD | 322 070 |
| Non-optimal policy 1 | 245 250 |
| Non-optimal policy 2 | 245 220 |
| Non-optimal policy 3 | 283 880 |
| Non-optimal policy 4 | 261 510 |

**Example 6.4.** The same set-up of Example 6.3.

We compare the MG-LSTD pricing policy with a non-optimal, but slightly more appropriate policy:

- Choose a random price consistently with its death rate, i.e., prices $c_1$ and $c_2$ are more likely to be chosen than $c_3$.

In this example, we have used $Q = 1000$ trajectories of $M = 20\,000$ iterations in length to apply the MG-LSTD algorithm. The calculated parameter vector $\hat{r}^*$ is the same of Example 6.3. The outcome of this example is shown in Fig. 9, where the long-term revenues of 200 experiments of the two different policies are plotted. Also in this case, the MG-LSTD pricing policy achieves a greater long-term revenue mean value than the non-optimal policy.

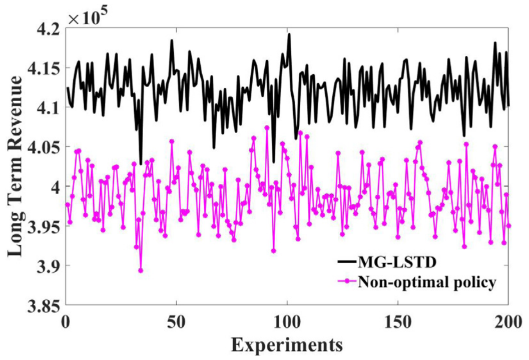

**Fig. 9.** Example 6.4: long-term revenues of 200 experiments when applying the MG-LSTD pricing policy and the non-optimal one.

### 6.1. Experimental evidence on the MG-LSTD convergence and the role of its input parameters

In this paragraph, we provide some numerical evidence on the convergence properties of the proposed MG-LSTD algorithm. To this aim, we verify such convergence in case of different values of the MG-LSTD algorithm input parameters (e.g., the matrix $\Sigma$). Moreover, being an LSTD based approach, the effectiveness of the MG-LSTD algorithm can rely on the model accuracy (in our case, the birth/death probability values $\lambda_i$'s and $\mu_i$'s associated to the prices $c_i$). The computed parameter vector $\hat{r}^*$ is sensitive to the model parameter values. All these aspects are shown in the next example.

**Example 6.5.** Number of prices $m = 3$, resources $N = 10$, $\alpha = 0.95$, price $c = [0.9\ 1\ 1.1]$, $\lambda = [0.6\ 0.5\ 0.3]$, $\mu = [0.2\ 0.2\ 0.4]$.

The following cases are addressed:

1. We choose different heuristic guesses $\bar{r}$ as the initial value for the parameter vector, and then verify the MG-LSTD algorithm convergence. As displayed in Fig. 10, all the executions of the MG-LSTD algorithm converge to the same parameter vector $\hat{r}^*$ regardless of its different initial values $\bar{r}$.
2. We choose a fixed value for the parameter $\sigma$, and then verify the MG-LSTD algorithm convergence with different matrix $\Sigma$ (see Fig. 11).
3. We choose a specific matrix $\Sigma$, and then verify the MG-LSTD algorithm convergence for different values of the parameter $\sigma$ (see Fig. 12).
4. We choose different values for the matrix $\Sigma$, the parameter $\sigma$, and the initial value of the parameter vector $\bar{r}$. Then, we verify the MG-LSTD algorithm convergence (see Fig. 13).
5. We compare the single trajectory recursive LSTD algorithm with the MG-LSTD approach. As shown in Fig. 14, the recursive LSTD converges faster to the steady-state value $\hat{r}_\pi$ since the exploration through the state space is biased towards the applied policy (we recall that the recursive LSTD is used to calculate the approximate value function of a specific policy). Conversely, as for the MG-LSTD, the convergence rate is slower since the selection of the states to be visited is determined by the greedy control actions maximizing Eq. (14). However, as expected, its calculated parameter vector $\hat{r}^*$ is greater in norm than $\hat{r}_\pi$. This case proves the effectiveness of the MG-LSTD algorithm in generating a proper mixture of state visits (in spite of its slower convergence rate).
6. We analyze the results of the MG-LSTD algorithm for resource allocation problems with probabilities $\bar{\lambda} = [0.55\ 0.5\ 0.3]$, $\bar{\mu} = [0.15\ 0.2\ 0.35]$, and with probabilities $\lambda = [0.6\ 0.5\ 0.3]$, $\mu = [0.2\ 0.2\ 0.4]$. As expected, the computed parameter vector $\hat{r}^*$ depends on the model parameters (see Fig. 15).

In all the above cases, we have always considered $Q = 100$ trajectories with $M = 3000$ iterations in length to compute the parameter vectors. Moreover, we have shown the evolution of the parameter vector $L_2$-norm during their calculation over the Monte Carlo trajectories. Thanks to the fact that we use a linear feature-based architecture with the feature vectors having non-negative integer components (see paragraph 5.1), such norm can be linked to the expected total revenue obtained when applying a specific policy (such as the MG-LSTD pricing policy obtained by using Eq. (14)). This aspect can be used to prove that the MG-LSTD approach outperforms the recursive LSTD algorithm (without the need of calculating and comparing long-term revenues, as done in the previous examples).

### 6.2. Further remarks and some guidance on the MG-LSTD algorithm usage

This section has provided some empirical evidence on the MG-LSTD algorithm effectiveness and convergence properties when applied to resource allocation problems.

Even though such aspects are also supported by the theoretical results of LSTD based approaches available in the literature (see Section 4), we have to highlight that the application of MG-LSTD algorithm requires a careful selection of its key-parameters since their improper definition can affect its effectiveness and convergence. In this regard, the following elements have to be considered:

- Being a model-based approach, the computed parameter vector $\hat{r}^*$ is sensitive to the model parameter values, which determine the associated probability distributions. The MG-LSTD can also work in case we have a computer simulator (with the capability of generating samples according to such probability distributions). In both cases, the goodness of the sample generating mechanism can affect the quality of the resulting value function approximation.
- The convergence rate of the MG-LSTD algorithm is expected to increase in case of systems with a very large state space. Being based on Monte Carlo simulations, one can define regions of specific interest and select the initial states of the Monte Carlo trajectories from such regions without going to the expense of accurately evaluating the rest of the state space (Sutton & Barto, 2018), chapters 5 and 8.
- The selection of the feature vectors plays an important role in the computational costs of the MG-LSTD algorithm and the quality of the resulting value function approximation. Such aspect has not been investigated in the paper. To select a proper set of features, a good knowledge of the problem at hand is necessary at least.
- The number of trajectories $Q$ and their length $M$ should be defined in order to guarantee the convergence of the final computed parameter vector. In this respect, the parameter vector learned by the MG-LSTD algorithm under an inappropriate choice of such parameters can diverge (Tsitsiklis & Van Roy, 1997) (e.g., when $M = 1$ the MG-LSTD algorithm works in an off-policy fashion). From an empirical point of view, $M$ should be much greater than $Q$ to assure a significant progress in the calculation of the parameter vector over long enough trajectories.

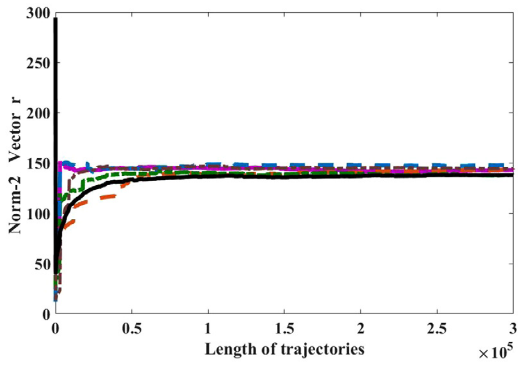

**Fig. 10.** Example 6.5: MG-LSTD algorithm convergence when choosing different initial conditions $\bar{r}$.

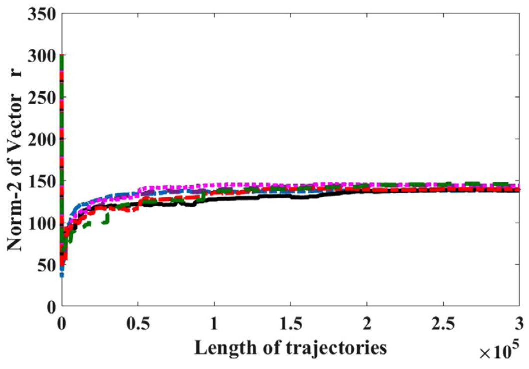

**Fig. 11.** Example 6.5: MG-LSTD algorithm convergence when choosing a fixed value for the parameter $\sigma$ and different matrices $\Sigma$.

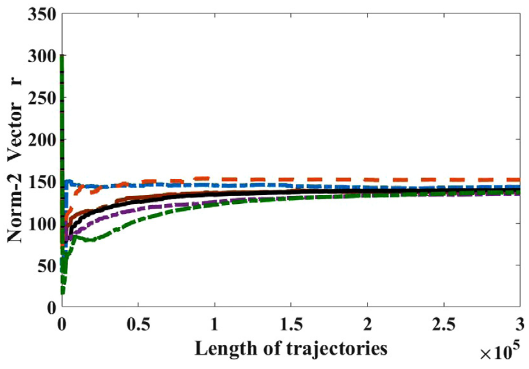

**Fig. 12.** Example 6.5: MG-LSTD algorithm convergence when choosing a fixed matrix $\Sigma$ and different values for the parameter $\sigma$.

- The selection mechanism of the initial state for each Monte Carlo simulation can bias the computed value function approximation. Choosing such initial states according to a fixed probability distribution over the entire state space can be inappropriate for systems with very large state space. To counteract this issue, the state space can be partitioned by applying approaches such as the feature-based aggregation method (Bertsekas, 2012a), section 6.5.

A study on the MG-LSTD convergence properties from an analytical point of view as well as a deep analysis of the computed parameter vector sensitivity to the model parameter values, both supported by more complex examples, are regarded as future work.

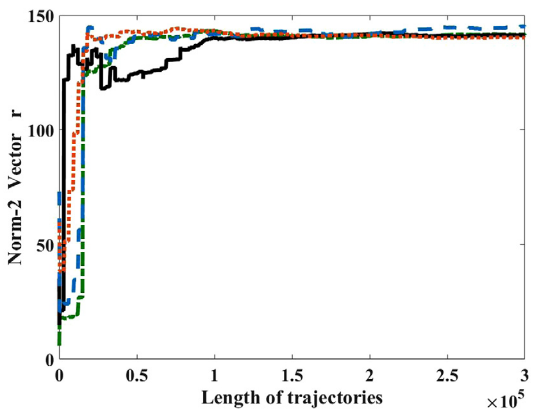

**Fig. 13.** Example 6.5: MG-LSTD algorithm convergence when changing the parameters $\bar{r}$, $\sigma$, and $\Sigma$.

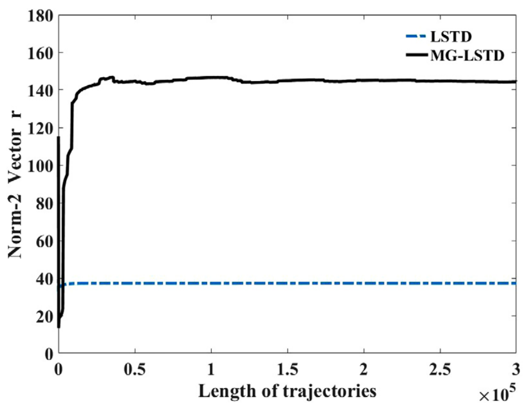

**Fig. 14.** Example 6.5: Comparison between the single trajectory recursive LSTD (dashed line) and the MG-LSTD (solid line) algorithms.

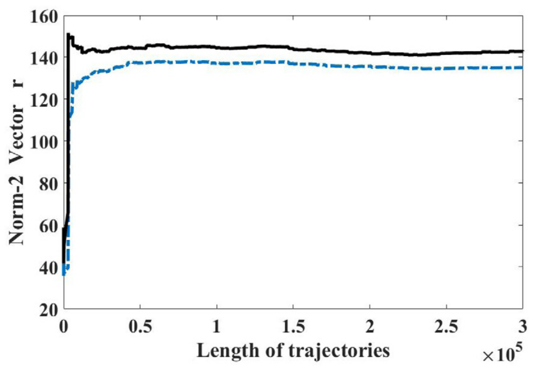

**Fig. 15.** Example 6.5: Sensitivity of the parameter vector $\hat{r}^*$ to the birth/death probability values ($\bar{\lambda}$, $\bar{\mu}$ (dashed line) and $\lambda$, $\mu$ (solid line)).

## 7. Conclusions

In this paper, we have presented a Least-Squares Temporal Difference (LSTD) based method called "Multi-trajectory Greedy LSTD" (MG-LSTD). It is an exploration-enhanced recursive LSTD algorithm with the policy improvement embedded within the LSTD algorithm iterations. This way, we can address the scalability issues both in the state and the action space for real-world applications modeled via Stochastic Dynamic Programming (SDP). The proposed algorithm has been analyzed by investigating the properties of the resulting value function approximation and its connection with the Least-Squares Policy Evaluation (LSPE).

We have applied the MG-LSTD algorithm to solve resource allocation problems. They have been modeled as a set of parallel Birth–Death Processes (BDPs), each of them corresponding to one admissible price. A dynamic pricing policy can be calculated in order to maximize the expected total revenue over an infinite time horizon. Some significant operational scenarios have been defined and solved to show the advantages in applying the pricing policies computed by the MG-LSTD.

At the same time, we have provided some experimental evidence on the MG-LSTD algorithm convergence properties in function of its key-parameters. Such convergence properties have been also addressed from a more qualitative point of view, by analyzing the MG-LSTD key-aspects and its connections with other approaches (and the related convergence results) reported in the literature.

As for future work, we intend to investigate the role of different exploration mechanisms in the value function approximation. Moreover, we plan to analyze the MG-LSTD convergence properties from an analytical point of view. Finally, we intend to extend the presented formulation for resource allocation problems to incorporate further requirements, such as the possibility of handling advance resource requests together with the related overbooking issues.

## CRediT authorship contribution statement

**Ali Forootani:** Conceptualization, Methodology, Formal analysis, Writing of the manuscript, Numerical simulations, Data curation. **Massimo Tipaldi:** Conceptualization, Methodology, Formal analysis, Writing of the manuscript, Validation. **Majid Ghaniee Zarch:** Writing - review & editing, Validation. **Davide Liuzza:** Writing - review & editing, Validation, Supervision. **Luigi Glielmo:** Writing - review & editing, Validation, Supervision.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

Arulkumaran, K., Deisenroth, M. P., Brundage, & Bharath, A. A. (2017). Deep Reinforcement Learning A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38.

Bellman, R. (1957). *Introduction to probability*. USA: Princeton University Press.

Bertsekas, D. P. (2011a). Temporal difference methods for general projected equations. *IEEE Transactions on Automatic Control*, *56*(9), 2128–2139.

Bertsekas, D. P. (2011b). Approximate policy iteration: a survey and some new methods. *Journal of Control Theory and Applications*, *9*(3), 310–335.

Bertsekas, D. P. (2012a). *Dynamic programming and optimal control, Vol. II* (4th ed.). Belmont, Massachusetts: Athena Scientific.

Bertsekas, D. P. (2012b). Lambda-policy iteration: A review and a new implementation. In F. Lewis, & D. Liu (Eds.), *Reinforcement learning and approximate dynamic programming for feedback control*. IEEE Press.

Bertsekas, D. P. (2017). *Dynamic programming and optimal control, Vol. I* (4th ed.). USA: Athena Scientific, Belmont, Massachusetts.

Bertsekas, D. P. (2019a). Feature-based aggregation and deep reinforcement learning: A survey and some new implementations. *IEEE/CAA Journal of Automatica Sinica*.

Bertsekas, D. P. (2019b). *Reinforcement learning and optimal control*. Belmont, Masachusetts: Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (2000). *Introduction to probability*. USA: Massachusetts Institute of Technology.

Bertsekas, D. P., & Yub, H. (2009). Projected equation methods for approximate solution of large linear systems. *Journal of Computational and Applied Mathematics*, *227*, 27–50.

Browne, C. B., Powley, E., White house, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., et al. (2012). Survey of Monte Carlo tree search methods. *IEEE Transaction on Computational Intelligence and AI in Games*, *4*(1).

Crawford, F. W., Minin, V. N., & Suchard, M. A. (2014). Estimation for general birth-death processes. *Journal of the American Statistical Association*, *109*(506), 730–747.

D'Angelo, G., Tipaldi, M., Palmieri, F., & Glielmo, L. (2019). A data-driven approximate dynamic programming approach based on association rule learning: Spacecraft autonomy as a case study. *Information Sciences*, *504*, 501–519.

Dann, C., Neumann, G., & Peters, Jan (2014). Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research (JMLR)*, *15*, 809–883.

De Farias, D. P., & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, *51*(6), 850–865.

Deng, K., Chen, Y., & Belta, C. (2017). An approximate dynamic programming approach to multi-agent persistent monitoring in stochastic environments with temporal logic constraints. *IEEE Transactions on Automatic Control*, *62*(9), 4549–4563.

Forootani, A., Iervolino, R., & Tipaldi, M. (2019). Applying unweighted least-squares based techniques to stochastic dynamic programming: theory and application. *IET Control Theory & Applications*, *13*(15), 2387–2398.

Forootani, A., Tipaldi, M., Ghaniee Zarch, M., Liuzza, D., & Glielmo, L. (2019). Modeling and solving resource allocation problems via a Dynamic Programming approach. *Intenational Journal of Control*.

Gehring, C., Pan, Y., & White, M. (2016). Incremental Truncated LSTD. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence*.

Geist, M., & Pietquin, O. (2013). Algorithmic survey of parametric value function approximation. *IEEE Transactions on Neural Networks and Learning Systems*, *24*(6), 845–867.

Geramifard, A., Bowling, M., & Sutton, R. S. (2006). Incremental least-squares temporal difference learning. In *Proceedings of the national conference on artificial intelligence*.

Guo, W., Liu, F., Si, J., He, D., Harley, R., & Mei, S. (2016). On-line supplementary ADP learning controller design and application to power system frequency control with large-scale wind energy integration. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(8), 1748–1761.

Hasselt, H. v., & Wiering, M. A. (2007). Reinforcement Learning in Continuous Action Spaces. In *Proceedings of IEEE symposium on approximate dynamic programming and reinforcement learning*.

Hoffman, M. W., Lazaric, A., Ghavamzadeh, M., & Munos, R. (2012). Regularized Least Squares Temporal Difference learning with nested $l_2$ and $l_1$ penalization. In *European workshop on reinforcement learning*. (pp. 102–114).

Howard, R. A. (1960). *Dynamic programming and markov processes*.

Keerthisinghe, C., Verbic, G., & Chapman, A. C. (2016). A fast technique for smart home management: ADP with temporal difference learning. *IEEE Transactions on Smart Grid*, *9*(4), 3291–3303.

Keller, P., Mannor, S., & Precup, D. (2006). Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning. In *Proceedings of the 23rd international conference on machine learning*.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, *4*, 1107–1149.

Nedić, A., & Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, *13*(1), 79–110.

Powell, W. B. (2011). *Wiley series in probability and statistics*, *Approximate dynamic programming: solving the curses of dimensionality* (2nd ed.).

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature International Journal of Science*, *550*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning an introduction* (2nd ed.). Cambridge, MA: MIT Press.

Tang, Y., He, H., Wen, J., & Liu, J. (2015). Power system stability control for a wind farm based on adaptive dynamic programming. *IEEE Transactions on Smart Grid*, *6*(1), 166–177.

Tipaldi, M., & Glielmo, L. (2018). A survey on model-based mission planning and execution for autonomous spacecraft. *IEEE Systems Journal*, *12*(4), 3893–3905.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5).

Yu, H. (2012). Least Square Temporal Difference methods: an analysis under general conditions. *SIAM Journal on Control and Optimization*, *50*(6), 3310–3343.

Yu, H., & Bertsekas, D. P. (2009). Convergence results for some temporal difference methods based on least squares. *IEEE Transactions on Automatic Control*, *54*(7), 1515–1531.