A hybrid Architecture for Function Approximation

HASSAB ELGAWI Osman
Computational Intelligence and Systems Science, Tokyo Institute of Technology-Japan
Email: osman@isl.titech.ac.jp

***Abstract*—This paper proposes a new approach to build a value function estimation based on a combination of temporal-different (TD) and on-line variant of Random Forest (RF). We call this implementation Random-TD. First RF is induced into on-line mode in order to deal with large state space and memory constraints, while state-action mapping is based on the Bellman error, or on the TD error. We evaluate the potential of the proposed procedure in terms of a reduction in the Bellman error with extended empirical studies on high-dimensional control problems (Ailerons, Elevator, Kinematics, and Friedman), a standard reinforcement learning benchmark on which several linear function approximators have previously performed poorly. The results demonstrate that a hybrid function approximation (Random-TD) can significantly improve the performance of TD methods.**

***Index Terms*—reinforcement learning, function approximation, TD-learning, random forests.**

I. Introduction

LARGE applications of reinforcement learning (RL) require the use of generalizing function approximators (FA) for finding an optimal policy for solving problems associated with very large state spaces, partially observable states, and non-stationary environments. Three distinct strategies have been proposed to learn an optimal policy using reinforcement information, 1) *value-function approach*[1], in which all function approximation efforts goes into estimating a value function and using that to compute a deterministic policy ("greedy" policy), 2) *a stochastic policy* approximate directly using an independent function approximator with its own parameters and 3) *policy gradient approach*, the policy parameters are updated approximately proportional to gradient: $\Delta\theta \approx \alpha\frac{\partial\rho}{\partial\theta}$, where θ denote the vector of policy parameters and ρ the performance of corresponding policy and α is a positive define step size. Good estimate of value function is highly depending on engineering feature space to find features represent accurately the value of a policy. Towards this goal, several studies have applied nearest neighbor, RBF and CMAC function approximations, but these cannot scale to problems with many features, especially if some features are irrelevant, even with careful engineering, continuous problems with more than 10 dimensional remain daunting [9]. Comparing to above mentioned approximators, decision-tree based ensemble learning algorithms have the desirable property of having less parameters to be tune and of their ability to deal with large state space; hence, such algorithms are easier to use. Ensemble decision tree methods such as *bagging* [3], *boosting* [6], *Random Forests* (RF) [4] and their invariants also generate extra information that allow to estimate the importance of each explanatory variable. e.g., based on permutation accuracy and on impurity decrease, RF measures variable importance by randomly permuting the values of the variable m for the out-of-bag[2] (OOB) cases for tree k, if variable m is important in the classification, then the accuracy of the prediction should decrease. On the other hand, we can consider the accumulated reduction at nodes according to the criteria used at the splits, an idea from the original CART [5] formulation. However, decision tree is always constructed in batch mode; the branching decision at each node of the tree is induced based on the entire set of training data. On-line[3] algorithms, sometimes prove to be more useful in the case of very large data sets, and situation where memory is limited. In most of the decision tree construction approaches and their applications [14], [15], the decision tree is modeled for classification tasks. Decision trees are employed for function approximation mostly in the context of reinforcement learning for value function estimation [16]–[18]. In this paper, we explore the potential of employing a hybrid function approximator to represent the value function. To this end a combination of RF and TD learning, both operate on on-line mode is proposed. We call this implementation Random-TD. The input space is divided into m different regions and at each sub-space a local estimation is performed in order to find a good feature matrix Ψ. To our knowledge, there have been no published results combining RF algorithm and RL. A possible reason for this lack of algorithms is that RF generally operates in batch mode, over a whole data set, while RL algorithms typically use on-line training. Our first aim was to extend the standard RF algorithm for on-line classification and to improve the TD learning methods. We partially succeeded in the first goal by using our recent proposed on-line RF based on correlation raking to generate new incremental feature space [7] and in the second by using a combination of TD-learning and on-line RF. We provide a motivation for this new approach in terms of a reduction in the Bellman error. Moreover, the novelty of our approach is its orientation towards the application of knowledge based function approximation, as a result of aggregate supervisor learning with TD learning. From this novel approach, we hope to highlight some of the strengths of RF approach to the RL

[1]Three categories: Dynamic Programming (DP), Monte Carlo (MC) methods, and Temporal Difference (TD) methods (For example, Q-learning, Sarsa, and Actor-Critic), according to the schemes for updating the value functions

[2]There is on average $I/e \approx 36.8$ of instances not taking part in construction of the tree, provides a good estimate of the generalization error (without having to do cross-validation).

[3]The distinction between on-line and off-line methods here refer to issues of time reversal in the computation. On-line methods process incoming data strictly in the order it is received, while off-line methods are therefore more desirable for real-time learning applications

problems. The reader is advised not to view this paper as supervised learning vs. RL discussion, so hybrid (aggregated) approaches are advisable. We demonstrate the effectiveness of Random-TD in approximating functions with extended empirical studies on a high-dimensional control problems (Ailerons, Elevator, Kinematics, and Friedman), all from Function Approximation Repository, publicly available at [11]. This paper is organized as follows. In Section II we highlight on reinforcement learning and function approximation. Section III presents random forest in reinforcement learning. Section IV describes empirical evaluations with experimental results for function approximation. In Section V we conclude and present venues for future work. Once the basis functions are constructed, the function approximator is trained as usual. alternatively, we devised on linear approximation of function value based on RF and the bellman error.

II. Function Approximation

Before proceeding with our presentation we need to introduce and familiarize the reader with our definitions and approaches by briefly review reinforcement learning (RL) and value function approximation. Finally, we present our novel implementation of on-line RF in the context of RL.

A. Reinforcement Learning

Reinforcement Learning (RL) is an approach of addressing a wide variety of planning and optimal control methods [1] to solve sequential decision making problems that can be modeled by *Markov Decision Processes* (MDPs) of a tuple $\langle S, A, P, R\rangle$ where S is the space of possible *states* of the environment, A is a set of *actions* available to the agent, $P : S\times A\times S \rightarrow [0, 1]$ defines a conditional probability distribution over *state transitions* given an action, and $R : S \times A \rightarrow R$ is a *reward function* (payoff) assigning immediate reward to action. The main idea of RL is that through trial and error, rather than being implicitly told, continuous interactions between agent and its dynamic environment can be made in order to satisfy the goal of autonomy and the adaptability.

B. Value Function Approximation

For large or continuous state space, value-function V^π can not be represented explicitly (Bellman's "curse of dimensionality" [13]), so instead there are two possible ways to learn the optimal value-function. One is to estimate the model (i.e., the transition probabilities and immediate costs) while the other is to estimate the optimal action-values directly. Various dynamic programming methods such as the value- or policy-iteration methods [12] are often used for approximation. In our modeling, we assume a large but limit state space represented by feature matrix Ψ, at any given time step $t = 0, 1, 2, \cdots$, an agent perceives its *state* s_t and selects an *action* a_t. By exploring a state space an agent tries to learn the best *policy*, $\pi : S \rightarrow A$, which maps states to actions. The system (environment) responds by given the agent some (possibly zero) numerical *reward* $r(s_t)$ and changing into state $s_t+1 = \delta(s_t, a_t)$. Estimate approximation for reward represented as a vector $R \in \Re^{|s|}$, are incremented by $r_t\phi_t$ and $\phi_t(\phi_t-\gamma\phi_{t+1})$, where ϕ_t, ϕ_{t+1} are feature vectors at time step t and $t + 1$, and the transition probabilities under π can be represented as a matrix $P \in \Re^{|s|\times|s|}$. The state transition may be determined solely by the current state and the agent's action or may also involve stochastic processes. For MDPs, we can define V^π formally as

$$V^\pi(s) = E^\pi\left\{\sum_{t=0}^{\infty}\gamma^t R_t|s_0 = s\right\} \quad (1)$$

Where $E^\pi\{\}$ denotes the expected value given that the agent follows policy, π, and $0 < \gamma < 1$ is the discount factor. Note that the value of the terminal state, if any, is always zero. A randamoized stationary policy can be identified with conditional probabilities $\pi(a|s)$, which specify the probability of choosing action a at state s. From mathematical point of views, value function V^π which can be represented as vector $V \in \Re^{|s|}$, is an approximate solution to Bellman's equation, which is then used to construct near optimal policies.

$$V^\pi = \sum_{a\in A(s)}\pi(a|s)\left(R(s,a) + \gamma\sum_{\acute{s}\in S}p(\acute{s}|s,a)V^\pi(\acute{s})\right), \quad (2)$$

A direct solution of the above equation may not be feasible, either due to unknown environment parameters (p and R) or due to the cardinality of the state space. In either case, temporal difference (TD) methods may be employed to estimate V^π. However, as can be seen in section III we are using TD aggregated with RF to estimate V^π. In this case, the value function approximation has the form $\hat{V} = \Phi\theta$, where Φ is a $|S| \times m$ matrix m which each row contains the feature vector for a particular state, and θ is a vector of m parameters. Typically it is desirable to have $m \ll |S|$. Usually only Φ is assumed to be given, and the learning algorithm adjusts θ. The *optimal policy*,π^*, can be defined in many ways, but is typically defined as the policy that produces the greatest cumulative reward over all states s.

$$\pi^* = \arg\max_\pi V^\pi(s), (\forall s) \quad (3)$$

where $V^\pi(s)$ is computed from Eq 1. Alternatively, $V^\pi(s)$ could be computed by summing the rewards over a finite horizon h:

$$\pi^* = \sum_{i=0}^{h} r_t + i \quad (4)$$

If the agent learns with the one-step learning, the agent can perform on-line learning, because it learn based on one action state value whereas other algorithms calculate current state value according to rewards of all actions.

III. Random Forests in Reinforcement Learning

In this section we explore the possibility of using our on-line RF algorithm for function approximation. Our implementation is structured by the desire to build practical agent applications and the desire to supply a behavioral guarantee based on a convergent learning algorithm. Based on the shortcomings of non-linear approximations of function value, which are usually

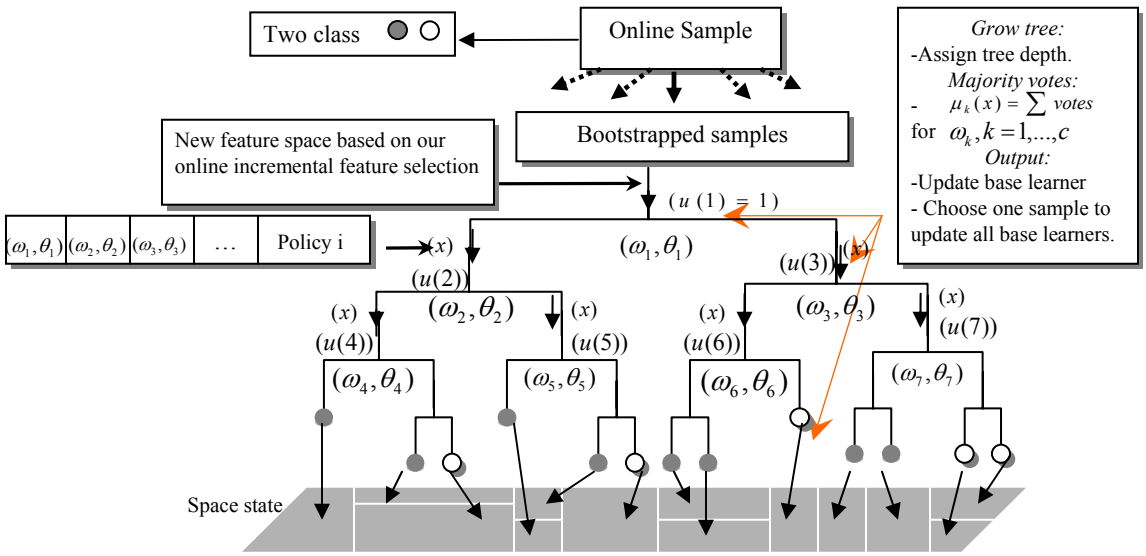

Fig. 1. A simple parameter representation of weights for a forest. The fitness of the policy is the reward "payoff" when the agent uses the corresponding random forest as its decision policy.

slow to converge and yield in computational complexities. We devised on linear approximation based on combination of RFs.

A. Online RF

The structure of on-line forest is shown in Figure 1. Based on variables ranking we develop a new, conditional permutation scheme for the computation of variable importance measure [7]. The resulting incremental variable importance is show to reflect the true impact of each predictor variable more reliably than the original marginal approach. According to features ranking results, different yet random feature subsets are used as new feature spaces for learning a diverse base-learners. The diversity stem from fully growing (unpruned) independent tree- base learners. Individual trees in RF are incrementally generated by specifically selected subsamples from the new feature spaces. In contrast to off-line random forests, where the root node always represents the class in on-line mode, for each training sample, the tree adapts the decision at each intermediate node (nonterminal) from the response of the leaf nodes, which characterized by a vector (w_i, θ_i) with $\|w_i\| = 1$. Root node numbered as 1, the activation of two child nodes $2i$ and $2i+1$ of node i is given as

$$u_{2i} = u_i.f(w_i^{'}x + \theta_i) \tag{5}$$

$$u_{2i+1} = u_i.f(-w_i^{'}x + \theta_i) \tag{6}$$

where x is the input pattern, u_i represents the activation of node i, and $f(.)$ is chosen as a sigmoidal function. Consider a sigmoidal activation function $f(.)$, the sum of the activation of all leaf nodes is always unity provided that the root node has unit activation. The forest consist of fully grown trees of a certain depth l. The general performance of the on-line forests depends on the depth of the tree. It is not clear how to select the depth of the on-line forests. One alternative is to create a growing on-line forests where we first start with an on-line forest of depth one. Once it converges to a local optimum, we increase the depth by adding one more level to the tree. Thus, we create our on-line forest by iteratively increasing its depth. However, we found that the nubmber of trees one needs for good performance eventually tails off as new data vectors are considered. Since after a certain depth, the performance of on-line forest does not vary to a great extent, the user may choose K (the number of trees in forest) to be some fixed value or may allow it to grow up to the maximum possible which is at most $|T|/N_k$, where N_k the user-chosen number of the size of each decision tree. The construction of on-line forest is based on the majority voting. If there are m decision trees, the majority voting method will give a correct decision if at least $floor(m/2)+1$ decision trees gives correct outputs. If each tree has probability p to make a correct decision, then the forest will have the following probability p to make a correction decision.

$$P = \sum_{\mathrm{i=floor(m/2)+1}}^{m} \binom{m}{i} p(1-p) \tag{7}$$

When classifying a new instance, we first estimate the average margin of the trees on the instances most similar to the new instance and then, after discarding the trees with negative margin, weight the tree's votes with the margin.

B. Futurization State Space

The state space is partitioned into m disjoint group, in order to transfer a state s from the input space to a vector of input features, then value function is estimated from feature space, $\hat{V} = \vec{\theta}^{\mathrm{T}} \cdot \vec{F}_s$, where $\vec{\theta}$ is the parameter vector and $\vec{F}_s$ is the feature vector. Represented state as feature vectors: for each

s:

$$\phi_s = [(\phi_s(1), \phi_s(2), \cdots, \phi_s(n))]^T \quad (8)$$

$$V_t(s, \theta_t) = \theta_t^T \phi_s = \sum_{i=1}^{n} \theta_t(i)\theta_s(i) \quad (9)$$

In Figure 1 you can see a decision tree which divides the state space into 13 regions of different resolution. The decision trees consist of two types of nodes: *decision nodes*, corresponding to state variables and *least nodes*, which correspond to all possible actions that can be taken. In a decision node a decision is taken about one of the input variables. Each least node stores the state values for the corresponding region in the state space, meaning that a leaf node stores a value for each possible action that can be taken. The tree starts out with only one leaf that represents the entire input space. So in a leaf node a decision has to be made whether the node should be split or not. Once a tree is constructed it can be used to map an input vector to a least node, which corresponds to a region in the state space. We will use Temporal-difference (TD) learning [2] to associate a value with each region. TD method is viewed as stochastic approximation methods for solving Bellman's equation. TD methods are commonly update $V^\pi(s)$ by an amount proportional to the TD error, δ where.

$$\delta = r + \gamma V^\pi(\acute{s}) - V^\pi(s), \quad (10)$$

and $\gamma \in [0, 1]$ is a discount factor.

C. Random-TD Architecture

One possible way to combine TD with RF is to choose the best combined strategy $s_i^{\rightarrow} = s_i^{\rightarrow}(s)$ given the expected combined strategy for each learners involved and to represent the value function V^π as combined policy of entire value function, rather than base value of a given state. iteration [10] to produce a feature representation for a final linear mapping where all the learning task place. Unlike in lookup tables, the value function estimate depends on a parameter vector θ_t, and only the parameter vector is updated. Figure 2 demonstrates the architecture of the proposed Random-TD. It consists of two modules, a learning module and a hybrid module. These two modules interact with each other. The learning module is in the first level, in which both on-line RF and TD learner learn their control policies individually, based on its current policy and state. Then, each learner submits its decision of the selected action or the preference of actions to the hybrid module, where Random-TD learn the combined policies. Value function is updated based on input state, reward, following Eq 10. The update rule from step 4 in algorithm 1 is used to compute the new desired value for state s, denoted $T(s) = (1-\alpha)V(s) + \alpha(r + \gamma V(\acute{s}))$. This will be used as target value for state s to generate an error rate. On the other hand the depths of trees are modified such as to minimize this error. Typically, this step is achieved by simply applying gradient descent once. The action selector chooses an action based on the value function and some exploration and exploitation strategies. The hybrid module is at the second level, where the input information from learning module is dynamically

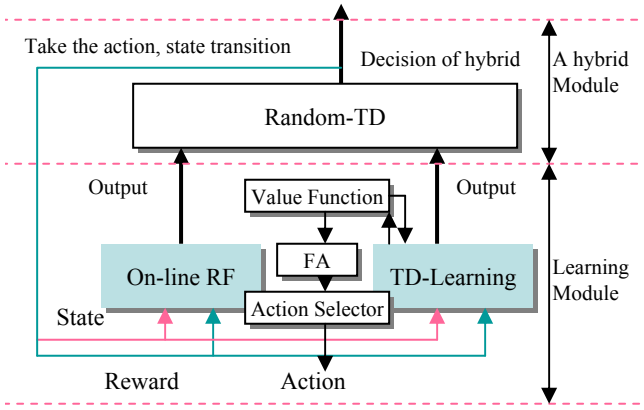

Fig. 2. Random-TD architecture.

aggregated. After that, the hybrid module sends a final decision of action back to the learning module. Then every learner in the learning module takes the action, transits to a new state, and obtains an instant reward. Then, each learner updates its policy. Repeat for a number of steps or until several criteria are satisfied. Because both the TD-algorithm and RF run on-line, this is frees us the curse of dimensionality in a sense that memory requirements need not be exponential in the number of dimensions. The overall effect is particularly efficient computationally. The Random-TD algorithm for value function approximation is shown in Algorithm 1. The function SELECT FEATURE (s_t, e_t, A, m) uses state s_t as data points and the Bellman error estimate e_t, while the transformation from state space to feature space is done on the previous step. Initially,

Algorithm 1 Random-TD Policy Evaluation

1: **Given** the dimensionality d to be used in each projection the number of features to be added in each iteration, m. the desire number of iteration K.
2: Initialize state s
3: Choose action a using policy π and observe reward r and next state $\acute{s}$
4: Update $V(s)$ such that $V(s) \leftarrow V(s) + \alpha[r + (s) - V(s)]$ where α is the learning rate and γ is the discount factor (both between 0 and 1)
5: $s \leftarrow \acute{s}$
6: Repeat steps 2-4 until episode ends
7: $\Phi^0 \leftarrow \vec{1}$
8: $\Phi^0 \leftarrow TD(s_t, r_t, \Phi^0)$
9: **for** $k = 1, 2, \cdots, K$ **do**
10: Estimate the Bellman residuals
11: $e_t \approx R(s_t + \gamma \sum_{s \in S} P_{st,s} \hat{V}_s^{k-1} - \hat{V}_{st}^{k-1})$
12: $A \leftarrow$ Random-TD (s_t, e_t, d)
13: $\Psi \leftarrow$ SELECT FEATURE (s_t, e_t, A, m)
14: $\Phi^k \leftarrow [\Phi^{k-1}, \Psi]$
15: $\Phi^k \leftarrow$ Random-TD(s_t, r_t, Φ^k)
16: $V^k \leftarrow \Phi^k \theta^k)$
17: **end for**
18: **return** $\hat{V}^k$

TABLE I
DESCRIPTION OF THE FUNCTION APPROXIMATION DATA SETS OBTAINED FROM THE BILKENT UNIVERSITY DATA REPOSITORY [11]

Dataset	Data size	D	t_{min}	t_{max}	N_{trn}	N_{tes}
Ailerons (AL)	7154	40	-0.0003	-0.00035	1000	6154
Elevator (EV)	8752	18	0.078	0.012	1000	7752
Kinematics (KI)	8192	8	1.458521	0.040165	800	7392
Friedman (FR)	40,768	10	30.522	-1.228	1300	39468

a single feature which is 1 everywhere is defined and TD is used to compute the parameters of the approximator. On every iteration the Bellman residuals e_t are estimated for each state. In general the model parameters P and R are not be available so the residuals can only be approximated based on an approximate model, or on sample data. The accuracy of the estimates is not crucial since on-line RF algorithm is robust to noise. In the results presented below, we use approximate Bellman errors, but using TD errors gives very similar results. In our experiments SELECT FEATURES is simply discritizes the state space onto $\Re^2$ into up to m states and returns the combined feature matrix. Random-TD is repeated to obtain a new approximation $\hat{V}^k$.

IV. EMPIRICAL EVALUATIONS

A. Datasets

In order to asses the potential of using Random-TD in approximating functions, we experimented with the algorithm described above, in the context of learning a high-dimensional control problems: Ailerons, Elevator, Kinematics, and Friedman, a standard reinforcement learning benchmark on which several linear function approximators have previously performed poorly. All these data sets are Function Approximation Repository,[4] publicly available at [11]. Table I summarizes the data sets in terms of the size of instance, number of input dimensions (D), the range of the target (predicted) (t_{min}, t_{max}), and the size of training and testing set (N_{trn}, N_{tes})

B. Experimental Settings

Before reporting our results, let us provide a brief description of the data sets as in [11]. THE AILERONS data set is a simulated control problem: The attributes describe the status of the aircraft, and the aim to predict the control action on the ailerons of an F16 aircraft . THE ELEVATOR data set is similar to the Ailerons, and obtained from the task of controlling an F16 aircraft. However, attributes here are different from the Ailerons domain, and the goal (predictor) variable is related to an action taken on the elevators of the aircraft. THE KINEMATICS data set is concerned with the forward kinematics of an eight-link robot arm. THE FRIEDMAN data set is an artificial data set originally used by Friedman (1991) in approximating functions with MARS (multivariate adaptive regression splines).
We added and report results for Friedman, in order to demonstrate the capability of Random-TD to scale up in higher density of state.

[4]Available at: *http://funapp.cs.bilkent.edu.tr/DataSets/*

TABLE II
PERFORMANCE OF RANDOM-TD ON THE FUNCTION APPROXIMATION DATA SETS IN TABLE I, FOR DIFFERENT DEPTHS

Dataset	Absolute error				
	Depth=3	Depth=4	Depth=5	Depth=6	Depth=7
Ailerons	0.000111	0.000114	0.000112	0.000117	0.000126
Elevator	0.0702	0.0541	0.0483	0.0298	0.0301
Kinematics	0.1001	0.0672	0.0487	0.0425	0.0521
Friedman	1.1213	0.7322	0.5622	0.6074	0.5628

TABLE III
RESULTS SHOWING A COMPARISON USING RANDOM-TD, TD, AND Q-LEARNING. THE UNITS FOR ABSOLUTE BELLMAN ERROR. IM DENOTE THE IMPROVEMENT OF RANDOM-TD OVER TD-LEARNING

Dataset	Random-TD	TD-learning	Q-learning	Im %
Ailerons	0.000121	0.000125	0.000123	2.9
Elevator	0.0523	0.0543	0.0549	4.4
Kinematics	0.0493	0.0520	0.0507	3.1
Friedman	0.5472	0.7533	0.7512	9.6

C. Experimental Results

In this section we demonstrate the performance of Random-TD and also compare it with other function approximation paradigms. Our goal in these preliminary experiments is not necessarily to demonstrate the superiority of our approach in terms of sample complexity, but rather its availability as an alternate approach to the reinforcement learning function approximation problem. In table II we report the results in terms of the mean absolute Bellman error (BE) for the test data set, with $\epsilon = 0.1$ in all cases.

$$BE = \epsilon_x |y(x) - o(x)|, \tag{11}$$

where ϵ stands for the expected value, and $o(x)$ is the output.

We report the results for different depths of the tree. We initialize each w_i randomly with the constraint $\|w_i = 1\|$. We always initialize each $\theta_i = 0$.

On another set of experiment, we report a comparison results in Table III using Random-TD, TD-learning, and Q-learning. The results, are averaged over 20 independent runs. Although we also compared against backpropagation algorithm, However, We did not report the results of static backpropagation since we could not get results for these data sets using static backpropagation due to the memory limitation. As can be seen in Table III, the Random-TD always performs significantly better than the standard TD. Q-learning has done reasonably well against TD. In comparison to our algorithm, however, Q-learning has not done very well, which can be easily explained. Q-learning is not design for average reward problems, and that it is able to outperform TD. Q-learning was done with the discounting factor set to 0.99.

TABLE IV
CPU TIME TAKEN BY RANDOM-TD OF DIFFERENT DEPTHS FOR DIFFERENT DATA SETS OF FUNCTION APPROXIMATION

Dataset	CPU Time (seconds)				
	Depth=3	Depth=4	Depth=5	Depth=6	Depth=7
Ailerons	2189	4870	6321	11,602	25,498
Elevator	1100	2000	3723	8,844	22,033
Kinematics	892	1566	3211	6609	15,613
Friedman	3319	7900	14,198	33,928	80,067

The learning in the on-line mode relieves function approximator of the memory storage required as in the batch mode. Thus, Random-TD can be effectively used for larger data sets as well, trading memory requirements at the expense of time required to learn the class labels. In Table Ⅲ, we report the CPU time in seconds taken by the Random-TD algorithm for function approximation, including the learning in 200 epochs. We can observe that the CPU time increases exponentially with the depth of the tree. We have noted that Random-TD converges well before 200 epochs, although we report all the results for 200 epochs. The CPU time, as reported in Table I V, taken by the Random-TD can be reduced if the number of epochs is reduced, however, the results indicate we are still very cheap to compute. It is interesting to observe in Table Ⅳ that the Friedman data set took almost over 20 hours for 200 epochs . However, backpropagation could not handle even 5% of this data set for training due to the memory constraint.

V. Conclusion

Learning control problems are surprisingly complex problem. It needs to address how to learn from (possibly delayed) rewards, how to deal with very high-dimensional learning problems, and how to use efficient function approximators with robust performance. Despite the challenges when we aggregate supervised learning with TD-learning, we still reap benefits from both paradigms. From TD-learning we gain the ability to discover behavior that optimizes performance. From supervised learning we gain a flexible way to incorporate domain knowledge. In this paper we propose Random-TD representation as a new model for function approximation to solve problems associated with learning in large state and actions spaces. This reduces the size and complexity of the state and action space, because it causes values to be associated with regions in the state space instead of having to learn a value for each point in the state space. It also Its also contribute to avoid the problem of combinatorial explosion that existing formalisms such as the MDPs suffer when they learn value for combinations of actions. Random-TD is able to learn completely in the on-line mode, and since we do not freeze the learning, it can adapt to a changing situation, provided the rate of change is much slower than the rate of learning. No user-defined is required, the only free parameter of Random-TD is the forest depth, and we demonstrated that after a certain depth, the performance of Random-TD does not vary to a great extent. Our empirical results demonstrate the feasibility and indicate strong potential for this proposed model. Of course, a lot more experimentation is needed to assess the merits of this approach.

References

[1] Sutton, R., & Barto, A. "Reinforcement Learning: An introduction," *Cambring, MA: MIT Press*, 1998.

[2] Sutton, R. "Learning to predict by the method of temporal differences," *Machine Learning*, 3:9-44. 1988.

[3] Leo Breiman, "Bagging predictors," *Machine Learning*, 24(2):123.140, 1996.

[4] Leo Breiman, "Random Forests," *Machine Learning*, 45(1):5.32, 2001.

[5] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone, "Classification and regression trees," *Wadsworth Inc.*, Belmont, California, 1984.

[6] R. Schapire, Y. Freund, P. Bartlett, and W. Lee, "Boosting the margin: a new explanation for the effectiveness of voting methods," *Ann. Statist.*, 26(5):1651-1686, 1998.

[7] Hassab Elgawi Osman, "Online Random Forests based on CorrFS and CorrBE," *In Proc.IEEE workshop on online classification, CVPR*, 2008.

[8] Merz, C., & Murphy, P., UCI repository of machine learning databases, 1996. Available online at http://www.ics.uci.edu/mlearn/MLRepository.html.

[9] Munos, R., & Moore, A. "Variable resolution discretization in Optimal Control," *Machine Learning*, 49, 291-323, 2002

[10] Bertsekas, D., "Dynamic programming and optimal control," *Vol. 1 third edtition. Athena Scientific*

[11] Guvenir, H. A., & Uysal, I., Bilkent University function approximation repository, 2000. Available online at http://funapp.cs.bilkent.edu.tr/DataSets/.

[12] S. Ross., "Applied Probability Models with Optimization Applications," *Holden Day*, San Francisco, California, 1970

[13] R. Bellman. "Dynamic Programming," *Princeton University Press*, Princeton, New Jersey, 1957

[14] Chien, J., Huang, C., & Chen, S. "Compact decision trees with cluster validity for speech recognition," *In IEEE Int. Conf. Acoustics, Speech, and Signal Processing* (pp. 873-876). Piscataway, NJ: IEEE Press, 2002.

[15] Cho, Y. H., Kim, J. K., & Kim, S. H. "A personalized recommender system based on web usage mining and decision tree induction," *Expert Systems with Applications*, 23, 329-342, 2002.

[16] Pyeatt, L. D.,& Howe, A. E. "Decision tree function approximation in reinforcement learning," (Tech. Rep. No. CS-98-112). Fort Collins, CO: Colorado State University, 1998.

[17] Uther,W. T. B., & Veloso,M.M. "Tree based discretization for continuous state space reinforcement learning," *In Proc. Sixteenth National Conference on Artificial Intelligence (AAAI)*. Cambridge, MA: MIT Press, 1998.

[18] Wang, X., & Dietterich, T. "Effficient value function approximation using regression trees," In T. Dean (Ed.), *Proceedings of the IJCAI-99 Workshop on Statistical*, 1999