Model-based reinforcement learning using on-line clustering

Nikolaos Tziortziotis and Konstantinos Blekas
Department of Computer Science, University of Ioannina
P.O.Box 1186, Ioannina 45110 - Greece
Email:{ntziorzi,kblekas}@cs.uoi.gr

***Abstract*—A significant issue in representing reinforcement learning agents in Markov decision processes is how to design efficient feature spaces in order to estimate optimal policy. The particular study addresses this challenge by proposing a compact framework that employs an on-line clustering approach for building appropriate basis functions. Also, it performs a state-action trajectory analysis to gain valuable affinity information among clusters and estimate their transition dynamics. Value function approximation is used for policy evaluation in a least-squares temporal difference framework. The proposed method is evaluated in several simulated and real environments, where we took promising results.**

***Index Terms*—mixture models, on-line EM, clustering, model-based reinforcement learning**

I. Introduction

Reinforcement Learning (RL) aims at controlling an autonomous agent in unknown stochastic environments [1]. Typically, the environment is modelled as a Markov Decision Process (MDP), where the agent receives a scalar reward signal that evaluates every transition. The objective is to maximize its long-term profit that is equivalent to maximizing the expected total discounted reward. *Value function* is used for measuring the quality of a policy, which associates to every state the expected discounted reward when starting from this state and all decisions are made following the particular policy. However, in cases with large or infinite state spaces the value function cannot be calculated explicitly. In such domains a common strategy is to employ function approximation methodologies, by representing the value function as a linear combination of some set of basis functions [2].

The Temporal Difference (TD) family of algorithms [1] provides a nice framework for policy evaluation, where the least-squares temporal difference (LSTD) [3] is one of the most popular mechanism for approximating the value function of a given policy. The least square policy iteration (LSPI) [4] is an off-policy method that extends the LSTD to control problems, where the policy is refined iteratively. Recently, an online version of the LSPI have been proposed [5] that overcome the limitation of the LSPI in online problems. Also, kernelized RL methods [6] have been paid a lot of attention last years by employing kernel techniques to standard RL methods [7] and Gaussian Processes as a description model for the value function [8]. In most cases the basis functions used for estimating the value function remain fixed during the learning process, as for example a recent work presented in [9] where a number of fixed Fourier basis functions are used for value function approximation. However, there are some works where the basis functions are estimated during the learning process. In [10] for example, a steady number of basis functions are tuned in a batch manner by building a graph over the state space and then calculating the k eigenvectors of the graph Laplacian matrix. In another work [11] a set of k RBF basis function are adjusted directly over the Bellman's equation of the value function. Finally, in [12] the probability density function and the reward model, which are assumed to be known, are used for creating basis function from Krylov space vectors (powers of the transition matrix used to systems of linear equations).

In the particular work, we propose a *model-based* approach for value function approximation which is based on an on-line clustering approach for partitioning the state-action input space into clusters. This is done by considering an appropriate mixture model that is trained incrementally through an on-line version of the Expectation-Maximization (EM) algorithm [13], [14]. A kernel-based mechanism for creating new clusters is also incorporated. The number and the structure of the created clusters compose a dictionary of basis functions which used next for policy evaluation. In addition, during the clustering procedure the transitions among adjacent clusters are observed and the transition probabilities are computed, as well as their average reward. Policy is then evaluated at each step by estimating the linear weights of the value function through the least-squares framework. The proposed methodology has been tested to several known simulated and real environments where we measure its efficiency in discovering the optimal policy. Comparisons have been made using an online version of LSPI algorithm [5].

In Section 2, we briefly present some preliminaries and review the basic LSTD scheme for value function approximation. Section 3 describes the online clustering scheme and the model-based approach. In Section 4 the experimental results are presend and finally, in Section 5 we give conclusions and suggestions for future research.

II. Background

A *Markov Decision Process* (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is a set of states; $\mathcal{A}$ a set of actions; $P : \mathcal{S} \times \mathcal{A} \times S \rightarrow [0, 1]$ is a Markovian transition model that specifies the probability $P(\boldsymbol{s}'|\boldsymbol{s}, a)$ of transition

to state $\boldsymbol{s}'$ when taken an action a in state $\boldsymbol{s}$; $R : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function for a state-action pair; and $\gamma \in (0, 1)$ is the discount factor for future rewards. A *stationary policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a mapping from states to actions and denotes a mechanism for choosing actions. An *episode* is a sequence of transitions: $(\boldsymbol{s}_1, a_1, r_1, \boldsymbol{s}_2, \ldots)$.

The notion of *value function* is of central interest in reinforcement learning tasks. Given a policy π, the value $V^\pi(\boldsymbol{s})$ of a state $\boldsymbol{s}$ is defined as the expected discounted sum of rewards obtained when starting from this state until the current episode terminates:

$$V^\pi(\boldsymbol{s}) = E_\pi \left[R(\boldsymbol{s}_t) + \gamma V^\pi(\boldsymbol{s}_{t+1}) | \boldsymbol{s}_t = \boldsymbol{s} \right] \ , \tag{1}$$

that satisfies the Bellman equations, which expresses a relationship between the values of successive states in the same episode. Similarly, the state-action value function $Q(\boldsymbol{s}, a)$ denotes the expected cumulative reward as received by taking action a in state $\boldsymbol{s}$

$$Q^\pi(\boldsymbol{s}, a) = \overline{R}_a(\boldsymbol{s}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} P(\boldsymbol{s}' | \boldsymbol{s}, a) \max_a Q(\boldsymbol{s}', a) \ , \tag{2}$$

where $\overline{R}_a(\boldsymbol{s})$ specifies the average reward for executing a in $\boldsymbol{s}$. The objective of RL problems is to estimate an optimal policy π^* by choosing actions that yields the optimal action-state value function Q^*:

$$\pi^*(\boldsymbol{s}) = \arg\max_a Q^*(\boldsymbol{s}, a) \tag{3}$$

A common choice for representing the value function is through a linear function approximation using a set of k basis functions $\phi_j(\boldsymbol{s}, a)$:

$$Q(\boldsymbol{s}, a) = \boldsymbol{\phi}(\boldsymbol{s}, a)^\top \boldsymbol{w} = \sum_{j=1}^{k} \phi_j(\boldsymbol{s}, a) w_j \ , \tag{4}$$

where $\boldsymbol{w} = (w_1, \ldots, w_k)$ is a vector of weights which are unknown and must be estimated so as to minimize the approximation error. The selection of the basis functions is very important and must be chosen to encode properties of the state and action relevant to the proper determination of the Q values.

The LSTD approach combines the Bellman operator with the least-squares estimation procedure. If we assume a N-length trajectory of transitions $(\boldsymbol{s}_i, a_i, r_i, \boldsymbol{s}_{i+1})$ sampled from the MDP, a set of N equations is obtained:

$$r_i = (\boldsymbol{\phi}(\boldsymbol{s}_i, a_i) - \gamma \boldsymbol{\phi}(\boldsymbol{s}_{i+1}, a_{i+1}))^\top \boldsymbol{w} \ , \tag{5}$$

which can be further written as

$$\boldsymbol{R} = H \Phi \boldsymbol{w} \ , \tag{6}$$

where

$$H = \begin{bmatrix} 1 & -\gamma & 0 & \cdots & 0 \\ 0 & 1 & -\gamma & \cdots & 0 \\ \vdots & & & \vdots & \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} , \tag{7}$$

$$\Phi = [\phi(\boldsymbol{s}_1, a_1)^\top, \ldots, \phi(\boldsymbol{s}_N, a_N)^\top] \text{ and} \tag{8}$$

$$\boldsymbol{R} = [r_1, \ldots, r_N]. \tag{9}$$

Then, the LSTD approach estimates the weights of the above linear equation according to the least-squares solution:

$$\hat{\boldsymbol{w}} = (\Phi^\top H \Phi)^{-1} \Phi^\top \boldsymbol{R} \ . \tag{10}$$

III. On-line clustering and value function approximation

The proposed methodology is based on a policy evaluation scheme that incrementally separates the input space into clusters and estimates the transition probabilities among them. Thus a dictionary of features is dynamically constructed for modeling the value functions. To what follows we will assume that the input samples are state-action pairs, denoted as $x_n = (\boldsymbol{s}_n, a_n)$. We will also consider a finite action space of size M.

Suppose we are given a data set of N samples $\{x_1, x_2, \ldots, x_N\}$. The task of clustering aims at partitioning the input set into k disjoint clusters, containing samples with common properties. Mixture modeling [14] provides a convenient and elegant framework for clustering, where we consider that the properties of a single cluster j is described implicitly via a probability distribution with parameters θ_j. This can be formulated as:

$$p(x|\Theta_k) = \sum_{j=1}^{k} \pi_j p(x|\theta_j) \ , \tag{11}$$

where Θ_k denotes the set of mixture model parameters. The parameters $0 < \pi_j \leq 1$ represent the mixing weights satisfying $\sum_{j=1}^{k} \pi_j = 1$. Since we are dealing with two source of information (state-actions), in our scheme we assume that the conditional density for each cluster is written as a product of two pdfs:

- a Gaussian pdf $\mathcal{N}(\boldsymbol{s}; \boldsymbol{\mu}_j, \Sigma_j)$ for the state $\boldsymbol{s}$ and
- a multinomial pdf $Mu(a; \boldsymbol{\rho}_j) = \prod_{i=1}^{M} \rho_{ji}^{I(a,i)}$ for the action a, where $\boldsymbol{\rho}_j$ is a M-length probabilistic vector while $I(a, i)$ is a binary indicator function (1 if $a = i$; 0 otherwise).

This can be written as:

$$p(x|\theta_j) = \mathcal{N}(\boldsymbol{s}; \mu_j, \Sigma_j) Mu(a; \boldsymbol{\rho}_j) \text{ , where } \theta_j = \{\boldsymbol{\mu}_j, \Sigma_j, \boldsymbol{\rho}_j\} \ . \tag{12}$$

Mixture modelling treats clustering as an estimation problem for the model parameters $\Theta_k = \{\pi_j, \theta_j\}_{j=1}^k$ by maximizing the log-likelihood function:

$$L(\Theta_k) = \sum_{n=1}^{N} \log\{\sum_{j=1}^{k} \pi_j \mathcal{N}(\boldsymbol{s}_n; \boldsymbol{\mu}_j, \Sigma_j) Mu(a_n; \boldsymbol{\rho}_j)\} \ . \tag{13}$$

The Expectation-Maximization (EM) algorithm [13] is an efficient framework that can be used for this purpose. It iteratively performs two steps: The *E-step*, where the current posterior probabilities of samples to belong to each cluster are calculated:

$$z_{nj} = \frac{\pi_j p(x_n|\theta_j)}{\sum_{j'=1}^{k} \pi_{j'} p(x_n|\theta_{j'})} \ , \tag{14}$$

and the *M-step*, where the maximization of the expected complete log-likelihood is performed. This leads to closed-form update rules for the model parameters [14].

In our case the samples are non-stationary and are generated sequentially. We present here an extension of the EM algorithm [15] for online estimating mixture models that suits our particular needs. It consists of two phases: first we have a mechanism for deciding whether or not a new cluster must be created, and secondly the main EM procedure is performed for adjusting the structure of clusters so as to incorporate the new sample.

Lets assume that a random sample $x_n = (\boldsymbol{s}_n, a_n)$ is observed. The method first performs the E-step and calculates the posterior probabilities values z_{nj} (Eq. 14) based on the current k-order mixture model. The winner cluster $j^* \in [1, k]$ is then found according to the maximum posterior value, i.e.

$$j^* = \arg\max_{j=1}^{k} \{z_{nj}\}. \tag{15}$$

If the degree of belongingness of x_n to cluster j^*, as given by the next kernel function:

$$K(x_n, j^*) = K_s(x_n, j^*) K_a(x_n, j^*) \ , \tag{16}$$

where

$$K_s(x_n, j^*) = \exp(-0.5(\boldsymbol{s}_n - \boldsymbol{\mu}_{j^*})^\top \Sigma_{j^*}^{-1} (\boldsymbol{s}_n - \boldsymbol{\mu}_{j^*})),$$

and

$$K_a(x_n, j^*) = \prod_{i=1}^{M} \rho_{j^* i}^{I(a_n, i)} \ ,$$

is less than a predefined threshold value K_{min}, a new cluster $(k+1)$ must be created. This is done by initializing it properly:

$$\begin{aligned} \boldsymbol{\mu}_{k+1} &= \boldsymbol{s}_n, \\ \Sigma_{k+1} &= 0.5 \times \Sigma_{j^*}, \\ \rho_{k+1,i} &= \begin{cases} \xi, & \text{if } \ i = a \\ \frac{1-\xi}{M-1}, & \text{otherwise} \end{cases}, \end{aligned}$$

where ξ is set to a large value (e.g. $\xi = 0.9$). The M-step is applied next that provides a step-wise update procedure for the model parameters using the next rules:

$$\pi_j = (1 - \lambda)\pi_j + \lambda z_{nj} \ , \tag{17}$$

$$\boldsymbol{\mu}_j = \boldsymbol{\mu}_j + \lambda z_{nj} \boldsymbol{s}_n \ , \tag{18}$$

$$\Sigma_j = \Sigma_j + \lambda z_{nj} (\boldsymbol{s}_i - \boldsymbol{\mu}_j)(\boldsymbol{s}_i - \boldsymbol{\mu}_j)^\top \ , \tag{19}$$

$$n_{ji} = n_{ji} + I(a_n, i) z_{nj} \ \text{ and } \rho_{ji} = \frac{n_{ji}}{\sum_{l=1}^{M} n_{jl}} \ , \tag{20}$$

where the term λ takes a small value (e.g. 0.09) and it can be decreased over episode. That is necessary to remark here is the following: In the M-step of the normal (off-line) EM algorithm, the update rules for both Gaussian parameters have the quantity $\sum_{n=1}^{N} z_{nj}$ in the denominator (and the size N for the weights π_j) e.g. $\boldsymbol{\mu}_j = \frac{\sum_n z_{nj} \boldsymbol{s}_n}{\sum_n z_{nj}}$. During the on-line version, each new sample contributes to the computation of these parameters by a factor equal to $z_{nj} / \sum_{n=1}^{N} z_{nj}$. Therefore, when the number of observations becomes too large, the incoming new sample will barely influence the model parameters. To avoid this situation we have selected the above update rules where we fix the contribution of new samples to λ.

A. Model-based approximation

As it is obvious from the previous discussion, the EM-based on-line clustering approach performs a partitioning of the input (state-action) space into clusters that have also the same *transition dynamics*. Clusters can be seen as nodes of a directed (not full) graph that communicate to each other. The learning process construct new nodes in this graph (by performing a splitting process) and can also provide useful information (frequency and distance) between adjacent nodes. In another point of view, the proposed scheme can be seen as a type of relocatable action model (RAM), that has been proposed recently [16] and provides a decomposition or factorization of the transition function.

We assume a trajectory of transitions $(\boldsymbol{s}_i, a_i, r_i, \boldsymbol{s}_{i+1})$ in the same episode. During the on-line clustering we maintain for each cluster $j = 1, \ldots, k$ the following quantities:

- $\bar{t}_{j,j'}$: the mean number of time-steps between two successively observed clusters j, j'
- $\overline{R}(j, j')$: the mean total reward of the transition from cluster j to cluster j'
- $n_{j,j'}$: the total number of times (frequency) that we have observed this transition
- n_j: the total number of times that the cluster j is visited.

Furthermore, another two useful quantities can be calculated: The transition probabilities $P(j'|j)$ between two (adjacent) clusters from their relative frequencies, i.e. $P(j'|j) = \frac{n_{j,j'}}{n_j}$, and the mean value of the reward function for any cluster j as $\overline{R}(j) = \sum_{j'} P(j'|j) \overline{R}(j, j')$. These can be used for the policy estimation process.

The equation for the action value function for a cluster j then becomes

$$Q(j) = \overline{R}(j) + \sum_{j'} P(j'|j) \gamma^{\bar{t}_{j,j'}} Q(j') \ , \tag{21}$$

where the summation is made over the neighbourhood of cluster j (adjacent clusters j' where $P(j'|j) > 0$). Thus, a set of k equations are available for the k quantities $\overline{R}(j)$ (observations):

$$\overline{\boldsymbol{R}}_k = H_k \Phi_k \boldsymbol{w}_k \ , \tag{22}$$

where we have considered linear function approximation for the action value function. In this case the kernel design matrix $\Phi_k = [\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_k]$ is explicitly derived from the online clustering solution using the kernel function of Eq. 16:

$$[\Phi_k]_{jj'} = \exp(-0.5(\boldsymbol{\mu}_j - \boldsymbol{\mu}_{j'})^\top \Sigma_{j'}^{-1} (\boldsymbol{\mu}_j - \boldsymbol{\mu}_{j'})) \boldsymbol{\rho}_j^\top \boldsymbol{\rho}_{j'} \ . \tag{23}$$

Also, the matrix H_k contains the coefficients of Eq. 21, i.e. at each line j we have $[H_k]_{jj} = 1$ and $[H_k]_{j,j'} = -P(j'|j) \gamma^{\bar{t}_{j,j'}}$ in case where two clusters are adjacent ($P(j'|j) > 0$). Finally,

$\overline{\boldsymbol{R}}_{\boldsymbol{k}}$ is the vector of the calculated mean reward values per cluster, i.e. $\overline{\boldsymbol{R}}_{\boldsymbol{k}} = [\overline{R}(1), \ldots, \overline{R}(k)]$. The least-square solution for the linear weights $\boldsymbol{w}_k$ can be obtained then as

$$\hat{\boldsymbol{w}}_k = (\Phi_k^\top H_k \Phi_k)^{-1} \Phi_k^\top \overline{\boldsymbol{R}}_{\boldsymbol{k}}. \tag{24}$$

Thus, for an input state-action pair $x = (\boldsymbol{s}, a)$ we can estimate the action value function according to the current policy as:

$$Q(\boldsymbol{s}, a) = \boldsymbol{\phi}(\boldsymbol{s}, a)^\top \hat{\boldsymbol{w}}_k, \tag{25}$$

where $\boldsymbol{\phi}(s, a) = [K(x, 1), \ldots, K(x, M)]$. The above procedure is repeated until convergence, or a number of episodes is found. The method starts with a single cluster $k = 1$ initialized by the first sample taken by the agent. At every time step the EM-based on-line clustering procedure and the policy evaluation stage are sequentially performed. The overall scheme of the proposed methodology is given in Algorithm 1.

Algorithm 1 General framework of the proposed methodology

1: Start with $k = 1$ and use first point $x_i = (\boldsymbol{s}_1, a_1)$ for initializing it. Set a random value to weight w_1. $t = 0$.
2: **while** convergence or maximum number of episodes not found **do**
3: Suppose previous input $x_i = (\boldsymbol{s}_i, a_i)$.
4: Observe new state $\boldsymbol{s}_{i+1}$.
5: Select action according to the current policy $a_{i+1} = \arg\max_{l=1}^{M} Q(\boldsymbol{s}_{i+1}, l)$.
6: Find the winning cluster $j^* = \arg\max_{j=1}^{k} \{z_{nj}\}$.
7: **if** $K(x_{i+1}, m_{j^*}) < K_{min}$ **then**
8: Create a new cluster ($k = k + 1$) and initialize its prototype m_k with x_{i+1}.
9: Create a new weight w_k of linear model and initialize it randomly. $\boldsymbol{w}_t = \boldsymbol{w}_t \cup w_k$.
10: **else**
11: Update the prototype m_{j^*} of the winning cluster using Eqs. 17-20.
12: **end if**
13: Obtain the new k basis functions as: $\phi_j(\boldsymbol{s}, a) = K((\boldsymbol{s}, a), m_j)$, $\forall\ j = 1, \ldots, k$.
14: Update the environment statistics.
15: Update the model weights $\boldsymbol{w}_t$ according to Eq. 24.
16: $t = t + 1$
17: **end while**

IV. Experimental results

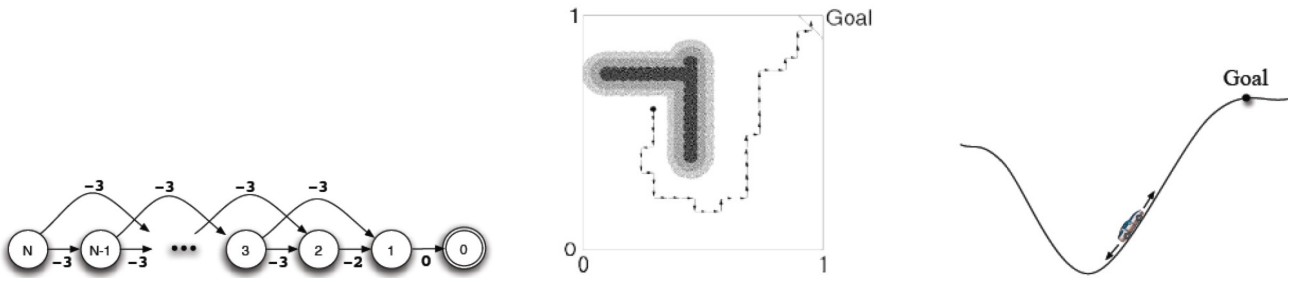
A series of experiments have been conducted in a variety of well-known simulated benchmarks as well as in real environments, in order to study the performance of the proposed model-based approach. More specifically, the three well-known simulated benchmark where was used in our experiments is the Boyan's Chain, the Puddle World and the Mountain Car, illustrated in Fig.1. At the same time, using the Pioneer/People mobile robot (Fig.2) we achieve to examine the proposed methodology in more realistic environments which incorporates in their models a number of physical restrictions. Comparisons have been made by using two famous methodologies: the least square temporal difference (LSTD) [3] in the case of policy evaluation problem (Boyan's Chain), i.e to evaluate the value function of a given policy, and the online least square policy iteration (LSPI) [5] in the case of control learning problem, i.e. discover the optimal policy. In the case of online LSPI, the action value function is representing as a linear model by using an equidistant fixed $N \times N$ grid of Gaussian radial basis functions (RBFs) over the state space, while each basis function is replicated for every action so that each action has each own parameters. Furthermore, the policy is updated after each 10 steps. In all domains, the discount factor γ was set equal to 1, while the thresholod K_{min} for creating new clusters was set as $K_{min} = 0.7$. Finally, in order to introduce stochasticity into the transitions and to attain better exploration of the environment, the actions are chosen ϵ-greedy. In the case of the proposed method the ϵ is initially set equal to 0.1 and decreases steady at each step. On the other hand, in the online LSPI the exploration probability is initially set to value $\epsilon_0 = 1$, and decays exponential once each time step with the decay rate, $\epsilon_d = 0.9962$.

A. Experiments in Simulated environments

The first series of experiments was made with the classical Boyan's chain problem (Fig. 1(a)) with $N = 13$ and $N = 98$ states [3]. In this application states are connected in a chain, where an agent located in a state $x > 2$ can move into states $s - 1$ and $s - 2$ with the same probability receiving a reward of -3 ($r = -3$). On the other hand, from states 2 and 1, there are only deterministic transitions to states 1 and 0 where the received rewards are -2 and 0, respectively. Every episode starts at state N and terminates at state 0. In the specific domain, both the policy evaluation problem as well as the problem of discovering the optimal policy have been considered. It must be noted that both LSTD and online LSPI methods had a number of RBF kernels equal to the number of states with a kernel width equal to 1. Also, we have allowed our method to construct the same number of clusters, in an attempt to focus our study on the effect of the clusters' transition probabilities. The results are illustrated in Fig. 3 plotting the root mean square error (RMSE) between the true and the estimated value function (1st problem), as well as the mean returns of the last 100 episodes (2nd problem) per method. As it is obvious, the proposed method performs better exploration estimating the optimal value function more accurately than the LSTD. On the control problem, both methodologies achieve to discover the optimal policies but the proposed method converges in a higher rate than the online LSPI.

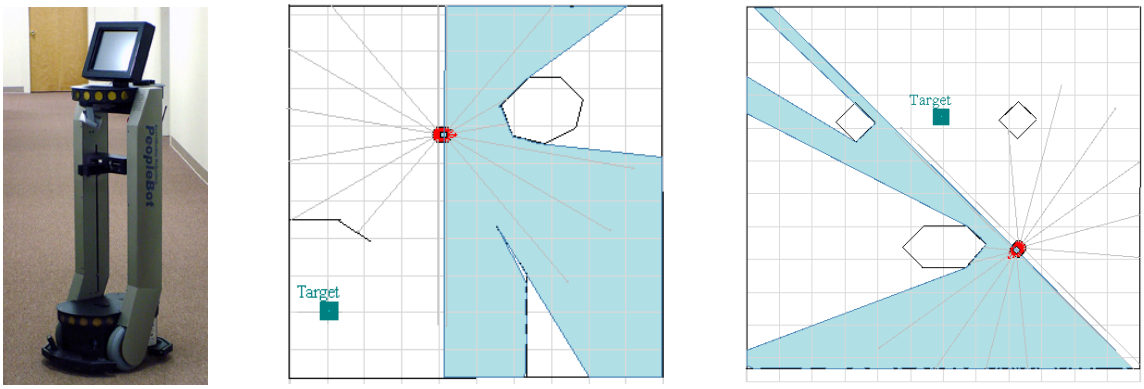
Another two simulated environments used in our experiments are the Puddle World [17] and the Mountain Car [1] (Figs. 1(b),(c)), found on the RL-Glue Library[1]. The Puddle World is a continuous world with two oval puddles and the goal is to reach the upper right corner from any random

[1] Available at http://library.rl-community.org/wiki



(a) Boyan's Chain (b) Puddle World (c) Mountain Car

Fig. 1. Simulated experimental domains



(a) Pioneer/PeopleBot (b) Stage world S_1 (c) Stage world S_2

Fig. 2. The mobile robot and the 2D-grid maps used in our experiments. These are two snapshots from the MobileSim simulator with visualization of the robot's laser and sonar range scanners.

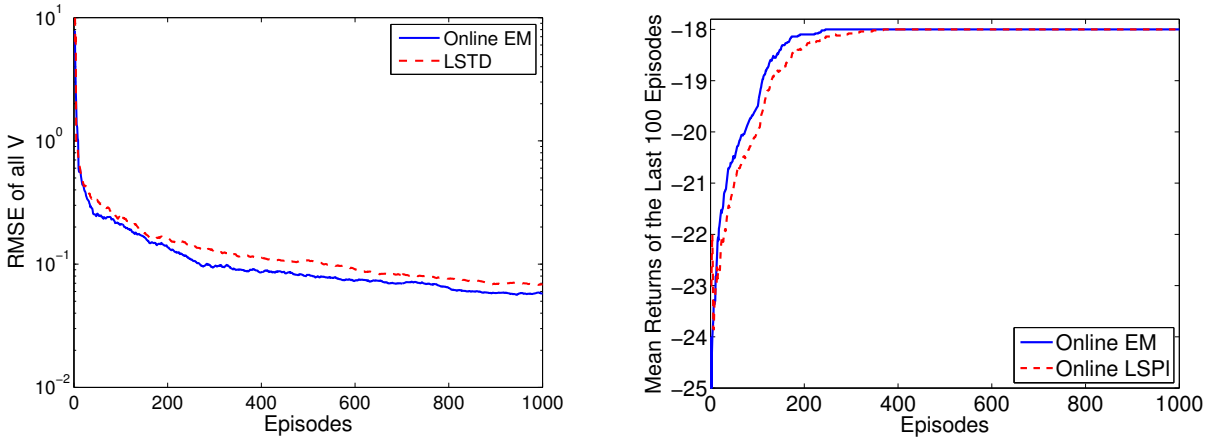
position, avoiding the two puddles. The environmental states are 2-dimensional (x and y coordinates) and it can choose one of four actions that correspond to the four major compass directions: up, right, left, or right. The received reward is -1 except for the puddle region where a penalty between 0 and -40 is received, depending on the proximity to the middle of the puddle. In the Mountain Car simulation problem the objective is to drive an underpowered car up a steep mountain road from a valley to the top of the right slope. There are two continuous environmental variables (horizontal position and velocity) and three possible actions: $+1$ (full throttle forward), -1 (full throttle reverse) and 0 (zero throttle). The reward that received at each time step is $r = -1$ except for the case where the goal state is reached ($r = 0$). At the beginning of each episode, the car is standing motionless in a random initial position. An episode is terminated either when the car reaches the goal, or the total number of steps exceeds a maximum allowable value (e.g. 1000).

In both problems, comparisons have been made with the online LSPI method where an equidistant fixed 20×20 grid of Gaussian RBFs is used over the state spaces. The depicted results are illustrated in Fig. 4 that gives the mean number of returns received during the last 30 episodes. As it is obvious, our method manages to discover faster the optimal policy, where in the case of mountain car the difference is quite noticeable. That is interesting to note here is that (as it was expected) experiments have shown a significant sensitivity of the online LSPI to the value of the RBF's kernel width parameter. Here we show the best results found.

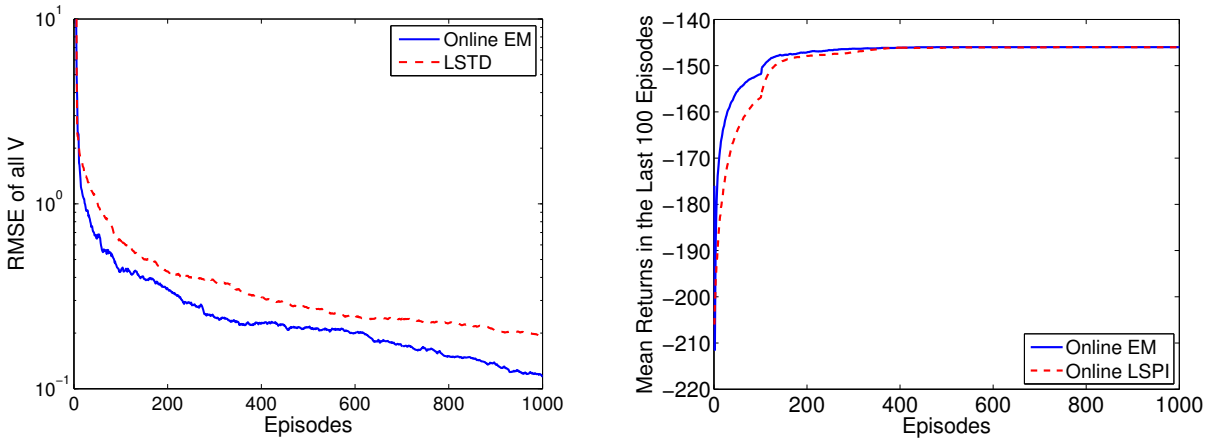
B. Experiments in Real Environments

A number of experiments have also been conducted in a number of real environments by using the wheeled real mobile robot platform Pioneer/PeopleBot, shown in Fig. 2(a), based on the robust P3-DX base. The robot is equipped with advanced tools for communication and control, like the ARIA (Advanced Robot Interface for Applications) library which provides a nice framework for controlling and receiving data from the MobileRobots platforms. At the same time a plethora of sensors are included into the specific type of robot, such as sonar, laser, bumpers and a pan-tilt-zoom camera. For the purposes of our experiments, only the sonar and laser sensors were used in the case of the obstacle avoidance. Furthermore, an embedded motion controller provide at each time step, the robot state such as the robot position (x, y, θ), sensors range sensing data, e.t.c. Due to numerous physical restrictions, such as the strict battery life, the training of the specific methodologies are achieved by using the MobileSim[2] simulator. The specific simulator is built on the famous Stage

[2]more details can be found at http://robots.mobilerobots.com/wiki/MobileSim

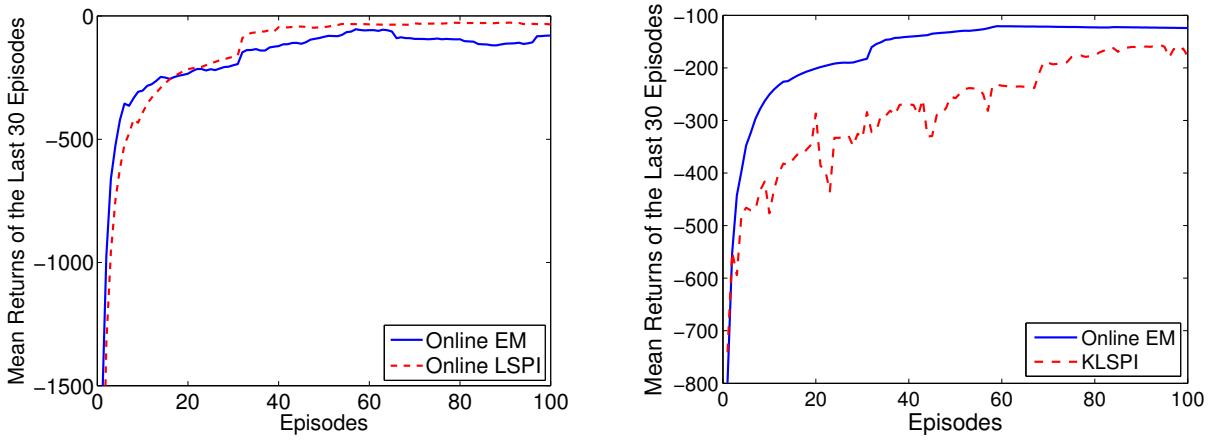


(a)13-states chain



(b) 98-states chain

Fig. 3. Comparative results on policy evaluation and learned policy in Boyan's chain domain with 13 and 98 states. Each curve is the average of 30 independent trials.



(a) Puddle World (b) Mountain Car

Fig. 4. Comparative results in the simulated environments.

platform and manages to simulate the real environment with satisfactory precision and realism.

Two different grid maps (stage worlds) have been selected during our experiments, as shown in Fig.2(b),(c), each of which has its own peculiarities (different types of obstacles). The specific two worlds have been designed and edited by using the Mapper tool kit. The objective of the robot in these tasks is to find a steady landmark (shown with a rectangular green box in each map of Fig.2(b),(c)) with the minimum number of steps, starting from any position in the world and performing a finite number of actions. The particular tasks are episodic and a new episode starts when one of the following incidents comes first: the maximum allowed number of steps per episode is expired (in our case was set to 100), an obstacle is hit, or the target is reached. The state space consists of two continuous variables: the x and y coordinates which specify the situation of the robot in the world. At each time step, the robot receives an immediate reward of -1, except in the case that an obstacle is hit where the received reward is -100. The action space has been discretized into the 8 major compass winds, while the length of each step was set equal to $1m$.

The comparative results on the two worlds are illustrated in Fig. 5, where each plot represents mean returns received by the agent in the last 100 episodes. In the case of LSPI algorithm, we have used an equidistant fixed 10×10 grid of Gaussian RBFs is used over the state spaces (800 RBFs are used). As it becomes obvious, the proposed methodology achieves to discover an optimal policy in a much higher rate. On the other hand, our method don't need a so huge number of basis function as the online LSPI do, in order to discover an optimal policy. More specifically, the proposed algorithm constructs approximately $300 - 400$ clusters in both world stages, that specify the basis functions of our model. Finally, Figs. 6,7 represents the learned policies of both methods after 500 episodes.

V. Conclusion

In this study we have presented a model-based reinforcement learning scheme for learning optimally in MDPs. The proposed method is based on online partitioning the state-action space into clusters and simultaneously constructing a Markov transition matrix by counting the observed transitions that each cluster conformation undergoes over time steps. It is our intention to further pursue and develop the method in two directions: At first we can employ regularized least-squares methods, such as Lasso or a Bayesian sparse methodologies to eliminate the problem of overfitting. Also, during our experiments we have observed a tendency of our method to produce large number of clusters, some of them may become inactivated. Thus, a mechanism for merging clusters to the body of the online EM procedure constitutes an interesting direction for future work.

References

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

[2] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2010.

[3] J. A. Boyan, "Technical update: Least-squares temporal difference learning," *Machine Learning*, pp. 233–246, 2002.

[4] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[5] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Online least-squares policy iteration for reinforcement learning control," in *Proceedings of the 2010 American Control Conference*, 2010, pp. 486–491.

[6] G. Taylor and R. Parr, "Kernelized value function approximation for reinforcement learning," in *International Conference on Machine Learning*, 2009, pp. 1017–1024.

[7] X. Xu, H. Hu, and B. Dai, "Adaptive sample collection using active learning for kernel-based approximate policy iteration," in *Proceedings of the Adaptive Dynamic Programming and Reinforcement Learning*, 2011, pp. 56–61.

[8] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with gaussian process," in *International Conference on Machine Learning*, 2005, pp. 201–208.

[9] G. Konidaris, S. Osentoski, and P. Thomas, "Value function approximation in reinforcement learning using the fourier basis," in *AAAI Conf. on Artificial Intelligence*, 2011, pp. 380–385.

[10] S. Mahadevan and M. Maggioni, "Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes," *Journal of Machine Learning Research*, vol. 8, pp. 2169–2231, 2007.

[11] I. Menache, S. Mannor, and N. Shimkin, "Basis Function Adaptation in Temporal Difference Reinforcement Learning," *Annals of Operations Research*, vol. 134, pp. 215–238, 2005.

[12] M. Petrik, "An analysis of laplacian methods for value function approximation in mdps," in *International Joint Conference on Artificial Intelligence*, 2007, pp. 2574–2579.

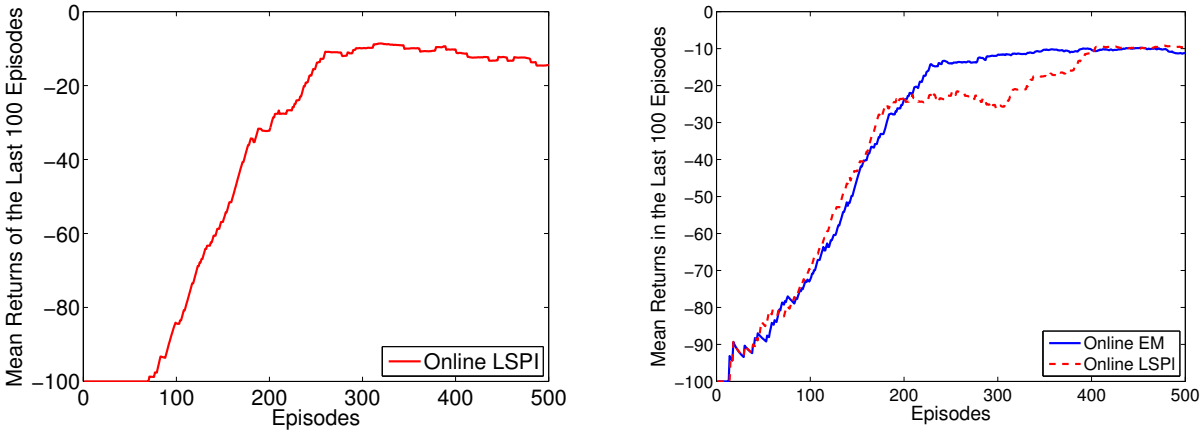
[13] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from incomplete data via the EM algorithm," *J. Roy. Statist. Soc. B*, vol. 39, pp. 1–38, 1977.

[14] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] R. Neal and G. Hinton, "A view of the em algorithm that justifies incremental, sparse, and other variants," in *NATO Adv. Study Inst. on Learn. Graph. Models*, 1998, pp. 355–368.

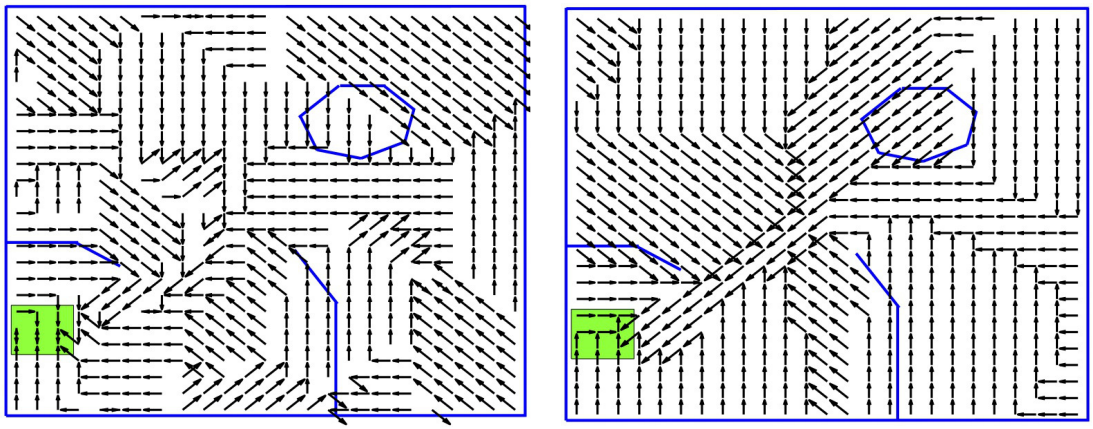
[16] B. R. Leffler, M. L. Littman, and T. Edmunds, "Efficient reinforcement learning with relocatable action models," in *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, 2007, pp. 572–577.

[17] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems 8*. MIT Press, 1996, pp. 1038–1044.



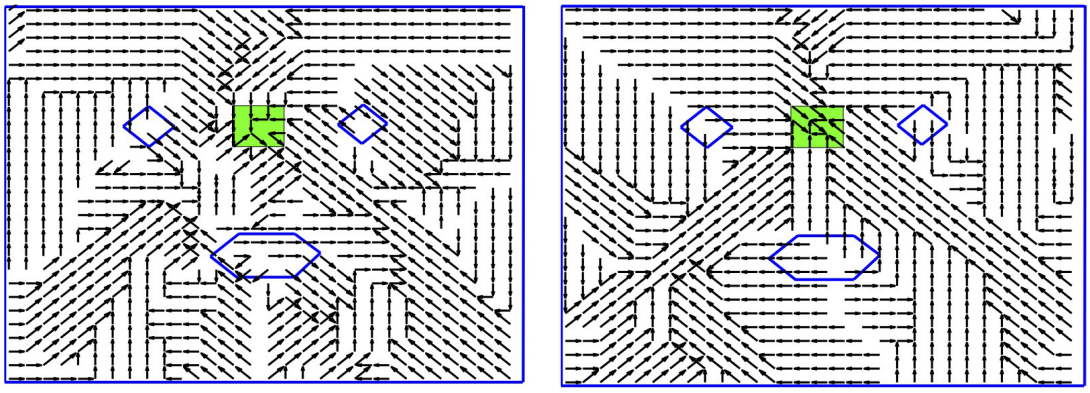
(a) Stadium S_1 (b) Stadium S_2

Fig. 5. Comparative results in two simulated environments.



(a) Online EM (b) Online LSPI

Fig. 6. Learned policies by both comparative methods in the case of test world S_1.



(a) Online EM (b) Online LSPI

Fig. 7. Learned policies by both comparative methods in the case of test world S_2.