A Data-based Online Reinforcement Learning Algorithm with High-efficient Exploration

Yuanheng Zhu* and Dongbin Zhao†
The State Key Laboratory of Management and Control for Complex Systems,
Institution of Automation, Chinese Academy of Sciences, Beijing, China
*Email: yuanheng.zhu@gmail.com
†Email: dongbin.zhao@ia.ac.cn

***Abstract*—An online reinforcement learning algorithm is proposed in this paper to directly utilizes online data efficiently for continuous deterministic systems without system parameters. The dependence on some specific approximation structures is crucial to limit the wide application of online reinforcement learning algorithms. We utilize the online data directly with the kd-tree technique to remove this limitation. Moreover, we design the algorithm in the Probably Approximately Correct principle. Two examples are simulated to verify its good performance.**

I. Introduction

Online reinforcement learning (RL) draws a lot of attention both from the computer science [1]–[4] and optimal control science [5]–[9], because it uses the online data to achieve an optimal policy through the interaction with the environment. Compared to the offline reinforcement learning, the efficient usage of online data, or the trade-off of exploration and exploitation becomes more critical. Besides, some issues related to practical implementation, e.g. the convergence rate and the obtained optimality, are also taken into consideration. Lots of efforts [10]–[18] have been devoted to solve such problems from different aspects.

Among many studies to overcome these problems, the probably approximately correct (PAC) is one of the most effective approaches. In the running process of an online learning algorithm, if the sum of steps when it implements non-optimal actions is finite and bounded, then it is called a PAC algorithm. Considering finite Markov Decision Processes (MDPs) with finite states, a lot of PAC algorithms have been proposed, including E^3 [19], RMAX [20], MBIE [21], Delayed Q-learning [22], etc.

For recent years, many researchers have concentrated on continuous-state systems to solve online optimal control problems in the PAC principle. Bernstein and Shimkin [23] propose the ARL algorithm for continuous deterministic systems. They prove their algorithm has a determinate finite time bound. But, the implementation requires some parameters of systems. So it is partially dependent on system information, which limits its application.

In this paper, we consider the optimal control problem of continuous deterministic systems and propose an online data-based RL algorithm. Without relying on any specific approximation structure, the online data are used directly. A kd-tree technique is adopted. The implementation is based on the current collected data and is applicable for arbitrary control problems. As we adopt kd-tree and consider the PAC principle on continuous-state systems, we term our algorithm as kd-CPAC.

The paper is organized as follows. Section II introduces the background for RL and Section III describes the kd-tree technique adopted in the algorithm. The whole process of kd-CPAC algorithm is presented in Section IV. We simulate two examples in Section V to verify the performance of our algorithm. The end is our discussion and conclusion.

II. Formulation of Online Reinforcement Learning

A continuous-state system with deterministic transition function can be represented by a 4-tuple, (S, A, R, F), where S is a continuous state space, A is a discrete action set, $R(s,a)$ is the reward function, and $F(s,a)$ is the deterministic transition function. Suppose the state space is bounded, not infinitely extended. The reward function also has an interval, namely $r_{min} \leq R(s,a) \leq r_{max}$. Note that in this case, R and F are both unknown to algorithms. So the only available information is online observations (s,a,r,s'), where r is the received reward and s' is the next-step state at (s,a).

During the interaction with the environment, we assume that at time t, the agent has experienced a history of states and actions, denoted by

$$h_t = \{s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t\}.$$

In the online case, the policy is *non-stationary* as algorithms can modify it at any moment. So actions are selected following a series of policies $\pi = \{\pi_t\}_{t=0}^{\infty}$, namely $a_t = \pi_t(s_t)$.

To evaluate the performance of a policy, we adopt the *discounted return criterion*. Given a policy π and an initial state $s_0 = s$, the discounted return is defined as

$$J^{\pi}(s) \triangleq \sum_{t=0}^{\infty} \gamma^t r_t \,\big|_{s_0=s, a_t=\pi_t(s_t)}$$

where γ is the *discount factor* satisfying $0 < \gamma < 1$. Note that in some systems, agents may stop and get stuck at some *terminal states*. Then the discounted return in this case is modified to

$$J^{\pi}(s) \triangleq \sum_{t=0}^{T-1} \gamma^t r_t + V(s_T) \,\big|_{s_0=s, a_t=\pi_t(s_t)}$$

where s_T indicates the terminal state and $V(s_T)$ is a predefined value to estimate the success or failure at s_T.

978-1-4799-4552-8/14/$31.00 ©2014 IEEE

The target of RL is maximizing the value of $J^\pi(s)$ and the corresponding policy is called the *optimal policy*, $\pi^* \triangleq \arg\max_\pi J^\pi$. Here, we adopt *optimal action-value function* or *optimal Q function* for the implementation of RL. It is defined in a Bellman principle as

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

and the optimal policy is generated by

$$\pi^*(s) = \arg\max_a Q(s,a).$$

Besides, as the reward function is bounded in an interval $[r_{min}, r_{max}]$, so the value function also has an upper bound, denoted by $V_{max} = \frac{r_{\max}}{1-\gamma}$.

As the information of systems is only from online observations, the storage of these samples is a major problem in the algorithm. Next, we will give a brief description about kd-tree, which is used to store online samples.

III. KD-TREE FOR THE STORAGE OF SAMPLES

Kd-tree, as an efficient approach to split the state space and store data, has been applied widely in the field of RL [24], [25].

Suppose a sample is denoted by $(\hat{s}, \hat{a}, \hat{r}, \hat{s}')$. We take $\hat{s}$ as the key to refer the sample. For convenience, each action corresponds to a kd-tree and there are $|A|$ kd-trees– $|A|$ indicates the number of actions. At the beginning, each tree has an empty root, which occupies the whole state space. When samples arrive, they are stored in the root. When the number of stored samples reaches a maximum number N_{split}– *split condition*, the space of the root is split into two nodes by a *split hyperplane* at *split dimension*. At the same time, the N_{split} samples are also divided separately into two children. This process will continue if more samples arrive and the depth of the tree will increase larger and larger.

The split dimension and hyperplane is determined by the following principle. Calculate the variance of each dimension among the samples in the node which is going to be split. For better comparison, states are normalized by the span of the state space before the calculation. The dimension corresponding to the maximum variance is selected as the split dimension. Then choose the median value at this dimension among samples as the split hyperplane.

When a new sample $(\hat{s}, \hat{a}, \hat{r}, \hat{s}')$ is required to be added in kd-trees, search the kd-tree of action $\hat{a}$ for the leaf which $\hat{s}$ belongs to. Then put the sample in the leaf node.

For arbitrary states, it is also convenient to find their neighboring samples in kd-trees. Given a state s, the *neighboring samples* of s refer to the set of samples $(\hat{s}, \hat{a}, \hat{r}, \hat{s}')$ which satisfy $d(s, \hat{s}) \leq \delta$. d is a metric $d : S \times S \to \mathbb{R}$ to specify the distance between two states, and δ is the *neighboring distance*. Start from the root and estimate the area of each node if it is close to s within δ. If not, the children and the included samples are also far away and there is no need to look into them. Otherwise, if the distance between the node and s is less than δ, and the node has children, then continue to estimate each child in the same way until reaching leaves. Compare the samples in the leaves with s and output those whose distances are less than δ as neighboring samples.

Based on the principle of storing samples with kd-tree, we present the kd-CPAC algorithm in the following section.

IV. KD-TREE BASED CONTINUOUS PAC ALGORITHM

A. Data set

Based on the previous section, suppose the current time is t and we have a data set $D_t = \{(\hat{s}_i, \hat{a}_i, \hat{r}_i, \hat{s}'_i)\}$ stored in kd-trees, in which are selective samples of the past time. As $\hat{r}_i$ and $\hat{s}'_i$ are determined by $\hat{s}_i$ and $\hat{a}_i$, so we can simplify the expression of a sample by the pair $(\hat{s}_i, \hat{a}_i)$. Or just $\hat{s}_i$ if given $\hat{a}_i$.

Separate samples at different actions into different sets. Use $D_t(a)$ to represent the set of samples belonging to a. Furthermore, for an arbitrary state s at action a, we construct a *neighboring set*–$C_t(s,a)$, to include the neighboring samples $(\hat{s}_i, a, \hat{r}_i, \hat{s}'_i)$ in $D_t(a)$ that have $d(s, \hat{s}_i) \leq \delta$, where δ is the predefined neighboring distance. If there exists no such samples, we declare $C_t(s,a) = \varnothing$. So $C_t(s,a)$ indicates the set of samples that is δ-close to s and we can further use them to approach (s,a).

B. Data-Based Q Iteration

Based on the data set, we can utilize it to define a *Data-Based Q Iteration* (DBQI) operator.

Definition 1 (DBQI operator) Given a function $g : S \times A \to \mathbb{R}$ and arbitrary s and a, the DBQI operator $\mathcal{T}$ is defined as

$$\mathcal{T}(g)(s,a) = \begin{cases} \min\limits_{\hat{s}_i \in C_t(s,a)} \left[\hat{r}_i + \gamma \max\limits_{a'} g(\hat{s}'_i, a')\right], & \text{if } C_t(s,a) \neq \varnothing \\ V_{\max}, & \text{otherwise} \end{cases} \tag{1}$$

where $(\hat{s}_i, a, \hat{r}_i, \hat{s}'_i)$ denotes the neighboring samples of (s,a) if $C_t(s,a)$ is not empty.

DBQI operator means for a pair (s,a), if its $C_t(s,a)$ is empty, we assign the value of (s,a) with the upper bound of value function, $V_{\max}$. Otherwise, we use the neighboring samples in $C_t(s,a)$ to approach its value, more concretely, the minimum one corresponding to the right side of the equation in (1). It is obvious that the calculation of DBQI operator is totally based on the stored samples.

We can prove $\mathcal{T}$ is a contraction operator, so there exists a fixed solution that has $\hat{Q}_t = \mathcal{T}(\hat{Q}_t)$. $\hat{Q}_t$ is called *Data-Based Q Function* (DBQF). To calculate $\hat{Q}_t$, Value Iteration (VI) [26], [27] or Policy Iteration (PI) [28], [29] can be used.

Based on value iteration, to calculate $\hat{Q}_t$, we first initialize a function $\hat{Q}_t^{(0)}$ which can be assigned to any value. Usually $\hat{Q}_t^{(0)}$ is equal to 0 or $V_{\max}$. Then calculate the Q values of stored samples $(\hat{s}, \hat{a}, \hat{r}, \hat{s}') \in D_t$ by

$$\hat{q}_t^{(0)}(\hat{s}, \hat{a}) = \hat{r} + \gamma \max_{a'} \hat{Q}_t^{(0)}(\hat{s}', a').$$

Furthermore, a new $\hat{Q}_t^{(1)}$ can be obtained by

$$\hat{Q}_t^{(1)}(s,a) = \begin{cases} \min\limits_{\hat{s}_i \in C_t(s,a)} \hat{q}_t^{(0)}(\hat{s}_i, a), & \text{if } C_t(s,a) \neq \varnothing \\ V_{\max}, & \text{otherwise} \end{cases}$$

The above equation is totally equal to the process of calculating $\hat{Q}_t^{(1)}$ from $\hat{Q}_t^{(0)}$ by (1). Then, this calculation is iterated.

In conclusion, suppose we have $\hat{Q}_t^{(j)}$ of the j-th iteration, calculate Q values of stored samples using

$$\hat{q}_t^{(j)}(\hat{s}, \hat{a}) = \hat{r} + \gamma \max_{a'} \hat{Q}_t^{(j)}(\hat{s}', a').$$

Then $\hat{Q}_t^{(j+1)}$ at the $(j+1)$-th iteration is obtained by

$$\hat{Q}_t^{(j+1)}(s,a) = \begin{cases} \min\limits_{\hat{s}_i \in C_t(s,a)} \hat{q}_t^{(j)}(\hat{s}_i, a), & \text{if } C_t(s,a) \neq \varnothing \\ V_{\max}, & \text{otherwise} \end{cases}$$

As the above process is a variant of solving (1) by value iteration, so it is convergent and the result is the same with directly calculating $\hat{Q}_t$ by value iteration. Moreover, in the process, the only need is storing Q values of samples, and the values of $\hat{Q}_t$ over the whole state space are easy to obtain.

With $\hat{Q}_t$, a greedy policy is extracted and applied to the system online to obtain a new observation at the next step

$$\pi_t(s) = \arg\max_a \hat{Q}_t(s,a). \tag{2}$$

C. Known vs Unknown

The next issue is whether to add the new observation into D_t or not. At the beginning of the algorithm, D_0 is empty. As the implementation progresses, some observations are added in D_t, while some are omitted to avoid the data set increasing infinitely. The principle is only storing the samples that have useful information about systems. So a definition of known is given here.

Definition 2 (Known) Given an observation (s, a, r, s'). If $C_t(s,a) \neq \varnothing$ and there exists a sample $(\hat{s}_i, a_i, \hat{r}_i, \hat{s}_i') \in C_t(s,a)$ such that $\left| \max\limits_{a_1'} \hat{Q}_t(s', a_1') - \max\limits_{a_2'} \hat{Q}_t(\hat{s}_i', a_2') \right| \leq \varepsilon_K$, then the observation is *known*. Otherwise, we say it is *unknown*. The parameter ε_K is called *known error*.

When a new observation arrives, we determine if it is known or unknown first. If known, we regard it with no useful information for our algorithm. If unknown, the observation contains some knowledge we have not known. Then it is added into the data set and $D_t \to D_{t+1}$. Update $\hat{Q}_t$ and π_t to $\hat{Q}_{t+1}$ and π_{t+1}.

At most cases of online problems, there exists an initial state s_0 and each episode has a fixed $T_{episode}$ length. At the beginning of each episode, the system is set to s_0 and after $T_{episode}$ steps, the episode ends and the state is reset. For this kind of systems, the whole process of kd-CPAC is presented in the following. Note that no parameters about systems are involved.

Algorithm 1 Kd-CPAC Algorithm

Require: value function upper bound $V_{\max}$
$|A|$ kd-trees
neighboring distance δ
known error ε_K
initialize $D_0 \leftarrow \varnothing$, $\hat{Q}_0 \leftarrow V_{\max}$ and $\pi_0(s) = \arg\max_a \hat{Q}_0(s,a)$
for $t = 0, 1, 2, \ldots$ **do**
  observe (s_t, a_t, r_t, s_t')
  if (s_t, a_t) is unknown in D_t **then**
    (s_t, a_t, r_t, s_t') is added into D_t
    update $\hat{Q}_t$ according to (1)
    produce π_t according to (2)
  end if
  execute π_t on the system
end for no change of D_t happens in an episode

V. Examples

In this section, we apply kd-CPAC to two different problems, Mountain Car and Inverted Pendulum. Mountain car is a 2-dimensional system with failure and success terminals. Inverted pendulum is also 2-dimensional but without terminals.

In the implementation, a distance metric d is required. Here, we choose d in a modified version of maximum norm. For two states s_1 and s_2, their distance is defined by

$$d(s_1, s_2) = \max_j \left| \frac{s_1^j - s_2^j}{S_{\sup}^j - S_{\inf}^j} \right|$$

where $S_{\sup}$ and $S_{\inf}$ is the upper and lower bound of the state space, and the superscript j indicates the j-th dimension. The value is normalized by $S_{\sup}$ and $S_{\inf}$, which is based on the same consideration in the calculation of variances when choosing split dimension in kd-tree.

A. Mountain car

The mountain car is a widely used system to test RL algorithms [24]. A schematic is illustrated in Fig. 1. At the beginning, the car is initialized at the bottom position ($p = -0.5$). By applying a horizontal force, the car can move left and right. The target is to reach the top of the mountain ($p = 1$). The system dynamics is denoted by

$$\ddot{p} = \frac{1}{1 + \left(\frac{\mathrm{d}H(p)}{\mathrm{d}p}\right)^2} \left(u - g \frac{\mathrm{d}H(p)}{\mathrm{d}p} - \dot{p}^2 \frac{\mathrm{d}H(p)}{\mathrm{d}p} \frac{\mathrm{d}^2 H(p)}{\mathrm{d}^2 p} \right)$$

where $p \in [-1, 1]$m is the horizontal position of the car, $\dot{p} \in [-3, 3]$m/s is its velocity, $u \in [-4, 4]$N is the horizontal force, $g = 9.81\text{m/s}^2$ is the gravitational acceleration, and H denotes the shape of the hill, defined as

$$H(p) = \begin{cases} p^2 + p, & \text{if } p < 0 \\ \frac{p}{\sqrt{1+5p^2}}, & \text{if } p \geq 0 \end{cases}.$$

In the simulation, the state variable is $s = [p, \dot{p}]^T$ and the action set is discretized by $A = \{-4, 4\}$. Whenever the car passes the left edge ($p = -1$) or its velocity is over 3 ($|\dot{p}| > 3$), we regard it as a failure and stop the driving. The success

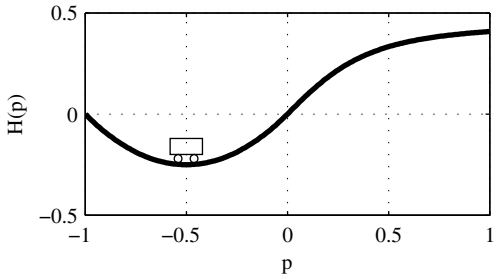

Fig. 1. A schematic of mountain car system.

TABLE I. MOUNTAIN CAR: LEARNED POLICIES' PERFORMANCE OF KD-CPAC AT DIFFERENT KNOWN ERRORS WHEN $\delta = 0.01$

ε_H	1.0	3.0	5.0	7.0	9.0
Length to success(s)	1.9	1.9	4.0	6.2	fail

condition is the car reaching the right edge ($p = 1$) within the speed limit ($|\dot{p}| \leq 3$). So the failure terminal is assigned with a low value, $V(s_{failure}) = -100$, while the success one with $V(s_{success}) = 0$. In the middle of the process, each step has a reward $r = -1$. The discount factor is $\gamma = 0.95$ and the sample time is 0.1s. The episode length $T_{episode}$ is set to 10s and the initial state $s_0 = [-0.5, 0]^T$. The split condition N_{split} chooses 20.

First, we fix the value of neighboring distance δ as 0.01 and study the impact of different known errors ε_H on the final learned policies of kd-CPAC. To evaluate a policy, the time length in an episode to success is adopted as its performance. The results are illustrated in Table I. Viewed from the tendency, it is concluded that smaller known errors lead to more optimal policies. This is consistent to our theoretical results. Furthermore, if the known error is too large, the policy can fail to move the car to the goal like the last experiment in Table I.

Then, we fix known error ε_H to 1.0 and examine the influence of neighboring distance δ. Similarly, we can conclude from Table II that a large neighboring distance leads to a bad policy. These two groups of experiments are consistent to our theoretical results that the smaller values δ and ε_H choose, the learned policy is more optimal.

Next, let $\delta = 0.02$ and $\varepsilon_H = 1.0$ and observe the process of kd-CPAC. After 100 trials of running, the algorithm stops and a total of 1216 samples are stored. The stored samples at each action are presented in Fig. 2, combined with the partitions of state space by the leaves in kd-trees. These figures illustrate that kd-tree can efficiently store samples for our algorithm. Apply the learned policy to the system starting from the initial state and the trajectories are depicted in Fig. 3. It is revealed that after 1.9 seconds, the car successfully reaches the goal. Besides, the policy is so efficient that only one turn of actions in the episode leads to the success.

TABLE II. MOUNTAIN CAR: LEARNED POLICIES' PERFORMANCE OF KD-CPAC AT DIFFERENT NEIGHBORING DISTANCES WHEN $\varepsilon_H = 1.0$

δ	0.01	0.015	0.02	0.025	0.03
Length to success(s)	1.9	1.9	1.9	2.0	fail

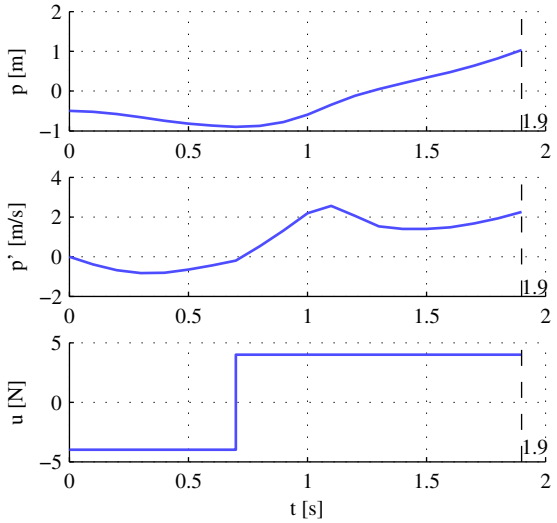

Fig. 3. Mountain Car: Trajectories of states and actions under the learned policy of kd-CPAC with $\delta = 0.02$ and $\varepsilon_H = 1.0$.

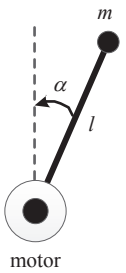

Fig. 4. A schematic of the inverted pendulum.

B. Inverted pendulum

In the second simulation, we adopt the inverted pendulum. It is a common example to estimate online algorithms [2].The inverted pendulum is a device that rotates a mass in a vertical plane and is driven by a DC motor. A schematic is presented in Fig. 4 and its dynamics can be denoted by

$$\ddot{\alpha} = \frac{1}{J}\left(mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right)$$

where α and $\dot{\alpha}$ are the angle and angular velocity of the pendulum, satisfying the bound $[-\pi, \pi)$ rad and $[-15\pi, 15\pi]$ rad/s respectively. u is the control action applied to the DC motor and constrained to $[-3, 3]$V. For simulation, dynamics parameters are adopted the same as [2], given in Table III and the sample time is set to 0.01s.

The goal is to swing up the pendulum from the bottom and balance it at the top. So the state input is $s = [\alpha, \dot{\alpha}]^T$ and the control action is discretized into 3 discrete values, $A = \{-3, 0, 3\}$. The reward is designed by $r(s, a) = -s^T Q s$, where $Q = diag(5, 0.1)$. The discount factor is set $\gamma = 0.98$. The episode length $T_{episode}$ is 6s and each trial starts from $[\pi, 0]^T$.

In this experiment, we still choose $N_{split} = 20$ but set $\delta = 0.005$ and $\varepsilon_K = 30.0$. After 143 episodes of learning, the

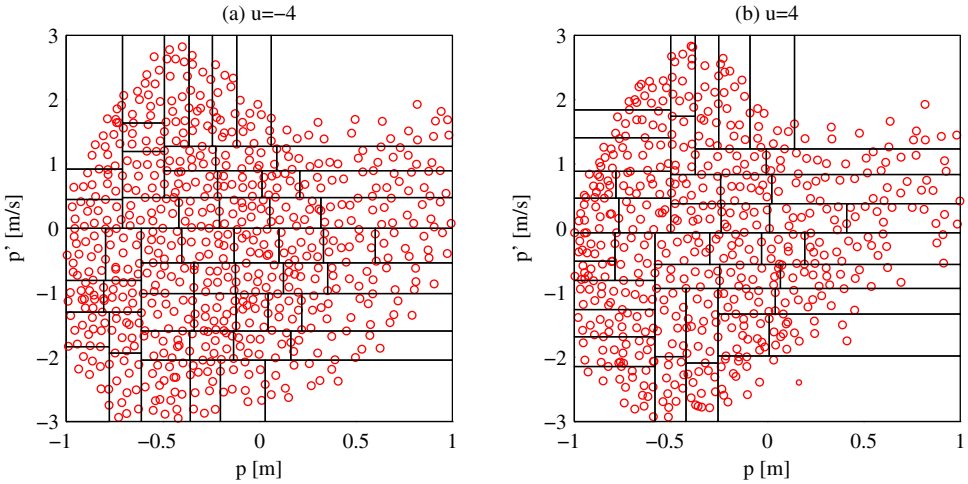

Fig. 2. Mountain Car: Partitions of state space and the stored samples at each action of kd-trees with $\delta = 0.02$ and $\varepsilon_H = 1.0$.

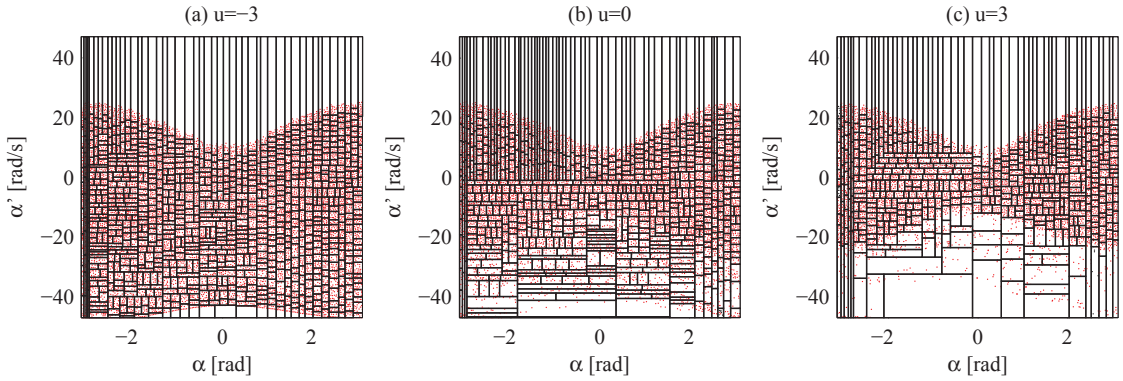

Fig. 5. Inverted Pendulum: Partitions of state space and the stored samples at each action of kd-trees with $\delta = 0.005$ and $\varepsilon_K = 30.0$.

TABLE III. PARAMETERS OF INVERTED PENDULUM

Symbol	Value	Meaning
m	0.055	mass
g	9.81	gravitational acceleration
l	0.042	distance from centre to mass
J	1.91e-4	moment of inertia
b	3e-6	viscous damping
K	0.0536	torque constant
R	9.5	rotor resistance

algorithm stops and it stores 29684 samples. Stored samples in three kd-trees are depicted in Fig. 5. As the problem is complicated, more samples are stored and the state space is partitioned by kd-trees to smaller sizes. Next, the learned policy is applied to the system to observe its performance. For comparison, an offline model-based Fuzzy Q-Iteration from [2] is also applied to the same system. In its implementation, we set triangular fuzzy partitions with 51 equidistant cores for both state variables. After the offline learning, a convergent policy is obtained. We implement these two policies with respect to kd-CPAC and Fuzzy Q-Iteration to inverted pendulum and their results are illustrated in Fig. 6. By comparison, it is obvious that the policy of kd-CPAC algorithm (blue solid lines) has a better performance than Fuzzy Q-Iteration algorithm (green dashed lines), as kd-CPAC policy needs less steps to swing up and balance the pendulum. So even our algorithm is online and has no information about the system, the learned policy is still more optimal.

VI. CONCLUSION

In this paper, we consider continuous deterministic systems and propose a new online RL algorithm, kd-CPAC. During the online running, the algorithm selectively stores samples and utilizes them directly to produce policies. These policies are prone to explore unvisited areas, which helps to collect system information.

To avoid the dependence on approximation structure in the implementation, we utilize the online data directly. It benefits the algorithm with high efficiency of online data. To store samples, a kd-tree technique is adopted. It helps to divide the state space according to samples and store them in a tree structure. Based on kd-tree, it is convenient for the algorithm to locate samples and search for neighboring samples for arbitrary states.

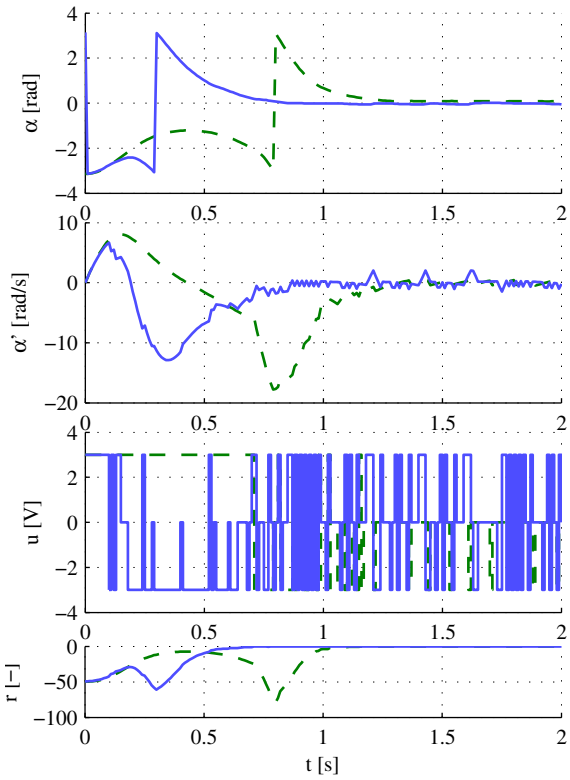

Fig. 6. Inverted pendulum: Trajectories of states, actions and rewards. The blue solid lines indicate the policy of kd-CPAC, while the green dashed lines refer to the policy of Fuzzy Q-Iteration with 51×51 triangular fuzzy partitions.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (NSFC) under Grants No. 61273136, No. 61034002, and Beijing Natural Science Foundation under Grant No. 4122083.

REFERENCES

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

[2] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. New York: CRC Press, 2010.

[3] A.-H. Tan, Y.-S. Ong, and A. Tapanuj, "A hybrid agent architecture integrating desire, intention and reinforcement learning," *Expert Syst. Appl.*, vol. 38, no. 7, pp. 8477–8487, 2011.

[4] L. Tang, Y.-J. Liu, and S. Tong, "Adaptive neural control using reinforcement learning for a class of robot manipulator," *Neural Comput. Appl.*, pp. 1–7, 2013.

[5] D. Wang, D. Liu, D. Zhao, Y. Huang, and D. Zhang, "A neural-network-based iterative GDHP approach for solving a class of nonlinear optimal control problems with control constraints," *Neural Comput. Appl.*, vol. 22, no. 2, pp. 219–227, 2013.

[6] Q. Wei and D. Liu, "Stable iterative adaptive dynamic programming algorithm with approximation errors for discrete-time nonlinear systems," *Neural Comput. Appl.*, vol. 24, no. 6, pp. 1355–1367, 2014.

[7] B. Wang, D. Zhao, C. Alippi, and D. Liu, "Dual heuristic dynamic programming for nonlinear discrete-time uncertain systems with state delay," *Neurocomputing*, vol. 134, pp. 222–229, 2014.

[8] Q. Yang and S. Jagannathan, "Reinforcement learning controller design for affine nonlinear discrete-time systems using online approximators," *IEEE Trans. Syst. Man Cybern. B*, vol. 42, no. 2, pp. 377–390, April 2012.

[9] H. Zhang, L. Cui, and Y. Luo, "Near-optimal control for nonzero-sum differential games of continuous-time nonlinear systems using single-network adp," *IEEE Trans. Cybern.*, vol. 43, no. 1, pp. 206–216, Feb 2013.

[10] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge Univ., Cambridge, U.K., 1989.

[11] S. ten Hagen and B. Kröse, "Neural Q-learning," *Neural Comput. Appl.*, vol. 12, no. 2, pp. 81–88, 2003.

[12] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge University Engineering Department, Cambridge, England, Tech. Rep. TR 166, 1994.

[13] D. Liu, D. Wang, D. Zhao, Q. Wei, and N. Jin, "Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using globalized dual heuristic programming," *IEEE Trans. Automat. Sci. Eng.*, vol. 9, no. 3, pp. 628–634, July 2012.

[14] S. B. Thrun, "The role of exploration in learning control," in *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. White and D. Sofge, Eds. Florence, Kentucky 41022: Van Nostrand Reinhold, 1992.

[15] D. Zhao, Z. Hu, Z. Xia, C. Alippi, and D. Wang, "Full range adaptive cruise control based on supervised adaptive dynamic programming," *Neurocomputing*, vol. 125, pp. 57–67, 2014.

[16] D. Zhao, B. Wang, and D. Liu, "A supervised actor-critic approach for adaptive cruise control," *Soft Computing*, vol. 17, no. 11, pp. 2089–2099, 2013.

[17] D. Zhao, X. Bai, F. Wang, J. Xu, and W. Yu, "DHP for coordinated freeway ramp metering," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 990–999, 2011.

[18] X. Bai, D. Zhao, and J. Yi, "The application of ADHDP(λ) method to coordinated multiple ramps metering," *International Journal of Innovative Computing*, vol. 5, no. 10(B), pp. 3471–3481, 2009.

[19] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Mach. Learn.*, vol. 49, no. 2-3, pp. 209–232, Nov. 2002.

[20] R. I. Brafman and M. Tennenholtz, "R-max - a general polynomial time algorithm for near-optimal reinforcement learning," *J. Mach. Learn. Res.*, vol. 3, pp. 213–231, Mar. 2003.

[21] A. L. Strehl and M. L. Littman, "A theoretical analysis of model-based interval estimation," in *Proc. 22nd Int. Conf. Machine Learning (ICML'05)*, 2005, pp. 856–863.

[22] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC model-free reinforcement learning," in *Proc. 23rd Int. Conf. Machine Learning (ICML'06)*, 2006, pp. 881–888.

[23] A. Bernstein and N. Shimkin, "Adaptive-resolution reinforcement learning with polynomial exploration in deterministic domains," *Mach. Learn.*, vol. 81, no. 3, pp. 359–397, Dec. 2010.

[24] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Mach. Learn.*, vol. 49, no. 2-3, pp. 291–323, Nov. 2002.

[25] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, Dec. 2005.

[26] H. Li and D. Liu, "Optimal control for discrete-time affine nonlinear systems using general value iteration," *IET Control Theory and Applications*, vol. 6, no. 18, pp. 2725–2736, 2012.

[27] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof," *Trans. Sys. Man Cyber. Part B*, vol. 38, no. 4, pp. 943–949, Aug. 2008.

[28] D. Liu, X. Yang, and H. Li, "Adaptive optimal control for a class of continuous-time affine nonlinear systems with unknown internal dynamics," *Neural Comput. Appl.*, vol. 23, no. 7-8, pp. 1843–1850, 2013.

[29] L. Zuo, X. Xu, C. Liu, and Z. Huang, "A hierarchical reinforcement learning approach for optimal path tracking of wheeled mobile robots," *Neural Comput. Appl.*, vol. 23, no. 7-8, pp. 1873–1883, 2013.