# Hierarchical Approximate Policy Iteration with Binary-Tree State Space Decomposition

Xin Xu, *Member, IEEE*, Chunming Liu, Simon X. Yang, *Senior Member, IEEE*, and Dewen Hu, *Senior Member, IEEE*

***Abstract*— In recent years, approximate policy iteration (API) has attracted increasing attention in reinforcement learning (RL), e.g., least-squares policy iteration (LSPI) and its kernelized version, the kernel-based LSPI algorithm. However, it remains difficult for API algorithms to obtain near-optimal policies for Markov decision processes (MDPs) with large or continuous state spaces. To address this problem, this paper presents a hierarchical API (HAPI) method with binary-tree state space decomposition for RL in a class of absorbing MDPs, which can be formulated as time-optimal learning control tasks. In the proposed method, after collecting samples adaptively in the state space of the original MDP, a learning-based decomposition strategy of sample sets was designed to implement the binary-tree state space decomposition process. Then, API algorithms were used on the sample subsets to approximate local optimal policies of sub-MDPs. The original MDP was decomposed into a binary-tree structure of absorbing sub-MDPs, constructed during the learning process, thus, local near-optimal policies were approximated by API algorithms with reduced complexity and higher precision. Furthermore, because of the improved quality of local policies, the combined global policy performed better than the near-optimal policy obtained by a single API algorithm in the original MDP. Three learning control problems, including path-tracking control of a real mobile robot, were studied to evaluate the performance of the HAPI method. With the same setting for basis function selection and sample collection, the proposed HAPI obtained better near-optimal policies than previous API methods such as LSPI and KLSPI.**

***Index Terms*— Adaptive dynamic programming, approximate policy iteration, binary-tree, hierarchical reinforcement learning, Markov decision processes, time-optimal control.**

## I. Introduction

REINFORCEMENT learning (RL) is a machine learning framework for solving sequential decision problems that can be modeled as Markov decision problems (MDPs). In recent years, RL has been widely studied in the neural network community and in operations research [1], [2].

Manuscript received December 17, 2009; revised August 23, 2011; accepted August 25, 2011. Date of publication October 10, 2011; date of current version December 1, 2011. This work was supported in part by the National Natural Science Foundation of China under Grant 61075072, Grant 60835005, and Grant 90820302, the Fork Yingtong Youth Teacher Foundation under Grant 114005, the Program for New Century Excellent Talents in University and the National Fundamental Research Program of China under Grant 2007CB311001.

X. Xu, C. Liu, and D. Hu are with the College of Mechatronics and Automation, National University of Defense Technology, Changsha 410073, China (e-mail: xinxu@nudt.edu.cn; lccmmm@126.com; dwhu@nudt.edu.cn).

S. X. Yang is with the School of Engineering, University of Guelph, Guelph, ON N1G 2W1, Canada (e-mail: syang@uoguelph.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2011.2168422

In RL, the learning agent interacts with an initially unknown environment and modifies its action policies to maximize its cumulative payoffs. Thus, RL provides an efficient framework for solving learning control problems that are difficult or even impossible for supervised learning and mathematical programming methods. However, despite some empirical successes, including backgammon [3], job-shop scheduling [4], elevator scheduling [5], and helicopter flight control [6], it remains difficult for RL to solve MDPs with large or continuous spaces. In such cases, many RL algorithms cannot converge to a good near-optimal policy and require numerous training samples. RL algorithms have difficulty with dimensionality, the exponential growth of the number of parameters to be learned with the size of any compact encoding of system states [7].

To improve the generalization ability of RL algorithms requires the study of RL theories and algorithms based on approximate value functions or policies. Until recently, the three main categories of approximate RL methods have included value function approximation (VFA) [8], [9], policy search [10], and actor-critic methods [11], [12]. Among these approximate RL methods, VFA has been widely studied. According to the basic properties of function approximators, there are two different kinds of VFA methods: linear [8], [12] and nonlinear VFA [3], [9], [13]. Although RL with nonlinear VFA exhibits better approximation ability than linear VFA, the empirical results of RL applications using nonlinear VFA commonly lack a rigorous theoretical analysis and the nonlinear features are usually determined by manual selection, e.g., the structures of multiple layer perceptrons [14], [15]. The actor-critic algorithms [11] are another class of RL methods for MDPs with continuous spaces. Unlike VFA-based methods, actor-critic algorithms approximate the value functions and policies of an MDP separately to encourage the realization of generalization in MDPs with continuous spaces. Many recent studies of actor-critic methods have focused on adaptive critic designs, which usually require an approximated model of the plant dynamics.

As a popular method studied in operations research, policy iteration can be viewed as an actor-critic method because the value functions and policies are approximated separately. Approximate policy iteration (API) methods have been studied in recent years to solve MDPs with large or continuous spaces. In [16], a model-free API algorithm called least-squares policy iteration (LSPI) was presented, offering a RL method with better properties in convergence, stability, and sample complexity than previous RL algorithms. Nevertheless, the approximation structure in LSPI may lead to degenerated performance when improperly selecting features. In [17], a kernel-based LSPI (KLSPI) algorithm was presented for MDPs with large or continuous state spaces, but the final policies of LSPI and KLSPI are greatly influenced by the approximation precision of value functions and the selected basis functions.

Recent attempts to overcome the obstacles associated with dimensionality have turned to principled ways of problem decomposition or exploiting temporal abstraction, which are called hierarchical approaches to RL [18]. As indicated in [19], existing work in hierarchical RL (HRL) has followed three trends: focusing on subsets of the state space in a divide and conquer approach (state space decomposition) [20], grouping sequences or sets of actions together (temporal abstraction) [21], and ignoring differences between states based on the context (state abstraction) [22]–[24]. These three trends have led to three main approaches to HRL: options formalism [17], the hierarchies of abstract machines approach [25], and the MAXQ framework [20], in which the model of semi-Markov decision processes is commonly used as a formal basis [26]. Although function approximators can be combined with HRL, few successful applications of existing HRL approaches to MDPs with large or continuous spaces have occurred. In addition, it remains difficult to decompose the state space of MDPs automatically or construct options so that global optimal policies can be well approximated.

This paper proposes a novel hierarchical API (HAPI) approach with binary-tree state space decomposition for RL in a class of episodic MDPs with terminal states, formulated as absorbing MDPs for goal-directed time-optimal control tasks. Although we focus primarily on deterministic MDPs, we also discuss the extension of HAPI to stochastic cases and test it empirically in the experiments. The learning control objective is to find an optimal path to the goal state with minimal state transition steps. In the proposed approach, by decomposing the original MDP into multiple sub-MDPs with smaller state spaces, multiple API algorithms can be used on the sub-MDPs to obtain better near-optimal local policies. Then, the final global policy can be derived by combining the local policies in each sub-MDP. After integrating the binary-tree state space decomposition method with two typical API algorithms, this paper presents the hierarchical LSPI algorithm (HLSPI) and the hierarchical KLSPI algorithm (HKLSPI) for absorbing MDPs. We analyze the hierarchical optimality of the combined policy in HLSPI and HKLSPI and illustrate that higher approximation precision of value functions can be realized for policy evaluation in sub-MDPs. Based on the theoretical results on performance error bounds in API, better performance can be obtained for every local policy in HAPI. Thus, the final combined policy of HLSPI or HKLSPI achieves better performance than previous LSPI or KLSPI algorithms.

Compared to existing LSPI/KLSPI, HAPI provides a new hierarchical learning framework for API in absorbing MDPs. The major novelties of HAPI include an adaptive decomposition of absorbing MDPs and the approximation or achievement of the hierarchical optimality of the combined policy. Therefore, the improvement of local policies leads to a better combined policy. Another major novelty implements API algorithms in a hierarchical way to achieve higher approximation precision of policy evaluation in sub-MDPs to reduce the performance errors of local policies and improve the combined policy performance significantly because of its hierarchical optimality. Extensive experimental results illustrate the superiority of HLSPI/HKLSPI over existing LSPI/KLSPI.

Section II provides an introduction on MDPs and the previous API algorithms. Section III presents the HAPI method and analyzes the hierarchical optimality of HAPI. This section also provides the performance error bounds of local and combined near-optimal policies. In Section IV, a deterministic chain MDP and a stochastic chain MDP demonstrate the improved precision of HAPI-based VFA. Simulation on the mountain-car problem and the experimental results of the path-tracking of a real mobile robot illustrate the effectiveness of the proposed method. Section V draws conclusions and suggests some future work.

## II. Short Overview of MDP and API

### A. Markov Decision Processes

An MDP $M$ is denoted as a tuple $\{X, A, R, P\}$, where $X$ is the state space, $A$ is the action space, $P$ is the state transition probability, and $R$ is the reward function. A stochastic stationary policy $\pi$ (or just stationary policy) maps states to distributions over the action space. When referring to such a policy $\pi$, we use $\pi\ (a|x)$ to denote the probability of selecting action $a$ in state $x$ by $\pi$. A deterministic stationary policy directly maps states to actions, denoted as

$$a_t = \pi(x_t) \qquad t \geq 0. \tag{1}$$

A deterministic policy defined by (1) can be viewed as a special case of a stochastic stationary policy, where $\pi(a|x) = 1$ for $a = a_t$ and $\pi(a|x) = 0$ for other actions. When the actions $a_t$ ($t \geq 0$) satisfy (1), policy $\pi$ is followed in the MDP $M$. A stochastic stationary policy $\pi$ is said to be followed in the MDP $M$ if $a_t \sim \pi(a\,|x_t),\ t \geq 0$.

The objective of a decision maker is to estimate the optimal policy $\pi*$ satisfying

$$J_{\pi^*} = \max_{\pi} J_{\pi} = \max_{\pi} E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{2}$$

where $0< \gamma < 1$ is the discount factor and $r_t$ is the reward at time-step $t$, $E^{\pi}[\bullet]$ stands for the expectation with respect to the policy $\pi$ and the state transition probabilities, and $J_{\pi}$ is the expected total reward along the state trajectories by following policy $\pi$. In this paper, $J_{\pi}$ is also called the performance value of policy $\pi$.

The state-action value function $Q^{\pi}(x,a)$ is defined as the expected, discounted total rewards when taking action $a$ in state $x$ and following policy $\pi$ thereafter:

$$Q^{\pi}(x, a) = E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\,|x_0 = x, a_0 = a\right]. \tag{3}$$

The action value function $Q^{\pi}(x,a)$ satisfies the following Bellman equation [7]:

$$Q^{\pi}(x_t, a_t) = E^{\pi}\left[r(x_t, a_t) + \gamma\, Q^{\pi}(x_{t+1}, a_{t+1})\right]. \tag{4}$$

For an MDP, a deterministic optimal policy $\pi^*(x)$ maximizes the expected, discounted total reward of state $x$

$$\pi^*(x) = \arg\max_a Q^{\pi^*}(x, a). \tag{5}$$

A state $x$ of an MDP is a terminal (or absorbing) state if the process never leaves after entering: $x_{t+1} = x$ holds provided that $x_t = x$, regardless of the action selected at time $t$. An MDP with terminal states is episodic. In this paper, we focus on episodic MDPs with large or continuous state spaces.

### *B. API*

For MDPs with large or continuous state spaces, computing the state-action value function $Q^{\pi}(x,a)$ for each state-action pair is impractical. To realize generalization, the LSPI algorithm has been widely studied in recent years [13]. In LSPI, the state-action value function $Q^{\pi}(x,a)$ can be approximated using a linearly weighted combination of $n$ basis functions

$$\hat{Q}^{\pi}(x, a) = \vec{\phi}^T(x, a)W \tag{6}$$

where $W = (w_1, w_2, \ldots, w_n)^T$ is the weight vector and $\vec{\phi}(x, a)$ is the basis function vector, denoted by

$$\vec{\phi}(x, a) = (\phi_1(x, a), \phi_2(x, a), \ldots, \phi_n(x, a))^T. \tag{7}$$

Let

$$\Phi = \begin{pmatrix} \vec{\phi}^T(x_1, a_1) \\ \vec{\phi}^T(x_2, a_2) \\ \vdots \\ \vec{\phi}^T(x_m, a_m) \end{pmatrix}, \quad \Phi' = \begin{pmatrix} \vec{\phi}^T(x'_1, a'_1)) \\ \vec{\phi}^T(x'_2, a'_2)) \\ \vdots \\ \vec{\phi}^T(x'_m, a'_m)) \end{pmatrix},$$

$$R = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{pmatrix}$$

where $D = \{(x_i,a_i, r_i, x'_i,a'_i)|i = 1,2,\ldots,m\}$ is a set of collected samples from an initial policy $\pi[0]$ and $a'_i \sim \pi[0](a|x'_i)$.

Then, the least-squares fixed-point solution for action VFA [27] and the corresponding improved policy can be obtained as follows [13]:

$$\begin{cases} W^{\pi[t]} = \left(\Phi^T\left(\Phi - \gamma\,\Phi'\right)\right)^{-1}\Phi^T R \\ \pi\,[t+1](x) = \arg\max_a \vec{\phi}^T(x, a)\,W^{\pi[t]} \qquad t = 0, 1, \ldots \end{cases} \tag{8}$$

The KLSPI algorithm presented in [17] describes basis functions by kernel-based features: $\{k(x, x_j)\}$ $(j = 1,2,\ldots, m)$, where $x_j$ is a selected sample state and $k(.,.)$ is a Mercer kernel function. For any finite set of points $\{x_1, x_2,\ldots, x_m\}$, the kernel matrix $K = [k(x_i, x_j)]_{m\times m}$ is positive definite. According to the Mercer Theorem [28], a Hilbert space $H$ and a mapping $\varphi$ from $X$ to $H$ exist, such that

$$k(x_i, x_j) =< \varphi(x_i), \varphi(x_j) > \tag{9}$$

where $< \cdot,\cdot >$ is the inner product in $H$. Although the dimension of $H$ may be infinite and the nonlinear mapping $\varphi$ is usually unknown, all the computation in the feature space can still be performed if it is in the form of inner products.

As discussed in [17], KLSPI can be viewed as a kernelized version of LSPI. In KLSPI, a kernel sparsification process is designed based on the approximately linear dependence (ALD) analysis originally proposed in [29]. By using the kernel methods, KLSPI has better performance than LSPI in learning control tasks with continuous state spaces. Moreover, kernel-based features can be automatically obtained based on the ALD analysis for kernel sparsification.

According to the theoretical results in [30], the performance errors of API are upper-bounded by the approximation errors in policy evaluation steps, making it necessary to develop new API methods with reduced errors in policy evaluation.

## III. HAPI WITH BINARY-TREE STATE SPACE DECOMPOSITION

The HAPI approach proposed in this paper has two distinctive features compared with previous HRL and API methods. One is the binary-tree state space decomposition of absorbing MDPs, performed during the learning process. In the proposed HAPI method, by using the learned value functions, the original MDP automatically decomposes into smaller sub-MDPs. More importantly, as discussed subsequently, the hierarchical optimality of the combined policy can be ensured or approximated. Second, by applying API algorithms on the sub-MDPs, the approximation precision of value functions and the quality of local near-optimal policies improve. Because of the hierarchical optimality, the combined global policy performs better with the improvement of local policies. We analyze the performance of HAPI theoretically and verify it empirically.

### *A. Binary-Tree Decomposition of Absorbing MDPs*

Because a broad class of optimal control problems can be modeled as episodic MDPs with goal states, or absorbing MDPs, we focus on absorbing MDPs for goal-directed optimal control tasks. Assuming that the intermediate rewards for non-terminal states are much smaller than the terminal rewards, this paper seeks an optimal policy to reach the terminal states with minimal state transition steps. The MDPs studied in this paper are a class of shortest-path problems. To simplify notations, we focus on deterministic absorbing MDPs and the extensions to stochastic MDPs with terminal states will be discussed in Section III-C. Specifically, in Section IV, we illustrate that the proposed HAPI approach can solve both deterministic and stochastic shortest-path problems.

Consider a deterministic absorbing MDP denoted by $M_0$. Let $X_T$ denote the set of terminal states. The reward function of $M_0$ is assumed as

$$r(x_t, a_t, x_{t+1}) = \begin{cases} r_T, & x_{t+1} \in X_T \\ r, & x_{t+1} \notin X_T \end{cases} \tag{10}$$

where $r_T - r \geq \xi > 0$, and $\xi$ is a large positive number.

In this section, we present the binary-tree state space decomposition strategy for deterministic absorbing MDPs. We first introduce the idea of state space decomposition based on the expected costs to the absorbing states, assuming a known optimal value function. We also present the approach for state space decomposition using approximated value functions.

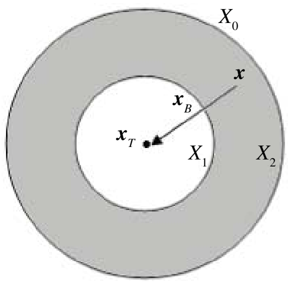


Fig. 1. Illustration of the state space decomposition procedure based on optimal value functions.

*1) State Space Decomposition for Absorbing MDPs:* For a deterministic absorbing MDP $M_0$, a direct strategy for state space decomposition divides the state space into sub-spaces or sub-MDPs based on the expected total costs or rewards for reaching the absorbing states. We adopt the notations $X_i$ and $M_i$ to denote the sub-spaces and sub-MDPs, respectively. Let $V^*(x)$ be the optimal value function of state $x$. Based on the optimal value function, an optimal state trajectory or path with the maximum cumulative reward can be found from any state to the goal state. Let $P(x, x_T) = \{x, x_1, x_2, \ldots, x_T\}$ denote the optimal state trajectory or path from any state $x$ to the goal state $x_T$. The length of $P(x, x_T)$ is denoted as $L(x, x_T)$.

According to the reward function defined in (9), in which all the intermediate rewards have the same value and the terminal reward is much larger than the intermediate rewards, if we assume $0 < \gamma < 1$, it can be inferred that

$$\text{if } L(x, x_T) > L(x^{'}, x_T) \quad \text{then} \quad V^*(x) < V^*(x^{'}). \tag{11}$$

The state space $X_0$ of $M_0$ decomposes into two sub-sets by comparing the optimal value function of each state with a threshold $\mu$

$$X_0 = X_1 \cup X_2 \tag{12}$$

where

$$X_1 = \{x \,|V^*(x) \geq \mu\} \tag{13}$$

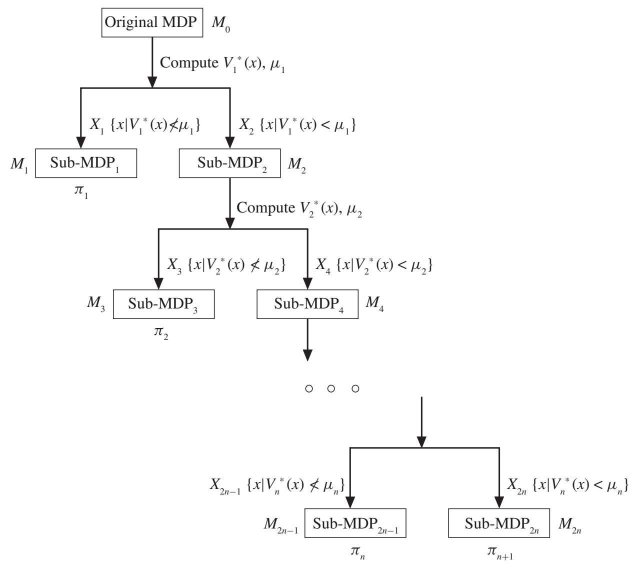
$$X_2 = \{x \,|V^*(x) < \mu\}. \tag{14}$$

By projecting the optimal state trajectory $L(x, x_T)$ onto a 2-D plane, Fig. 1 illustrates the preceding decomposition procedure, where the original state space $X_0$ is decomposed into two parts, $X_1$ (white area) and $X_2$ (gray area). The states in $X_1$ have fewer state transition steps to $x_T$ than the states in $X_2$. According to the Bellman optimality principle, $L(x, x_T)$ from a state in $X_2$ can be decomposed into two sub-paths $L(x, x_B)$ and $L(x_B, x_T)$ in $X_2$ and $X_1$, respectively. The state $x_B$ can be viewed as a boundary state between $X_1$ and $X_2$ given that $x_B$ belongs to $X_1$, and there is a state transition on the optimal path from a state in $X_2$ to $x_B$, or

$$\exists x \to x_B, \;\; x \in X_2, \; x_B \in X_1.$$

*2) Definition of the Boundary States and Sub-MDPs:* As illustrated in Fig. 1, the boundary set $B_{(1,2)}$ between $X_1$ and $X_2$ is defined as

$$B_{(1,2)} = \left\{x \,\middle|\, x \in X_1, \exists x' \to x, x' \in X_2\right\} \tag{15}$$



Original MDP
$M_0$
Compute $V_1^*(x)$, $\mu_1$
$X_1$ $\{x|V_1^*(x) \nless \mu_1\}$
$X_2$ $\{x|V_1^*(x) < \mu_1\}$
$M_1$
Sub-MDP$_1$
Sub-MDP$_2$
$M_2$
$\pi_1$
Compute $V_2^*(x)$, $\mu_2$
$X_3$ $\{x|V_2^*(x) \nless \mu_2\}$
$X_4$ $\{x|V_2^*(x) < \mu_2\}$
$M_3$
Sub-MDP$_3$
Sub-MDP$_4$
$M_4$
$\pi_2$
○ ○ ○
$X_{2n-1}$ $\{x|V_n^*(x) \nless \mu_n\}$
$X_{2n}$ $\{x|V_n^*(x) < \mu_n\}$
$M_{2n-1}$
Sub-MDP$_{2n-1}$
Sub-MDP$_{2n}$
$M_{2n}$
$\pi_n$
$\pi_{n+1}$

Fig. 2. Binary-tree state space decomposition procedure.

where $x' \to x$ denotes a state transition from $x'$ to $x$ and this state transition is on the optimal trajectory from $x'$ to $x_T$.

By using the same state transition model, $X_1$ and $X_2$ can be viewed as the state spaces of two sub-MDPs of $M_0$. Let $M_1$ and $M_2$ denote the two sub-MDPs in state space $X_1$ and $X_2$, respectively. For any state $x$ in sub-MDP $M_2$, if the optimal state transition path is from $x$ to a state in the boundary set $B_{(1,2)}$, $x$ is defined as a terminal state, and the reward function is redefined as

$$r(x, a, x') = \begin{cases} r_T, & x' \in B_{(1,2)} \\ r, & x' \notin B_{(1,2)}. \end{cases} \tag{16}$$

Then $M_2$ becomes an absorbing MDP with a smaller state space than $M_0$, and all the states in $M_1$ are near to the terminal states. As discussed in the following section, the approximation error of value functions has smaller upper bounds for states with fewer steps to the terminal states because a terminal reward determines the value functions of absorbing states exactly and directly. The value functions of intermediate states have to be approximated using the temporal-differences in state transitions, where LS-TD($\lambda$) and RLS-TD($\lambda$) algorithms are widely used [12], [27]. Therefore, using the approximated near-optimal policy in sub-MDP $M_1$ as one of the local policies benefits the final global policy, and the sub-MDP $M_2$ is selected to perform a similar binary-tree decomposition of its state space. The optimal policy of $M_2$ will find a shortest state-transition path from any state to the absorbing state set $B_{(1,2)}$.

*3) Binary-Tree State Space Decomposition:* The process of state space decomposition for absorbing MDP $M_0$ can be iterated and illustrated using a binary-tree structure, as shown in Fig. 2. In each iteration, the optimal value function $V_i^*$ of $M_{2i}$ is computed, and a threshold $\mu_i$ is computed based on the optimal value function

$$\mu_i = \beta \cdot \max_{x \in M_{2i}} V_i^*(x) \tag{17}$$

where $\beta$ is a constant value.

The threshold $\mu_i$ decomposes the state space of $M_{2i}$ into two sub-spaces $X_{2i+1}$ and $X_{2i+2}$, where $X_{2i+1}$ usually has a smaller size than $X_{2i+2}$ and the sub-space $X_{2i+2}$ is used for state space decomposition in the next iteration. If $\mu_i$ becomes large, the number of sub-spaces and computational complexity increase. The threshold determined by (15) is a balance between the depth of the binary-tree and the computational complexity. So, a medium value of $\beta$ between 0.2 and 0.4 can be easily selected for the balance. In this paper, the value of $\beta$ is fixed as 0.3, and the preceding formula of $\mu_i$ is very effective in all experiments. Similar to the definition of sub-MDP $M_1$ in $X_1$, a sub-MDP $M_{2i-1}$ can be defined in sub-space $X_{2i-1}$, and $M_{2i}$ can also be defined using the same reward definition in (14).

The procedure depicted in Fig. 2 only considers the idea case where the optimal value function can be obtained or approximated exactly. In practical applications, because the real optimal value function $V^*(x)$ is unknown, the approximated optimal value function is used for state space decomposition by performing API algorithms on the original MDP. After the decomposition of $M_0$ and the definition of $M_2$, a similar state space decomposition process can be performed on $M_2$ using API to approximate the optimal value function of $M_2$. The iterative procedure terminates when the size of $X_{2n}$ is small enough or the depth of the binary-tree reaches a given number, obtaining a set of sub-spaces $\{X_1, X_2, X_3, \ldots, X_{2n}\}$. More importantly, during the decomposition process, local optimal policies $\tilde{\pi}_i^*$ can also be approximated for each sub-MDP $M_{2i-1}$. For example, after performing LSPI on $M_0$, the weight vector $W$ for approximating the optimal value function can also determine the near-optimal policy for $M_1$.

The approximated local optimal policy for sub-MDP $M_{2n}$ is denoted as $\tilde{\pi}_{n+1}^*$, which is computed by performing API on $M_{2n}$ directly. The final near-optimal policy for the original MDP can be produced by combining the local near-optimal policies as follows:

$$\tilde{\pi}^*(x) = \begin{cases} \tilde{\pi}_i^*(x) & x \in X_{2i-1}, 1 \le i \le n \\ \tilde{\pi}_{n+1}^*(x) & x \in X_{2n}. \end{cases} \tag{18}$$

## B. HAPI Algorithm

Given that no explicit model information for API algorithms usually exists, data samples from state transitions are collected by exploring the whole state space of the original MDP $M_0$. In HAPI, to divide an absorbing MDP into sub-MDPs, the data samples are partitioned into different subsets of data samples based on the approximated value functions. Moreover, the reward functions in different subsets are redefined according to (16). Therefore, the subsets of data samples appear as the samples generated in different sub-MDPs. Then, API algorithms obtain local near-optimal policies of the sub-MDPs with higher precision, and the global optimal policy can be approximated as a combination of the local near-optimal policies. Fig. 3 presents the main procedures in HAPI, including the following steps.

*1) Adaptive Sample Collection:* The first step for HAPI uses adaptive sample collection to collect samples as the input of the algorithm. Because the performance of API

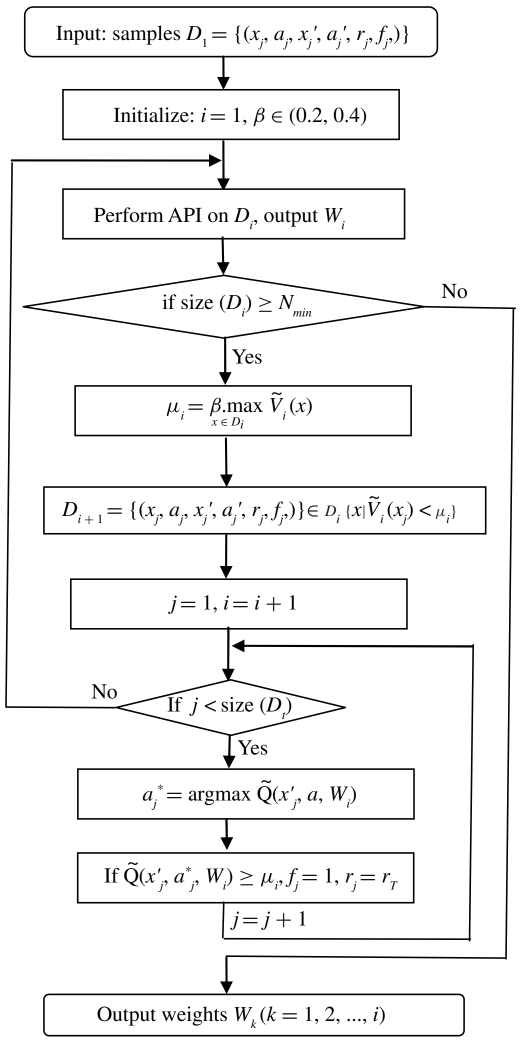

Input: samples $D_1 = \{(x_j, a_j, x_j', a_j', r_j, f_j)\}$
Initialize: $i = 1$, $\beta \in (0.2, 0.4)$
Perform API on $D_i$, output $W_i$
if size $(D_i) \ge N_{min}$
No
Yes
$\mu_i = \beta.\max_{x \in D_i} \tilde{V}_i(x)$
$D_{i+1} = \{(x_j, a_j, x_j', a_j', r_j, f_j)\} \in D_i \{x \mid \tilde{V}_i(x_j) < \mu_i\}$
$j = 1, i = i + 1$
No
If $j <$ size $(D_i)$
Yes
$a_j^* = \text{argmax}\ \tilde{Q}(x_j', a, W_i)$
If $\tilde{Q}(x_j', a_j^*, W_i) \ge \mu_i, f_j = 1, r_j = r_T$
$j = j + 1$
Output weights $W_k (k = 1, 2, ..., i)$

Fig. 3. Algorithm 1: the HAPI algorithm.

algorithms depends on the input set of collected samples, the sample collection process needs to explore the state and action space of an absorbing MDP uniformly. In previous studies of LSPI and KLSPI, samples were usually collected randomly. However, in some learning control tasks, when samples were collected with a stochastic policy, it took too many steps to reach the absorbing states, e.g., the Mountain-Car problem in the experimental section. In such cases, for sub-optimal policies, the samples near the goal or absorbing states will be visited with low frequencies, which hinder the success of both previous API algorithms and the HAPI method. In this paper, an adaptive sample collection method collects data samples and realizes a tradeoff between exploration and exploitation using online API during the sample collection process.

In the adaptive sample collection method, an online learning control loop is executed using API in every time step for sample collection, where an action selection strategy is used for the current state based on the outputs of online VFA. At time step $t$, an online API is implemented, where the data samples collected before time step $t$, $\{(x_i, a_i, x_i', a_i', r_i, f_i) | i = 1, 2, \ldots, t-1\}$, determine the policy. To realize a tradeoff between exploitation and exploration, we employed an epsilon-greedy action selection strategy. At each time step, a uniform random number is generated within the interval [0, 1], and a greedy action with the maximum action value is selected when the random number is greater than a given threshold $\eta$, where $\eta$ is a small positive number.

*2) Online Sample Set Decomposition:* Let $D_1 = (x_j, a_j, x_j^{'}, a_j^{'}, r_j, f_j)$ ($j = 1, 2, \ldots, N$, $f_j$ is the flag for terminal states) be the set of data samples collected in $M_0$ and the initial iteration number be $i = 1$. Let $D_i$ denote the sample set for sub-MDP $M_{2(i-1)}$. The first step in each iteration uses API to approximate the near-optimal policy of $M_{2(i-1)}$ using the data samples in $D_i$. After the convergence of API, a weight vector $W_i$ determines the value function of the near-optimal policy $\pi_i$. To decompose $M_{2(i-1)}$ into sub-MDPs iteratively, the estimated value functions in $M_{2(i-1)}$ partition the sample set $D_i$ into two subsets, where one subset $D_{i+1}$ is used for state space decomposition in the next iteration. Similar to (17), the threshold $\mu_i$ is automatically determined by the following equations:

$$\mu_i = \beta \cdot \max_{x \in D_i} \tilde{V}_i(x) \tag{19}$$

$$\tilde{V}_i(x) = \max_a \tilde{Q}_i(x, a, W_i) = \max_a \phi_i^T(x, a) W_i. \tag{20}$$

The subset $D_{i+1}$ for $M_{2i}$ is determined as follows:

$$D_{i+1} = \left\{ (x_j, a_j, x_j', a_j', r_j, f_j) \in D_i \,\middle|\, \tilde{V}_i(x_j) < \mu_i \right\}. \tag{21}$$

*3) Identification of Boundary States:* For every sample $(x_j, a_j, x_j^{'}, a_j^{'}, r_j, f_j)$ in $D_{i+1}$, a state is identified as a boundary state if the following condition is satisfied:

$$\tilde{Q}_i(x_j', a_j^*, W_i) \geq \mu_i \tag{22}$$

where

$$a_j^* = \arg\max_a \tilde{Q}_i(x_j', a, W_i). \tag{23}$$

A boundary state is a terminal state or absorbing state in $M_{2i}$, where the absorbing flag and reward are set as $f_j = 1$ and $r_j = r_T$.

The prior procedure is iterated until the number of remaining samples is smaller than a given number $N_{\min}$ or the iteration number $i$ reaches the maximum iteration number. When the size of $D_{2n}$ is no more than $N_{\min}$, the HAPI algorithm is completed, and a set of policies $\{\pi_1, \pi_2, \ldots, \pi_{n+1}\}$ is obtained, where $\pi_i$ is the local near-optimal policy of $M_{2i-1}$ ($1 \leq i \leq n$) using API and $\pi_{n+1}$ is the local near-optimal policy of $M_{2n}$. The final policy to be executed is a combination of the local policies $\{\pi_1, \pi_2, \ldots, \pi_{n+1}\}$.

*4) Termination Criterion and Policy Combination:* To terminate the HAPI algorithm, the minimal number of samples in $D_i$ can be selected, denoted by $N_{\min}$. For the selection of $N_{\min}$, the sample distribution and the maximal length from an initial state to the goal state influence it. If $N$ initial samples are uniformly collected from $M_0$ and the maximal trajectory length from an initial state to the goal state is $L$, an appropriate value of $N_{\min}$ can be selected between $10N/L$ and $30N/L$.

The termination criterion for a single API on a sample set $D_i$ is the maximum iteration number or the distance between two successive policies $\pi[i]$ and $\pi[i+1]$. The local near-optimal policies obtained in different iterations generate the real actions in the sub-spaces of the original MDP, i.e., the sub-MDPs. For the local near-optimal policy $\pi_i$, it will be the local policy to be executed in the state space of sub-MDP $M_{2i-1}$. Algorithm 2 describes the policy combination procedure.

**Algorithm 2** Policy combination of HAPI

```
1: Given:
   1) $x$, the current state;
   2) $\mu_i$, the threshold value determined in Algorithm 1;
   3) a set of local policies $\{\pi_1, \pi_2, \ldots, \pi_{n+1}\}$ determined by
      the weight vectors $W_i$ ($i = 1, 2, \ldots, n+1$)
2: $i = 1$.
3: While $i \leq n+1$.
   (3.1) Select the greedy action $a_i = \pi_i(x)$ according to $\pi_i$.
   (3.2) Compute
         $\tilde{V}^{\pi_i}(x) = \tilde{Q}^{\pi_i}(x, a_i, W_i)$.
   (3.3) If $\tilde{V}^{\pi_i}(x) \geq \mu_i$ or $i = n+1$, go to 4.
   (3.4) $i = i + 1$.
4: Return the selected action $a_i$.
```

In Algorithm 2, every local policy generates an action output according to its approximated value functions, but only one output will be the optimal action. Because the current state is within the state space of one sub-MDP $M_{2i-1}$ or $M_{2n}$, the selected action will be the output of local policy $\pi_i$ or $\pi_{n+1}$. In Algorithm 2, a loop is executed from $\pi_1$ to $\pi_{n+1}$, and the index $i$ of sub-MDP $M_{2i-1}$ or $M_{2n}$ can be determined when the value function $V^{\pi_i}(x)$ is greater than $\mu_i$ for the first time. If $i$ reaches the maximal value $n+1$, the state $x$ will be in the state space of $M_{2n}$ and the action output of $\pi_{n+1}$ will be selected.

## C. Performance Analysis and Discussions

In this section, the performance of HAPI will be analyzed based on the quality of final combined policies, measured in terms of the expected total rewards after performing the final policies along a complete state trajectory from starting states to terminal states. In particular, the performance errors between a near-optimal policy and the optimal policy will be considered, as measured by the differences between the total rewards received by following a near-optimal policy and the total rewards received by the optimal policy.

Generally speaking, the main advantage of HAPI over previous API algorithms is that the performance errors of final policies can be significantly reduced by decomposing the original MDP into sub-MDPs in a binary-tree structure and fusing the local policies together. In addition to state space decomposition, feature representation and sample complexity [31], [32] are two other factors for the performance of RL algorithms in MDPs with large or continuous state spaces. To illustrate the effectiveness of the binary-tree state space decomposition in HAPI, the performance comparison between HAPI and API uses the same feature representation and sample complexity. The performance evaluations and comparisons of different feature representation methods, such as kernel methods [17], neural networks [14], [15], [33], [34], and other tree-based RL methods, are beyond the scope of this paper. For the sample complexity problem, we consider the case of finite sample size, which is true for many real applications including robot control among others.

In the framework of HAPI, a local policy $\pi_i$ $(i = 1, 2,\ldots, n)$ is defined as the final near-optimal policy obtained after the convergence of a single API in a sub-MDP. The combined policy $\pi$ of HAPI is the policy obtained by Algorithms 1 and 2. It is also called a hierarchical policy composed of local policies in a hierarchy of sub-MDPs. In the following, the performance of HAPI is analyzed in three aspects. First, the performance error bounds of a local policy indicate that the performance error bounds of local policies can be reduced by performing API on smaller sub-MDPs with absorbing states. Second, the hierarchical optimality of the combined policy is verified so that the performance errors between the combined policy of HAPI and the best hierarchical policy are zero or very small. Therefore, with reduced performance errors in local policies, the performance errors between the combined policy in HAPI and the globally optimal policy can also be reduced.

*1) Performance Errors of a Local Policy in HAPI:* A local policy $\pi_i$ in HAPI is obtained by a single API using the samples with transformed reward from sub-MDP $M_i$. The performance error of the local policy is analyzed based on the existing results on performance error bounds of API. As described in [35], if the policy evaluation is accurate to within $\varepsilon$, API algorithms will yield a sequence of policies such that

$$\limsup_{m\to\infty} \left\| J^{\pi_m} - J^* \right\|_\infty \le \frac{2\gamma\,\varepsilon}{(1-\gamma)^2} \tag{24}$$

where $J^*$ is the performance value of the optimal policy and $J^{\pi_m}$ is the performance value of a policy in the $m$-th iteration.

The results in (24) provide performance bounds on the closeness to optimality of the approximate value functions obtained by successive policy improvement steps as a function of the maximum norm of value approximation errors during policy evaluation. However, because quadratic norms are commonly used as performance measures in TD learning algorithms for policy evaluation, the above results using maximum norms will have limited practical range. In [30], the performance error bounds were obtained for API algorithms using quadratic norms, where the main results are described by the following inequalities:

$$\limsup_{m\to\infty} \left\| J^{\pi_m} - J^* \right\|_\rho \le \frac{2\gamma}{(1-\gamma)^2} \limsup_{m\to\infty} \left\| \hat{V}^{\pi_m} - V^{\pi_m} \right\|_{\rho_m} \tag{25}$$

$$\limsup_{m\to\infty} \left\| J^{\pi_m} - J^* \right\|_\rho \le \frac{2\gamma}{(1-\gamma)^2} \limsup_{m\to\infty} \left\| \hat{V}^{\pi_m} - T^{\pi_m}\hat{V}^{\pi_m} \right\|_{\tilde{\rho}_m} \tag{26}$$

where $\rho$ is an arbitrary distribution on $X$ and $\rho_m$ $(m = 1, 2,\ldots)$ are distributions related to $\rho$ and the policies $\pi_m$ and $\pi^*$.

Given that in API, the approximation error between the estimated value function and the real value function cannot be computed directly, the Bellman residual errors are estimated as

$$\Delta r^\pi(x) = r\left(x, x'\right) + \gamma \hat{V}^\pi\left(x'\right) - \hat{V}^\pi(x) \tag{27}$$

where $x'$ is the successive state of state $x$ and $r(x, x')$ is the reward.

In [36], an error bound between the least-squares fixed-point solution of LS-TD learning and the real value function was derived. Furthermore, in [37], an unbiased estimation of a modified loss function for Bellman residual minimization (BRM) was obtained and when linearly parameterized functions are used, both LS-TD and the BRM-based approach converged to the same solution, minimizing the modified BRM loss function. When high precision of VFA is realized for the fixed-point solution of LS-TD, the Bellman residual error is likely bounded. In the following, an upper bound of the Bellman residual errors for policy evaluation in API algorithms is defined as:

$$\Delta R^\pi = \max_{x_t} \left|\Delta r^\pi(x_t)\right|. \tag{28}$$

Let the approximation error of the value function be

$$\Delta V^\pi(x_t) = V^\pi(x_t) - \hat{V}^\pi(x_t). \tag{29}$$

In the policy evaluation process of API, for a stationary policy $\pi$, the approximation error of value functions satisfies

$$\Delta V^\pi(x_t) = \Delta r^\pi(x_t) + \gamma\,\Delta V^\pi(x_{t+1}). \tag{30}$$

For MDPs with absorbing states, a sample trajectory with length $N$ consists of the following state transitions:

$$x_1 \xrightarrow{r_1} x_2 \xrightarrow{r_2} \cdots \xrightarrow{r_t} x_t \xrightarrow{r_{t+1}} \cdots \xrightarrow{r_{N-1}} x_N \tag{31}$$

where $x_N$ is an absorbing state. The number of state transition steps from an intermediate state $x_t$ to the absorbing state is $N - t$.

Based on (13), the following relation holds:

$$\begin{aligned} \left|\Delta V^\pi(x_t)\right| &= \left|\Delta r^\pi(x_t) + \gamma\,\Delta V^\pi(x_{t+1})\right| \\ &\le \left|\Delta r^\pi(x_t, a_t)\right| + \gamma\left|\Delta V^\pi(x_{t+1}, a_{t+1})\right| \\ &\le \Delta R^\pi + \left|\Delta V^\pi(x_{t+1})\right| \end{aligned} \tag{32}$$

$$\begin{aligned} \sum_{i=t}^{N-1}\left|\Delta V^\pi(x_i)\right| &\le (N-t)\Delta R^\pi + \sum_{i=t+1}^{N}\left|\Delta V^\pi(x_i)\right| \\ \left|\Delta V^\pi(x_t)\right| &\le (N-t)\Delta R^\pi + \left|\Delta V^\pi(x_N)\right|. \end{aligned} \tag{33}$$

Because the value function of the absorbing states can be updated by the real terminal rewards, the approximation error of the value function in $x_N$ can be made as small as possible, that is

$$\left|\Delta V^\pi(x_N)\right| \approx 0. \tag{34}$$

Then, the approximation errors of the value functions for non-absorbing states have upper bounds as follows:

$$\left|\Delta V^\pi(x_t)\right| \le (N-t)\Delta R^\pi. \tag{35}$$

Using these results, we infer that the approximation errors of states near the absorbing states can be lower than those states with more transition steps to the absorbing states. In the proposed HAPI method, with the binary-tree decomposition of state spaces and the reward transformation mechanism, the state spaces of the sub-MDPs are all near the absorbing states. Then, smaller approximation errors of action value functions can be realized for local policies. More importantly, because of the smaller state space of sub-MDPs, the LS-TD algorithms can have smaller Bellman residuals in each sub-MDP

$$\Delta R_i^\pi < \Delta R_{all}^\pi \tag{36}$$

where $\Delta R_i^\pi$ and $\Delta R_{all}^\pi$ are the Bellman residuals of TD learning in the sub-MDP and the whole MDP, respectively.

According to this analysis, the approximation errors of value functions for local policies in HAPI can be reduced by two factors: the average number of steps to the absorbing states and the possible reduction of Bellman residuals. Thus, based on the results in (25) and (26), the performance errors of the local policies in HAPI can also be reduced.

*2) Hierarchical Optimality of the Combined Policy:* The notion of hierarchical optimality for a hierarchical policy was introduced in [20]. Let $J(x)$ be the expected cumulative reward along the trajectory starting from $x$. Suppose an MDP $M = \{X, A, R, P\}$ has a hierarchy consisted of sub-MDPs $M_1$, $M_2$, ..., $M_n$. Formal definitions of a hierarchically optimal policy and a recursively optimal policy follow.

*Definition* 1 (*Hierarchical Optimality*)*:* Denote $\Pi = \pi^M = (\pi_{1j}, \pi_{2j}, \ldots, \pi_{nj})\}$ $(j = 1, 2, \ldots)$ as the set of all hierarchical policies consistent with the given hierarchy, where $\pi_{ij}$ is a local policy of $M_i$. If there is a hierarchical policy $\pi^* \in \Pi$, such that

$$\forall x \in X \quad \forall \pi^M \in \Pi,\ J^{\pi^*}(x) \geq J^{\pi^M}(x) \tag{37}$$

then $\pi^*$ is called a hierarchically optimal policy.

*Definition* 2 (*Recursive Optimality*)*:* Denote $\pi^M = (\pi_1, \pi_2, \ldots, \pi_n)$ as a hierarchical policy consistent with the given hierarchy, if for $i = 1, 2, \ldots, n$, $\pi_i$ is a local optimal policy of $M_i$, $\pi^M$ is called a recursively optimal policy.

Based on this definition, a hierarchically optimal policy $\{\pi_1, \pi_2, \ldots, \pi_{n+1}\}$ for $M$ achieves the highest cumulative reward among all policies consistent with the given hierarchy. The recursive optimality is a kind of local optimality in which the policy at each node is optimal given the policies of its children, meaning that recursive optimality is mainly due to the design of the hierarchy. When the sub-tasks are properly defined, recursive optimality becomes hierarchical optimality. In MAXQ, the task or state space decomposition is usually carried out by the designer and based on the assumption that the programmer identifies useful subgoals and defines subtasks that achieve these subgoals. MAXQ converges, with probability 1, to a recursively optimal policy. Furthermore, [19] explained that when the reward or task decomposition was properly defined, MAXQ can also find a hierarchically optimal policy.

The HAPI approach decomposes the state space during the learning process, and under certain assumptions, the hierarchical policy defined by the binary-tree decomposition in Fig. 1 can be guaranteed to be hierarchically optimal. In the following, for a deterministic absorbing MDP $M_0$, the reward function of which is defined by (9), if $\pi_i^*$ $(1 \leq i \leq n)$ is the optimal policy in sub-MDP $M_{2i-1}$, and $\pi_{n+1}^*$ is the optimal policy of $M_{2n}$, the hierarchical policy $\pi = \{\pi_1^*, \pi_2^*, \ldots, \pi_{n+1}^*\}$ is hierarchically optimal or the performance error between $\pi$, and the hierarchically optimal policy is bounded.

For any state $x \in X_{2n}$, using the hierarchical policy $\pi$, a state transition trajectory $T(x, x_T) = \{x, x_1, x_2, \ldots, x_T\}$ will be generated from $x$ to the absorbing state $x_T$. Although the starting state $x$ can also be in other sub-spaces, we assume $x \in X_{2n}$ in the sequel given that the analysis is similar.

Let $X_{2i-1}$ $(1 \leq i \leq n)$ be the sub-spaces, and let $B_{(i,i+1)}$ denote the boundary set of two adjacent sub-MDPs, $X_{2i-1}$ and $X_{2i+1}$. Define $B_{(n,n+1)}$ as the boundary set between $X_{2n-1}$ and $X_{2n}$ and $B_{(0,1)}$ as the absorbing states $x_T$ in $M_0$.

Because $\pi = \{\pi_1^*, \pi_2^*, \ldots, \pi_{n+1}^*\}$ is a hierarchical policy consistent with the binary-tree state space decomposition, a set of connected sub-trajectories $\{T_{n+1}, T_n, \ldots, T_1\}$ of $T(x, x_T)$ in every sub-MDP or sub-space can be defined, where $T_i$ $(1 \leq i \leq n)$ connects the states in boundary set $B_{(i,i+1)}$ to the states in $B_{(i-1,i)}$, and $T_{n+1}$ is the trajectory in $M_{2n}$ starting from $x$ to the states in $B_{(n,n+1)}$.

Let the total reward of $T(x, x_T)$ be denoted by $J(T)$. Then

$$J(T) = \sum_{i=1}^{n} J(T_i) + V^*(x) \tag{38}$$

where $J(T_i)$ is the total reward for each sub-trajectory $T_i$, and $V^*(x)$ is the optimal value function of $x$ in sub-MDP $M_{2n}$.

Let $\pi' = \{\pi_1', \pi_2', \ldots, \pi_n' \pi_{n+1}'\}$ denote the hierarchical optimal policy that achieves the highest cumulative reward among all policies consistent with the given hierarchy, where $\pi_i' (1 \leq i \leq n)$ is also the local optimal policy of $M_{2i-1}$ and $\pi_{n+1}'$ is the optimal policy of $M_{2n}$. According to $\pi'$, a state trajectory $T'(x, x_T)$ from $x$ to $x_T$ can be obtained. Similarly, we also define a set of connected sub-trajectory $\{T_{n+1}', T_n', \ldots, T_1'\}$ of $T'(x, x_T)$ in every sub-MDP. $T_i'$ $(1 \leq i \leq n)$ connects the boundary states in $B_{(i,i+1)}$ to the states in$B_{(i-1,i)}$, and $T_{n+1}'$ is the trajectory in $M_{2n}$ starting from $x$ to the states in $B_{(n,n+1)}$. Then, the total reward of $T'$ becomes

$$J(T') = \sum_{i=1}^{n} J(T_i') + V^*(x). \tag{39}$$

Because $T_i$, $T_i'$ $(i = 1, 2, \ldots, n)$ are the state transition trajectories starting from a boundary state in $B_{(i,i+1)}$, and $\pi_i^* (1 \leq i \leq n)$ is the optimal policy in sub-MDP $M_{2i-1}$, $J(T_i)$, and $J(T_i')$ equals the optimal values of the boundary states in $B_{(i,i+1)}$.

For the boundary states $x_i$ in $B_{(i,i+1)}$, we consider two cases: first, the case in which the boundary states have the same optimal value $V^*(x_i) = \mu_i$, and second, that the case in which there is a small interval for the optimal values of the boundary states.

*Case 1:* For $x_i \in B_{(i,i+1)}$, $V^*(x_i) = \mu_i$.

In this case, because of the uniqueness of the optimal value function in the boundary set, we obtain

$$\begin{aligned} J(T_i) &= J(T_i') = V^*(x_i), \quad i = 1, 2, \ldots, n \\ J(T) &= J(T') = \sum_{i=1}^{n} J(T_i') + V^*(x). \end{aligned} \tag{40}$$

Therefore, $\pi = \{\pi_1^*, \pi_2^*, \ldots, \pi_{n+1}^*\}$ achieves the highest cumulative reward among all policies consistent with the given hierarchy.

*Case 2:* For $x_i \in B_{(i,i+1)}$, $V^*(x_i)$ is not unique.

According to the definition of boundary states, a state transition occurs from $x'$ to $x_i$, where $V^*(x') < \mu_i$, $V^*(x_i) \geq \mu_i$. Given that $V^*(x') = r + \gamma V^*(x_i)$, and $r_T$ are much greater than the absolute value of $r$, $\mu_i$ will also be much greater than the absolute value of $r$. Then, we obtain

$$V^*(x') = r + \gamma V^*(x_i) < \mu_i \tag{41}$$

$$V^*(x_i) < \frac{(\mu_i - r)}{\gamma} \approx \frac{\mu_i}{\gamma} \tag{42}$$

$$\mu_i \leq V^*(x_i) < \frac{\mu_i}{\gamma}. \tag{43}$$

Given that a discount factor near 1 is usually used in RL, a small interval for the optimal values of boundary states occurs, i.e., for $x_i \in B_{(i,i+1)}$, $V^*(x_i)$ is near the value of $\mu_i$. Suppose two boundary states $x_1$ and $x_2$ exist, where $x_1 \in T_i$, $x_2 \in T_i'$. Because $J(T_i) = V^*(x_1)$ and $J(T_i') = V^*(x_2)$, the performance error bound between $T_i$ and $T_i'$ is

$$\left|J(T_i) - J(T_i')\right| = \left|V^*(x_1) - V^*(x_2)\right| < \frac{\mu_i(1-\gamma)}{\gamma}. \tag{44}$$

Then, the performance error bound between $T$ and $T'$ is

$$\left|J(T) - J(T')\right| < \sum_{i=1}^{n} \frac{\mu_i(1-\gamma)}{\gamma}. \tag{45}$$

Note that the previous bound is based on the worst case analysis. In practice, $n$ is a small number, and $\gamma$ is near 1. The performance error bound is thus also small. In [20], it is shown that a recursively optimal policy may not be a hierarchically optimal policy, given that the boundary states have different optimal values in a sub-MDP. Thus, for each sub-MDP, if the optimal values of boundary states do not differ much, which is shown in (43), the hierarchical optimality can be ensured or well approximated.

*3) Performance Errors of the Combined Policy in HAPI:* In the framework of HAPI, the performance value of a hierarchical policy is the sum of performance values of local policies. Therefore, if locally optimal policies are obtained for each sub-MDP, the hierarchically optimal policy is a combination of local optimal policies achieving the highest cumulative rewards for any state trajectories. In our previous analysis, we have verified that when $\pi_i^*(1 \leq i \leq n)$ is the optimal policy in sub-MDP $M_{2i-1}$, and $\pi_{n+1}^*$ is the optimal policy of $M_{2n}$, the combined policy in HAPI is a hierarchically optimal policy or the performance error between the combined policy and the hierarchically optimal policy is small.

For MDPs with large or continuous state spaces, only near-optimal policies with performance errors can be obtained. Based on the results in (25), (26), and (36), it is shown that the performance errors of local policies in HAPI can be reduced. Therefore, if near-optimal policies with reduced performance errors are obtained in sub-MDPs, the performance error of the combined policy in HAPI can be expected to be small and its performance will be better than a single API algorithm.

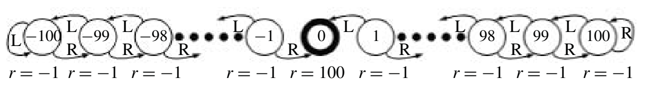


Fig. 4. Chain MDP.

*4) Further Discussions:* After satisfying certain assumptions, the proposed HAPI approach can also be extended to stochastic MDPs with terminal states. In addition to the reward function defined in (10), one basic assumption is that the optimal value function of a state satisfies $V^*(x) \propto 1/T(x), \forall x \in X$, where $T(x)$ is the expected time steps from $x$ to terminal states. Since the state space decomposition procedure presented in this paper is only based on optimal value functions, it can also be applied to stochastic MDPs. Fig. 1 indicates that $X_1$ includes all the states the expected cumulative rewards of which are greater than or equal to the threshold and other states included in $X_2$. Based on the definition of sub-spaces $X_1$ and $X_2$, two stochastic sub-MDPs $M_1$ and $M_2$ can also be defined in which the state transition probabilities within a sub-space equal the original MDP. If there is a state transition from a state $x$ in $X_2$ to $X_1$, the successive state will be regarded as a terminal state in $X_2$. In the following section, simulation results on a stochastic chain MDP illustrate the effectiveness of HAPI in stochastic MDPs with terminal states.

It is critical for the success of API algorithms to reduce the errors in policy evaluation. As illustrated in (25) and (26), the performance of the approximated policies can be bounded in quadratic norms by a function of the approximation errors of value functions and the Bellman residuals during the policy evaluation steps. In [38], a tighter upper bound on the performance loss of API algorithms is derived and indicates that the reduction of policy evaluation errors leads to lower upper bounds of performance losses. Therefore, the proposed HAPI algorithm reduces the performance error bounds of local policies by improving the approximation precision of value functions in policy evaluation steps. Moreover, because of the hierarchical optimality of the combined policy, the performance of the combined policy in HAPI also improves.

In addition to many applications in supervised learning, researchers publishing in RL literature have explored the function approximation using a tree-based structure. In [39], the U tree algorithm was proposed to generate a tree-based state discretization that efficiently finds the relevant state chunks of large propositional domains. The continuous U tree algorithm presented in [40] transfers regression tree techniques to RL. In [41], the fitted Q iteration algorithm uses several tree-based regression methods and obtains good results when combining the fitted Q iteration algorithm with tree-based ensemble methods such as extra-trees, tree bagging, and totally randomized trees. In [42], the U-tree algorithm was combined with option-based HRL algorithms to automate state abstraction in options, which can realize option-specific state abstraction to achieve better performance. The tree structure in HAPI focuses on state space decomposition in improving the performance of VFA, while previous tree-based RL methods use tree structures for

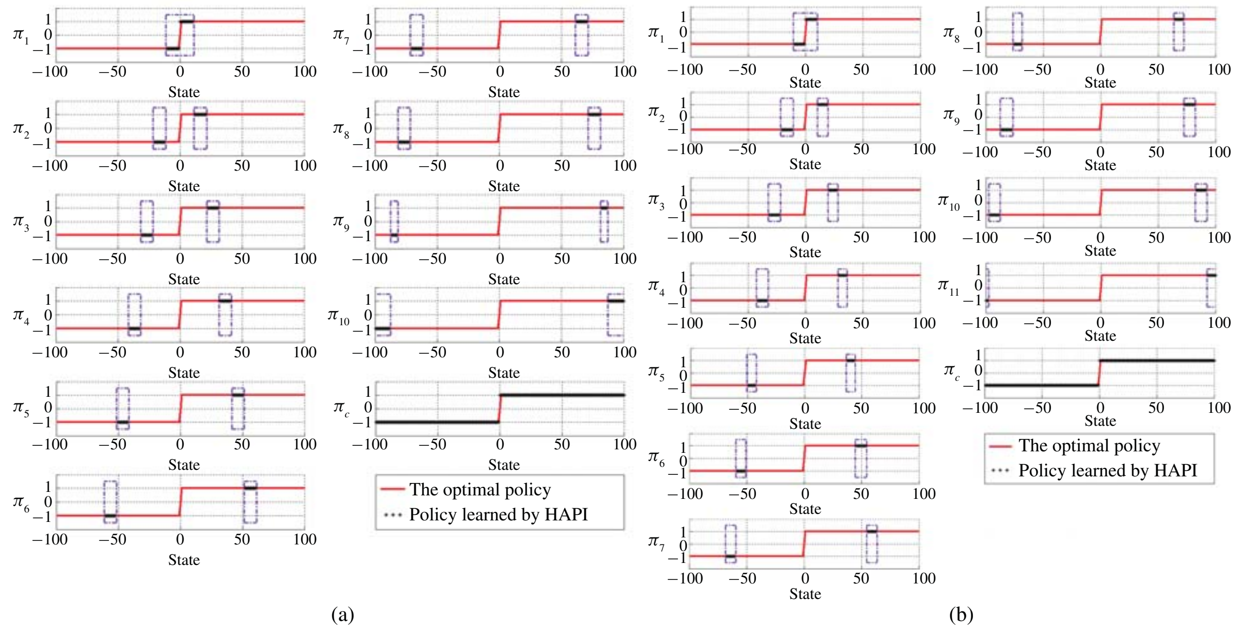


Fig. 5. Local ($\pi_i$) and combined ($\pi_c$) near-optimal policies learned by HKLSPI. ($N_{\min} = 20$) 1: "left," −1: "right." (a) Learning results for the deterministic MDP. (b) Learning results for the stochastic MDP.

feature representation or state abstraction in learning a value function. These two kinds of tree structures are complementary to each other. It can be expected that when other tree-based RL methods are integrated with HAPI, the performance of HAPI will be better than tree-based RL without state space decomposition.

In summary, by using the binary-tree space decomposition and the policy combination strategy, HAPI provides a new HRL framework for API, and it has advantages over LSPI and KLSPI in policy quality and stability, as illustrated in the following experiments. Furthermore, HAPI is a novel ensemble RL method differing from previous ensemble RL algorithms [43], [44] in that the samples are decomposed during the learning process and the policy combination is based on the principle of hierarchical optimality.

## IV. Experimental Results

### A. Chain MDP

The chain MDP problem considers a Markov chain with 201 states (indexed from −100 to 100), as illustrated in Fig. 4. The state numbered 0 is the absorbing state. At each non-absorbing state, there are two actions available, i.e., "go left" and "go right." According to the selected action, the next state becomes the left neighbor or the right neighbor of the current state. If the next state is a non-absorbing state, the reward of the state transition is −1, otherwise, the reward will be 100. The optimal action is "go right" when the state index is negative and "go left" when the state index is positive. The discount factor $\gamma$ is set at 0.9. For the choice of $\gamma$, the performance analysis in Section III.-B indicates that the value of $\gamma$ should be near 1 but not too near because the performance bounds in (44) are proportional to $1/(1-\gamma)$. In our experiments, a value of $\gamma$ between 0.9 and 0.95 led to good performance, but when $\gamma$ is equal to 0.99, the performance of HAPI and LSPI became worse.

Because the state space of the chain MDP is small and the problem is deterministic, the data samples were collected in a uniform way, in which each state-action pair was tested as a sample. So, the collected sample set consists of 400 samples. In the LSPI algorithm, the basis functions were selected as

$$\begin{cases} \phi(x,L) = \left(1, \frac{10x}{201}, \left(\frac{10x}{201}\right)^2, \left(\frac{10x}{201}\right)^3, \left(\frac{10x}{201}\right)^4, 0,0,0,0,0\right)^T \\ \phi(x,R) = \left(0,0,0,0,0,1, \frac{10x}{201}, \left(\frac{10x}{201}\right)^2, \left(\frac{10x}{201}\right)^3, \left(\frac{10x}{201}\right)^4\right)^T \end{cases} \tag{46}$$

where $x$ is the state index and "L" and "R" denote the actions of "go left" and "go right," respectively.

In the KLSPI algorithm, the kernel function and the data dictionary for the sparsified kernel matrix were selected as

$$\begin{cases} k\left(x_i, x_j\right) = e^{-\left(\frac{(x_i - x_j)}{10}\right)^2} \\ Dic = \{10 * i \,|\, i = -9, -8, \ldots, 8, 9\}. \end{cases} \tag{47}$$

To test the effectiveness of HAPI in stochastic MDPs, a stochastic model of the above chain MDP was simulated. In the stochastic model, at each non-absorbing state, there are still two actions available, i.e., "go left" and "go right." But for the "go left" action, there will be a probability of 0.9 to go to the left state (or go to itself for state −100), and the probability to the right state is 0.1 (or go to itself for state 100). Similar state transition probability is also defined for the "go right" action. The optimal policy equals the deterministic version. By automatically decomposing the original state space into 11 sub-spaces and combining the local policies, the proposed HAPI algorithm successfully obtained the optimal policy of the stochastic chain MDP problem. However, more data samples are needed because of the stochastic property of

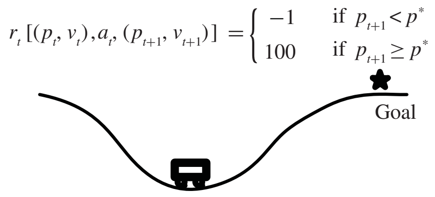

$r_t[(p_t, v_t), a_t, (p_{t+1}, v_{t+1})] = \begin{cases} -1 & \text{if } p_{t+1} < p^* \\ 100 & \text{if } p_{t+1} \geq p^* \end{cases}$
Goal

Fig. 6. Mountain-car task.

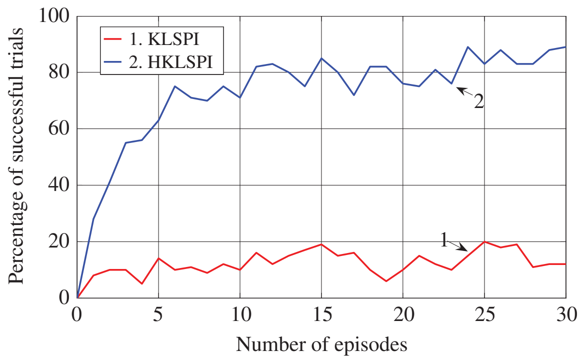


Fig. 7. Performance comparisons using percentage of successes.

the problem. In Fig. 5(b), the learning results of HAPI in the stochastic chain MDP are illustrated, where 10 000 samples were collected uniformly from the state space. In the stochastic case, LSPI and KLSPI also failed to find the optimal policy.

## B. Mountain-Car Problem

In the mountain-car task, there are two continuous states: $s_t = [p_t, v_t]$, where $p_t$ and $v_t$ are the position and velocity of the car. There are two possible actions ($a_t \in \{-1, +1\}$): full throttle forward (+1) and full throttle reverse (−1). The mountain road is defined as $h = \sin(3p_t)$, where $h$ is the height. The car moves according to the following dynamics:

$$\begin{cases} v_{t+1} = bound[v_t + 0.001a_t + g\cos(3p_t)] \\ p_{t+1} = bound[p_t + v_t] \end{cases} \tag{48}$$

where the *bound* operation enforces $p_{t+1} \in [-1.5, 0.5]$ and $v_{t+1} \in [-0.07, 0.07]$, and the acceleration of gravity $g = -0.0025$.

When $p_{t+1}$ reaches the left bound, $v_{t+1}$ is reset to zero. When it reaches the right bound, it reaches the goal (the star in Fig. 6), and the episode terminates. Each episode starts from the lowest point $p = -0.5$ with a velocity of 0, and ends when the car reaches the goal point ($p^* = 0.45$) or the number of steps is no less than 1000. The reward is designated as

$$r_t\left[(p_t, v_t), a_t, (p_{t+1}, v_{t+1})\right] = \begin{cases} -1 & \text{if } p_{t+1} < p^* \\ 100 & \text{if } p_{t+1} \geq p^*. \end{cases} \tag{49}$$

The experiments evaluated both API and HAPI algorithms. Because LSPI dealt with the problem of linear feature selection and its performance was usually inferior to KLSPI, only kernel-based features were used in the evaluations, i.e., performance comparisons were made between KLSPI and HKLSPI. The same sample sets with different episodes were used for

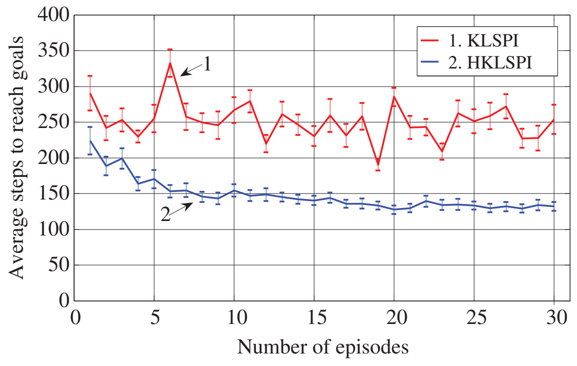


Fig. 8. Performance comparisons using averaged steps to goal.

both algorithms, in which the samples were collected using the adaptive sample collection method discussed in Section III. The discount factor was 0.90. The kernel function was

$$k\left(s_i, s_j\right) = e^{-\left[\left(\frac{p_i - p_j}{\sigma_1}\right)^2 + \left(\frac{v_i - v_j}{\sigma_2}\right)^2\right]} \tag{50}$$

where $s_i$ and $s_j$ denote $(p_i, v_i)$ and $(p_j, v_j)$, respectively, $\sigma_1 = 0.5, \sigma_2 = 0.05$.

For HKLSPI, the threshold value $\mu_i$ for space decomposition is automatically determined during the learning process, and the minimal number of remained samples is 100. The episode number of training samples varies from 1 to 30. For each episode number, a trial is defined as running API algorithms from a randomly initialized policy to convergence. In the experiments, 100 runs were executed to evaluate the performance of KLSPI and HKLSPI. A successful run means the final near-optimal policy can make the car reach the goal within 1000 steps. During the 100 runs, the number of successful trials for HKLSPI and KLSPI was compared, as depicted in Fig. 7. Furthermore, the performance curves of the final policies, measured as steps to reach the goal, averaged in 100 runs, together with the 95% confidence intervals, are illustrated in Fig. 8.

As depicted in Fig. 7, with the same set of training samples, the HKLSPI approach has much higher percentages of successful trials when compared with KLSPI, meaning that using the same experience data, HKLSPI can obtain a near-optimal policy with a much higher probability. Fig. 8 illustrates that HKLSPI has much better performance than KLSPI in terms of the quality of the final policies, i.e., the averaged steps to the goal. Therefore, the proposed HAPI approach can greatly improve the data efficiency and the quality of final policies for conventional API algorithms.

To compare the performance between HAPI and API in different problem settings, the mountain car problem was simulated with different dynamics models and the performance of RL algorithms was evaluated both in terms of the SR (computed in 100 independent runs) and the quality of final near-optimal policies (averaged steps to reach the goal state).

Tables I and II show the performance comparisons between KLSPI and HKLSPI using five different problem settings, including the standard parameter setting, the acceleration of gravity with additive uniform noises and Gaussian noises, and the actions or power with additive uniform noises and

TABLE I
PERFORMANCE COMPARISONS IN TERMS OF SUCCESS RATES (SR)*

| Models \ SR | $N = 10$ | | $N = 20$ | | $N = 30$ | |
|---|---|---|---|---|---|---|
| | KLSPI | HKLSPI | KLSPI | HKLSPI | KLSPI | HKLSPI |
| Model 1 | 0.10 | **0.71** | 0.10 | **0.76** | 0.12 | **0.89** |
| Model 2 | 0.11 | **0.85** | 0.12 | **0.84** | 0.09 | **0.89** |
| Model 3 | 0.13 | **0.71** | 0.20 | **0.82** | 0.16 | **0.86** |
| Model 4 | 0.10 | **0.83** | 0.12 | **0.84** | 0.13 | **0.86** |
| Model 5 | 0.11 | **0.70** | 0.15 | **0.86** | 0.16 | **0.85** |

TABLE II
PERFORMANCE COMPARISONS IN TERMS OF AVERAGE STEPS TO REACH GOALS (AS)*

| Models \ AS | $N = 10$ | | $N = 20$ | | $N = 30$ | |
|---|---|---|---|---|---|---|
| | KLSPI | HKLSPI | KLSPI | HKLSPI | KLSPI | HKLSPI |
| Model 1 | 266.50 | **154. 39** | 285.10 | **127.62** | 253.58 | **131.89** |
| Model 2 | 217.00 | **150. 21** | 222.25 | **142.75** | 235.00 | **129.37** |
| Model 3 | 237.90 | **146. 03** | 236.00 | **143.59** | 288.06 | **132.37** |
| Model 4 | 260.90 | **147. 66** | 286.33 | **147.8** | 291.15 | **129.27** |
| Model 5 | 292.50 | **152. 73** | 270.60 | **129.8** | 285.38 | **125.47** |

*Five different dynamics models were simulated to test the performance of HAPI and API:
Model 1: The standard mountain car problem.
Model 2: The acceleration of gravity with additive uniform noises in $[-0.0005,\ 0.0005]$.
Model 3: The acceleration of gravity with additive Gaussian noises $\mu = 0, \sigma = 0.00025$.
Model 4: Actions with additive uniform noises from $[-0.2,\ 0.2]$.
Model 5: Actions with additive Gaussian noises $\mu = 0, \sigma = 0.1$.

Gaussian noises. From Table I, the HKLSPI has much higher SR than KLSPI under different problem settings, where $N$ is the number of sampling episodes. In Table II, using the same number of sampling episodes, the quality of the final combined policy in HKLSPI is much better than KLSPI. In our experiments, when different kernel parameters were used, HKLSPI achieved better performance than KLSPI, in terms of the SR in 100 runs and the averaged steps to reach the goal.

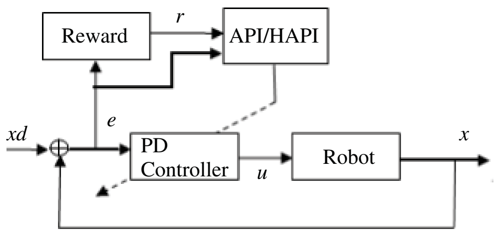


Fig. 9. API-based robot motion control system.

### C. Learning Control of a Real Mobile Robot

Robust motion control algorithms are fundamental to autonomous operation of mobile robots. Because of the nonlinear dynamics of robot systems and the uncertain disturbances from the environment, robot motion control presents difficult problems in engineering. Popular approaches that develop motion control algorithms range from model-based control to machine learning techniques, and all require a high level of expertise and effort for implementation. In the following, a path-tracking control strategy based on API combined with proportional-derivative (PD) control was studied using a real wheeled mobile robot. The advantage of RL for mobile robots is that optimized control policies can be obtained without much *a priori* knowledge on dynamics models of mobile robots and unknown external disturbances.

The mobile robot platform is a P3-AT wheeled mobile robot system. As a differentially driven platform, the robot receives control commands in terms of linear velocity and angular velocity, which are transformed by a built-in controller into a differential control strategy for the wheels. The position and orientation of the robot are measured by high-resolution optical encoders mounted on the driven wheels. Fig. 9 illustrates the API-based motion control system of the mobile robot, where API or HAPI algorithms are used to optimize the PD control parameters based on sampled trajectories.

The task for the mobile robot is to track a path with minimal tracking errors. In the PD controller, the following control law is used:

$$\omega_d(t) = k_p(t)e(t) + k_d(t)\dot{e}(t) \qquad (51)$$

where $e(t) = y_d(t) - y(t)$, $y(t)$, and $y_d(t)$ are the actual and desired positions in the vertical axis, respectively, and $k_p(t)$ and $k_d(t)$ are time-varying PD coefficients determined by API algorithms.

In the experiments, a global coordinate $(x,\ y,\ \theta)$ for the robot was defined, and the initial position of the robot was set in a small region around $(x,\ y) = (0, 0)$, with a randomly selected initial orientation. The desired path was $y = 5m$, and the absorbing states were defined as

$$\vec{x}_T \in \{(y,\theta)\,||y-5| < 0.2,\ |\theta| < 0.1\}. \tag{52}$$

When the states of the robot reach the absorbing states, the path-tracking error becomes very small. Because only lateral control was considered, the longitudinal velocity of the robot was set as a constant $v = 0.3$ m/s in simulation and real-time control. The action space of the MDP was defined as a series of PD coefficients expressed by $a(t) \in [(k_{p1}, k_{d1}), (k_{p2}, k_{d2}),\ldots,(k_{pn}, k_{dn})]$. Because of the difficulties in measuring the robot velocities, we only used the position and orientation information to define the states. In the above task, the state vector of the MDP is defined as $\vec{x} = (y, \theta)$.

The learning control objective is to realize a time-optimal control for the robot from a random initial configuration to the absorbing states defined in (52). To optimize the path-tracking performance, the reward function is defined as follows:

$$r_t = \begin{cases} 100 & |y_t - 5| < 0.2, |\theta_t| < 0.1 \\ -1 & \text{others.} \end{cases} \tag{53}$$

To facilitate sample collection, the commercial simulation software of the P3-AT mobile robot system generated sample trajectories. In the simulation software, random noises were added in the execution of the robot control commands to emulate disturbances in real world. A sample collection episode starts from a configuration $(x, y, \theta)$ uniformly distributed in the following interval:

$$[0.0,\ 1.0] \times [-1.0,\ 6.0] \times \left[\frac{-\pi}{18},\ \frac{10\pi}{18}\right].$$

Each episode has a maximum number of state transition steps of 10. The action policy is also a random policy to select the PD parameters $[k_p(t),\ k_d(t)]$ from three candidate combinations: $a \in \{[1, 60], [3, 50], [5, 40]\}$, manually selected with stability properties. API algorithms are used to approximate optimal control policies by learning a switching policy among the candidate PD parameters. The time step for online control is 0.5 s. Using the simulation software, 1000 episodes of data samples were collected for the implementation of KLSPI and HKLSPI. The final near-optimal policies of KLSPI and HKLSPI were evaluated in the real robot platform and the simulation software, obtaining similar results. In the experiments, to approximate the action value functions, a radial basis function (RBF) kernel was used. The same RBF width $\sigma$ was employed to compare the performance between KLSPI and HKLSPI. After the convergence of API and HAPI algorithms, the performance of the final policies were tested both on the real and simulated robot. The initial configuration of the robot was set as $(x, y, \theta) = (0, 0, 0)$.

Fig. 10 illustrates the path-tracking trajectories of KLSPI and HKLSPI in the real and simulated robot. For comparison, the best performance obtained by conventional PD control using one of the three candidate PD parameters is also shown. The initialization parameters were set as $\gamma = 0.9, N_{\min} = 50, \delta = 0.01$, and $\sigma = 4$. The points marked with large hexagrams on the trajectories represent the first time of the robot state to reach an absorbing state. The near-optimal policy of HKLSPI causes the robot to reach the absorbing state in a much shorter time, which is better than both KLSPI and conventional PD control.

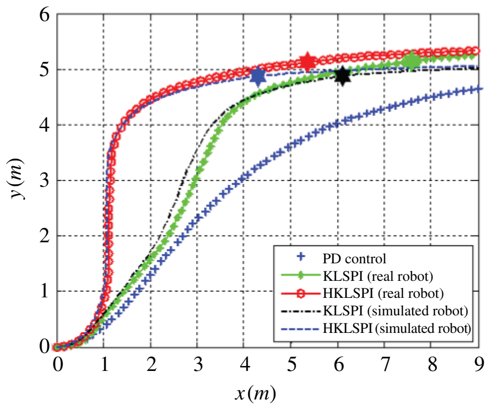


Fig. 10. Performance comparison between HAPI and API both in the simulated and real mobile robots.

TABLE III
PERFORMANCE COMPARISONS BETWEEN HAPI AND API

| | Data set 1 | | Data set 2 | | Data set 3 | |
|---|---|---|---|---|---|---|
| | API | HAPI | API | HAPI | API | HAPI |
| $N_{goal}$ | 79 | **55** | 58 | **50** | 52 | **48** |
| $E_{total}$ | 183.7 | **106.3** | 116.3 | **101.7** | 116.1 | **100.9** |

In addition to the results in Fig. 10, we tested the performance of API and HAPI using difference sample sets and different initial parameters and configurations and illustrated that HKLSPI can obtain better performance than KLSPI using the same sample sets and initialization parameters. Table III summarizes the performance comparisons between HAPI and API using three independent sample sets, where $N_{goal}$ is the state transition steps to goal and $E_{total}$ denotes the summed absolute tracking errors along the whole trajectory.

## V. CONCLUSION

In the proposed HAPI method, an absorbing MDP automatically divides into several sub-MDPs using the binary-tree space decomposition procedure, and the optimal policy of the original MDP can be approximated by the local near-optimal policies. Based on the binary-tree state space decomposition, the hierarchical optimality of the combined policy can be guaranteed or approximated well. In HAPI, because API algorithms are implemented in smaller sub-MDPs with higher precision of policy evaluation, the performance errors of local policies can be reduced. Therefore, because of its hierarchical optimality or approximate hierarchical optimality, the performance of the combined policy can also be improved significantly.

For high-dimensional complex decision problems, exploring the state space more efficiently was critical to the success of HAPI, and some recent progress has been made, namely, the fitted Rmax algorithm [45], the LSPI-Rmax algorithm [46], and others. Furthermore, other learning-based state space decomposition approaches for API in general MDPs require further investigation. In this paper, although the proposed

HAPI was successfully applied to an absorbing MDP with stochastic state transitions, more theoretical analysis and performance evaluations on stochastic MDPs will be our future work.

## Acknowledgment

The authors would like to thank the Associate Editor and anonymous reviewers for their valuable comments and suggestions, which greatly improved the quality of our paper.

## References

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

[2] F. Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE Comput. Intell. Mag.*, vol. 4, no. 2, pp. 39–47, May 2009.

[3] G. Tesauro, "TD-gammon a self-teaching backgammon program achieves master-level play," *Neural Comput.*, vol. 6, no. 2, pp. 215–219, Mar. 1994.

[4] W. Zhang and T. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *Proc. 14th Int. Joint Conf. Art. Intell.*, San Francisco, C.A., 1995, pp. 1114–1120.

[5] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Mach. Learn.*, vol. 33, nos. 2–3, pp. 235–262, 1998.

[6] A. Y. Ng, H. J. Kim, M. Jordan, and S. Sastry, "Autonomous helicopter flight via reinforcement learning," in *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press, 2004.

[7] R. E. Bellman, *Dynamic Programming*. Princeton, N.J: Princeton Univ. Press, 1957.

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

[9] X. Xu and H. G. He, "Residual-gradient-based neural reinforcement learning for the optimal control of an acrobat," in *Proc. IEEE Int. Symp. Intell. Control*, Vancouver, Canada, 2002, pp. 758–763.

[10] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Art. Intell. Res.*, vol. 15, no. 1, pp. 319–350, Jul. 2001.

[11] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithm," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2000.

[12] X. Xu, H. G. He, and D. W. Hu, "Efficient reinforcement learning using recursive least-squares methods," *J. Art. Intell. Res.*, vol. 16, no. 1, pp. 259–292, Jan. 2002.

[13] J. Fu, H. He, and X. Zhou, "Adaptive learning and control for MIMO system based on adaptive dynamic programming," *IEEE Trans. Neural Netw.*, vol. 22, no. 7, pp. 1133–1148, Jul. 2011.

[14] F. Y. Wang, N. Jin, D. R. Liu, and Q. L. Wei, "Adaptive dynamic programming for finite-horizon optimal control of discrete-time nonlinear systems with $\varepsilon$-error bound," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 24–36, Jan. 2011.

[15] H. G. Zhang, Y. H. Luo, and D. R. Liu, "Neural-network-based near-optimal control for a class of discrete-time affine nonlinear systems with control constraints," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1490–1503, Sep. 2009.

[16] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.

[17] X. Xu, D. W. Hu, and X. C. Lu, "Kernel based least-squares policy iteration for reinforcement learning," *IEEE Trans. Neural Netw.*, vol. 18, no. 4, pp. 973–992, Jul. 2007.

[18] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Syst.-Theory Applicat.*, vol. 13, nos. 1–2, pp. 41–77, Jan.–Apr. 2003.

[19] M. Ghavamzadeh and S. Mahadevan, "Hierarchical average reward reinforcement learning," *J. Mach. Learn. Res.*, vol. 8, pp. 2629–2669, Nov. 2007.

[20] 20 T. G. Dietterich, "Hierarchical reinforcement learning with the MaxQ value function decomposition," *J. Art. Intell. Res.*, vol. 13, no. 1, pp. 227–303, Aug. 2000.

[21] R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Art. Intell.*, vol. 112, nos. 1–2, pp. 181–211, Aug. 1999.

[22] T. G. Dietterich, "State abstraction in MAXQ hierarchical reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 994–1000.

[23] D. Andre and S. J. Russell, "State abstraction for programmable reinforcement learning agents," in *Proc. 18th Nat. Conf. Art. Intell.*, 2002, pp. 119–125.

[24] B. Hengst, "Safe state abstraction and reusable continuing subtasks in hierarchical reinforcement learning," in *Proc. AI: Adv. Art. Intell. Lecture Notes Comput. Sci.*, 2007, pp. 58–67.

[25] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 1998.

[26] Q. Y. Hu and W. Y. Yue, *Markov Decision Processes with Their Applications*. New York: Springer-Verlag, 2008.

[27] J. Boyan, "Technical update: Least-squares temporal difference learning," *Mach. Learn.*, vol. 49, nos. 2–3, pp. 233–246, 2002.

[28] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[29] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2275–2285, Aug. 2004.

[30] R. Munos, "Error bounds for approximate policy iteration," in *Proc. 20th Annu. Int. Conf. Mach. Learn.*, 2003, p. 560.

[31] C. Dimitrakakis and M. G. Lagoudakis, "Algorithms and bounds for rollout sampling approximate policy iteration," in *Proc. 8th Eur. Workshop, Recent Adv. Reinforce. Learn.*, Villeneuve d'Ascq, France, LNAI 5323. Jun.–Jul. 2008, pp. 27-40.

[32] R. Munos and C. Szepesvári, "Finite–time bounds for fitted value iteration," *J. Mach. Learn. Res.*, vol. 9, pp. 815–857, May 2008.

[33] B. H. Li and J. Si, "Approximate robust policy iteration using multilayer perceptron neural networks for discounted infinite-horizon Markov decision processes with uncertain correlated transition matrices," *IEEE Trans. Neural Netw.*, vol. 21, no. 8, pp. 1270–1280, Aug. 2010.

[34] J. Seiffertt and D. C. Wunsch, "Backpropagation and ordered derivatives in the time scales calculus," *IEEE Trans. Neural Netw.*, vol. 21, no. 8, pp. 1262–1269, Aug. 2010.

[35] A. D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific, 1996.

[36] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Automat. Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[37] A. Antos, C. Szepesvari, and R. Munos, "Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path," *Mach. Learn.*, vol. 71, no. 1, pp. 89–129, 2008.

[38] A. M. Farahmand, R. Munos, and C. Szepesvári. "Error propagation for approximate policy and value iteration," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2010.

[39] A. McCallum, "Reinforcement Learning with Selective Perception and Hidden State," Ph.D. thesis, Comput. Sci. Dept., Univ. Rochester., Rochester, NY, 1995.

[40] T. B. U. William and M. M. Veloso. "Tree based discretization for continuous state space reinforcement learning," in *Proc. AAAI-98 15th Nat./10th Conf. Art. Intell./Innovative Applicat. Art. Intell.*, Madison, WI, Jul. 1998, pp. 769–774.

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

[42] A. Jonsson and A. G. Barto, "Automated state abstraction for options using the U-tree algorithm," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2000.

[43] M. A. Wiering and H. V. Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Trans. Syst. Man Cybern. Part B: Cybern.*, vol. 38, no. 4, pp. 930–936, Aug. 2008.

[44] H. Alexander and U. Steffen, "Ensembles of neural networks for robust reinforcement learning," in *Proc. 19th Int. Conf. Mach. Learn. Applicat.*, Washington D.C., 2010, pp. 401–406.

[45] K. J. Nicholas and S. Peter, *Model-Based Exploration in Continuous State Spaces*, I. Miguel and W. Ruml, Eds. New York: Springer-Verlag, 2007, pp. 258–272.

[46] L. H. Li, M. L. Littman, and C. R. Mansley, "Online exploration in least-squares policy iteration," in *Proc. AAMAS-09 8th Int. Conf. Autonomous Agents Multiagent Syst.*, Budapest, Hungary, May 2009, pp. 733–739.

**Xin Xu** (M'07) received the B.S. degree in electrical engineering from the Department of Automatic Control, National University of Defense Technology (NUDT), Changsha, China, in 1996, and the Ph.D. degree in control science and engineering from the College of Mechatronics and Automation (CMA), NUDT.

He has been a Visiting Scientist for cooperation research with the Hong Kong Polytechnic University, Kowloon, Hong Kong, University of Alberta, Edmonton, AB, Canada, University of Guelph, Guelph, ON, Canada, and the University of Strathclyde, Glasgow, U.K., respectively. He is currently a Full Professor with the Institute of Automation, CMA, NUDT. He has co-authored four books and published more than 70 papers in international journals and conferences. His current research interests include reinforcement learning, learning controls, robotics, data mining, autonomic computing, and computer security.

Dr. Xu is one of the recipients that received the First Class Natural Science Award from Hunan Province, China, in 2009, and the Fork Ying Tong Youth Teacher Fund of China in 2008. He is a Committee Member of the IEEE Technical Committee on Approximate Dynamic Programming and Reinforcement Learning and the IEEE Technical Committee on Robot Learning. He has served as a PC Member or Session Chair in many international conferences.

**Chunming Liu** received the B.Sc. and M.Sc. degrees from the National University of Defense Technology, Changsha, China, in 2004 and 2006, respectively. He is currently pursuing the Ph.D. degree with the same university

His current research interests include intelligent systems, machine learning, and autonomous land vehicles.

**Simon X. Yang** (S'97–M'99–SM'08) received the B.Sc. degree in engineering physics from Beijing University, Beijing, China, in 1987, the M.Sc. degree in biophysics from the Chinese Academy of Sciences, Beijing, in 1990, another M.Sc. degree in electrical engineering from the University of Houston, Houston, TX, in 1996, and the Ph.D. degree in electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, in 1999.

He is currently a Professor and Head of the Advanced Robotics and Intelligent Systems Laboratory, University of Guelph, Guelph, ON, Canada. His current research interests include intelligent systems, robotics, sensors and multi-sensor fusion, wireless sensor networks, control systems, and computational neuroscience.

Prof. Yang has served as an Associate Editor of the IEEE TRANSACTIONS OF NEURAL NETWORKS, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, and several other journals. He is involved in the organization of many conferences. He is the General Chair of the IEEE International Conference on Automation and Logistics in 2011.

**Dewen Hu** (M'03–SM'06) was born in Hunan, China, in 1963. He received the B.Sc. and M.Sc. degrees from Xi'an Jiaotong University, Xi'an, China, in 1983 and 1986, respectively. He received the Ph.D. degree from the National University of Defense Technology, Changsha, China, in 1999.

He was with the National University of Defense Technology in 1986. He was a Visiting Scholar with the University of Sheffield, Sheffield, U.K., from October 1995 to October 1996. He was promoted to Professor in 1996. His current research interests include image processing, system identification and controls, neural networks, and cognitive science.

Dr. Hu is an Editorial Board member of *Neural Networks*.