2006 IEEE International Conference on
Systems, Man, and Cybernetics
October 8-11, 2006, Taipei, Taiwan

Self-Organizing Decision Tree Based on Reinforcement Learning and its Application on State Space Partition

Kao-Shing Hwang, *Member IEEE*, Tsung-Wen Yang, and Chia-Ju Lin

***Abstract*—Most of tree induction algorithms are typically based on a top-down greedy strategy that sometimes makes local optimal decision at each node. Meanwhile, this strategy may induce a larger tree than needed such that requires more redundant computation. To tackle the greedy problem, a reinforcement learning method is applied to grow the decision tree. The splitting criterion is based on long-term evaluations of payoff instead of immediate evaluations. In this work, a tree induction problem is regarded as a reinforcement learning problem and solved by the technique in that problem domain. The proposed method consists of two cycles: split estimation and tree growing. In split estimation cycle, an inducer estimates long-term evaluations of splits at visited nodes. In the second cycle, the inducer grows the tree by the learned long-term evaluations. A comparison with CART on several datasets is reported. The proposed method is then applied to tree-based reinforcement learning. The state spare partition in a critic actor model, Adaptive Heuristic Critic (AHC), is replaced by a regression tree, which is constructed by the proposed method. The experimental results are also demonstrated to show the feasibility and high performance of the proposed system.**

***Index Terms*—Decision trees, reinforcement learning.**

I. Introduction

THE problem of determining a model for a target system by observing the given input-output data pairs is generally referred to as system identification. The purposes of system identification can be organized as follows: 1) to predict a system's output when seeing a new input data. 2) to extract the relationship between input and output of the system, and 3) to design a controller based on the model of system [1].

The decision tree learning is a technique that uses tree structure to solve system identification problem. The process of constructing a decision tree, also known as tree induction, can be viewed as the problem of how to identify parameter in a decision tree. In general, the tree induction is a recursive procedure that assigns an optimal test to each visited node. It has been shown that the problem of constructing optimal decision trees that have minimal number of tests required to classify an unknown sample is a NP-complete problem [2]. Therefore, finding efficient heuristics for constructing near-optimal decision trees is necessary.

In this paper, we propose a self-organizing decision tree induction method that tackles the greedy problem. Our split selection criterion is not only based on the local evaluation but also the consequential accumulated evaluation after a spilt is taken.

Furthermore we apply the proposed method to a tree-based reinforcement learning. A critic actor model, Adaptive Heuristic Critic (AHC) [3], is equipped with a decision tree as its decoder, in which the decision tree is constructed by the proposed method.

The rest of the paper is organized as follows. Decision tree and reinforcement learning is introduced in Section II and Section III. In Section IV and V, we describe our proposed self-organizing decision tree method which is based on reinforcement learning. We demonstrate the experimental results for various data sets in Section VI. In Section VII next, we extend the proposed method for the growing of the decision tree in tree-based reinforcement learning. The simulation results of the tree-based AHC are reported in Section VIII. Conclusions and future research directions are included in Section IX.

II. Decision Tree Learning

Decision tree learning is a method for approximating discrete/continuous functions which are represented by the tree structures. This section describes the structure of decision tree and the typical decision tree constructing procedure.

A. Decision Tree

A decision tree is a tree structure that can approximate a function from a given labeled data set. It consists of two kinds of nodes: internal nodes and leaf nodes. Each internal node contains a test that determines which child node to visit. Each leaf node has a predicted label or value for the input patterns falling in the node.

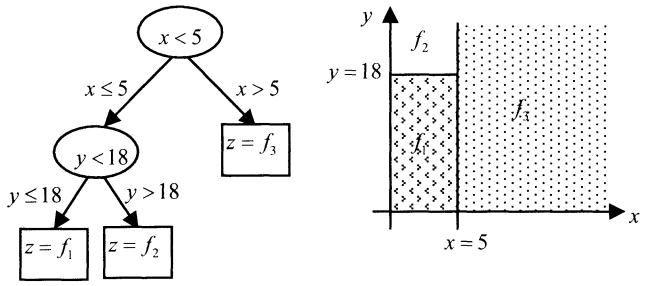
There are two kinds of decision trees. One is called classification tree used for classification problems. The other is called regression trees used for regression problems. In classification trees, each leaf node contains the predicted class for the given input. In regression trees, the value of each leaf node may be specified as a constant or an equation. Fig.1(a) shows a regression tree that approximates a function with two inputs, x and y, and one output z. In Fig.1(b), the input space is partitioned into three non-overlapping regions by the decision tree.

B. Decision Tree Induction

Decision Tree Induction is a procedure that constructs an optimal decision tree from a given labeled data set. Fig. 2 presents a typical algorithm for the top-down decision tree induction. An induction procedure consists of several iterations. At each iteration, the inducer considers the partition of the data

set that corresponds to a node and assigns an appropriate split for the node according to some splitting measure.

Top-down approaches involve three tasks: 1) the selection of a splitting criterion; 2) the decision about which nodes are terminal; 3) the assignment of each terminal node to a label value.



(a) (b)

Fig.1 A binary regression tree and its input space partitioning

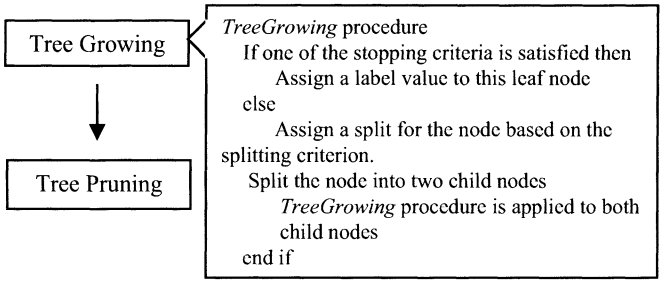


Fig. 2 A typical top-down decision tree induction algorithm

III. Reinforcement Learning Methods

A. Reinforcement Learning Problem

The reinforcement learning problem is a problem of how to learn from interaction to achieve a goal. The problem contains two components: the agent and the environment. The agent interacts with the environment at each of a sequence of discrete time steps. At each time step t, the agent observes a representation of the environment's state, $s_t \in S$, where S is the set of all possible states, and selects an action, $a_t \in A(s_t)$ based on a policy π_t, where $A(s_t)$ is the set of actions available in state s_t. Later the environment transits to a new state s_{t+1} and sends a feedback called reward, $r_{t+1} \in \Re$ to the agent.

B. Reinforcement Learning Methods

1) ***Q-Learning****:* Q-learning [4] is to learn the optimal action-value function $Q^*(s,a)$ that maps each state-action pair to a real value. The algorithm is shown in Fig. 3

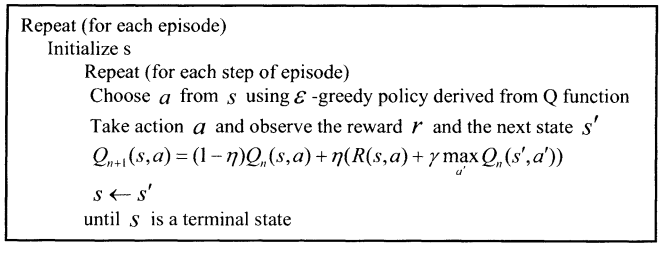


Fig. 3 Q-learning algorithm

2) ***Adaptive Heuristic Critic (AHC)****:* A critic actor model, AHC is proposed by Barto, Sutton, and Anderson [3]. The AHC learning system consists of two adaptive elements: ACE and ASE. ASE implements a policy function $\pi(s_t)$ that maps each state vector s_t into an action. ACE represents a state-value function $v(s_t)$ that evaluates the goodness of each state vector.

IV. Tree-based Reinforcement Learning

Continuous U tree [5] uses decision tree learning techniques to find a discretization of the continuous state space. There are two distinct concepts of state: sensory input I and discrete state s. The first type of state is the continuous state of the world. The second type of state is the position in the discretization. Each discrete state is an area in the sensory input space, and each sensory input falls within a discrete state. The translation between these two concepts of states is implemented by a state tree that describes the discretization. Each leaf node corresponds to an area of sensory input space. Fig. 4 summarizes the algorithm.

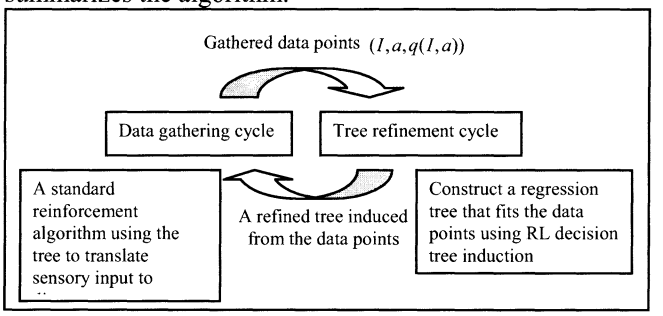


Fig. 4 Continuous U Tree algorithm

V. RL-based Decision Tree Induction

In this section, we give a description of the proposed self-organizing decision tree method, which is based on the reinforcement learning. The method consists of two cycles: split estimation and tree growing. In the first cycle, we model the induction procedure as reinforcement learning problem. The inducer traverses the subtree space to estimate the long-term evaluations of splits of the visited nodes. In the second cycle, the inducer grows the tree in much the same way as the conventional heuristic methods. The difference is that the inducer chooses the split with the maximum long-term evaluation among the available splits. Shown in Fig. 5 is the algorithm.

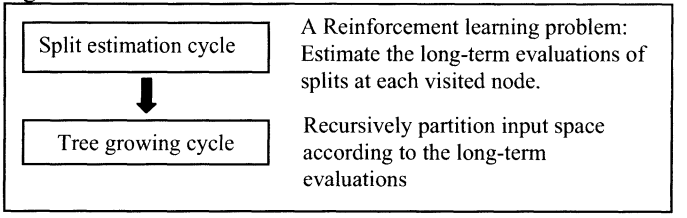


Fig. 5 RL-based decision tree induction

A. Top-down tree induction as decision processes

Top-down tree induction is a procedure that recursively partitions the input space into non-overlapping subspaces. Starting from the root node that contains all the data, the inducer makes sequential decisions about how to split each

visited node into two child nodes. A repository keeps subtrees that are visited but not yet assigned a split by the inducer. In each iteration, the inducer takes a subtree T from the repository and specifies a split for the root node of the subtree. One of the child subtrees caused by the split is pushed into the repository. The other one is taken for the next iteration. The process is repeated until the repository is empty.

A subtree space is defined as a space that contains all possible subtrees. Fig. 6(a) illustrates a subtree space that the inducer traverses in order to specify a test for each visited node sequentially. The circle represents a subtree. The triangle represents a subtree that has been specified a test by the inducer.

Fig. 6(b) shows the subspaces that correspond to the subtrees. Dotted rectangles denote subspaces. Each dashed line denotes the split that partitions the subspace into two smaller subspaces.

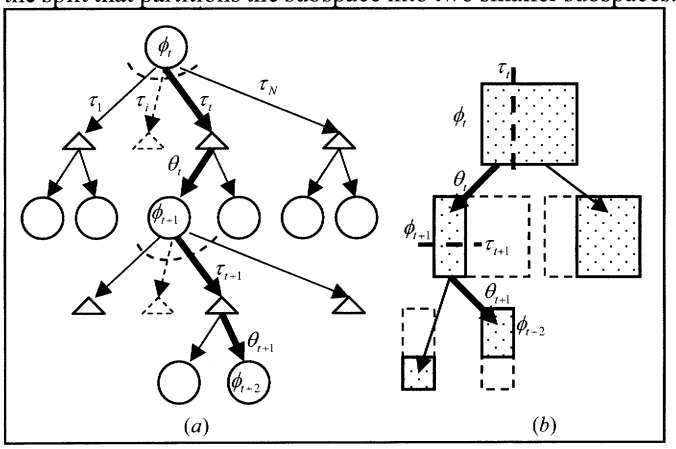


Fig. 6 A subtree space that the inducer traverses

B. Reinforcement learning problem

By representing the tree induction procedure as decision processes, tree growing can be modeled as a reinforcement learning problem. Fig. 7 shows the agent-environment interaction in decision tree induction problem. The agent receives the state representation s_t of the current visited node ϕ_t and chooses a split τ_t from the available split set according to the policy π_t. The split divides the node into two child nodes, $\varphi(\phi_t,\tau_t,0)$ and $\varphi(\phi_t,\tau_t,1)$. A local evaluation r_{t+1}, called reward, for the split corresponds to the next state s_{t+1}. The agent moves to one of child node and traverses the subtree space continually until the agent reaches a node that satisfies the following stopping conditions: 1) the data size is smaller than a threshold; 2) the depth of the node is larger than a threshold; or 3) the inducer determines the node to be a terminal node.

The decision tree learning consists of several episodes. An episode always starts from the state that contains all the training data and ends at the state that satisfies the stopping conditions. A subtree may be visited several times in order to gather the estimation of the splits.

The key elements in reinforcement learning problem are formulated as follows:

State representation

The state representation is a mapping from subtrees to state vectors. The root node of each subtree corresponds to a subspace that encloses the data set of the node. A subspace is defined as $\{x \in R^{N_{attr}} | l_i \le x_i \le u_i, i = 1 \sim N_{attr}\}$ where N_{attr} is the number of the attributes of the data set; l_i and u_i is the lower bound and upper bound of the variable x_i.

The left-top and right-bottom points of a subspace are denoted as x^{lt} and x^{rb}, respectively. The state vector of a subtree ϕ is designed as follows:

$$S(\phi) \equiv (x_1^{lt},\ldots,x_i^{lt},\ldots,x_{N_{attr}}^{lt},x_1^{rb},\ldots,x_i^{rb},\ldots,x_{N_{attr}}^{rb})$$

where x_i^{lt} is a component of x^{lt}; x_j^{rb} is a component of x^{rb}.

In classification problem with N_{attr} attributes, the state vector is then a $2N_{attr}$-dimensional vector.

Action set

The action set $A(s)$ of the state s contains a no-split action and all possible splits at the state. The no-split action is similar to the stopping criterion. It assigns a node to be a terminal node. Typically the split value is the average of the two adjacent data points. For a data set of size M, the number of available split for a numerical variable is less than or equal to M-1. For simplicity, the j-th split value s_{ij} of the i-th numerical variable x_i at a node is formally defined as $s_{ij} = l_i + \frac{u_i - l_i}{N_{split}+1}$j

where $j = 1\ldots N_{split}$; N_{split} is the number of splits of a single variable.

Reward function

The reward function $R(s_t,a_t)$ is designed as the impurity change due to the split a_t when the action is a split action. The reward function $R(s_t,a_t)$ is formally defined as follows:

$$R(s,a) = \begin{cases} \Delta E(s,a) & \text{if } a \text{ is a split action} \\ 0 & \text{if } a \text{ is a no-split action} \end{cases}$$

where $\Delta E(s,a)$ is the impurity change due to the split a in the node ϕ

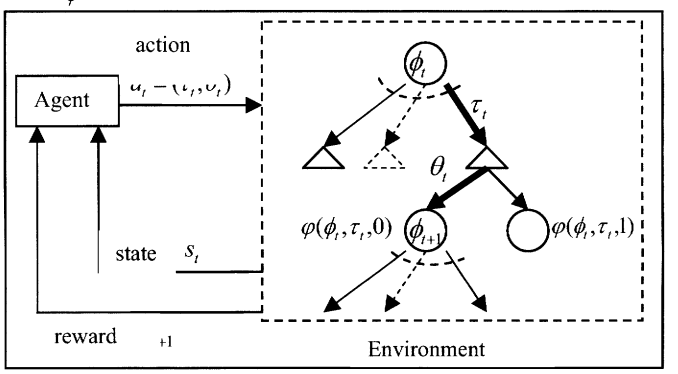


Fig. 7 The agent-environment interaction in reinforcement learning

Action-Value function

The action-value function $Q^{\pi}(s,a)$ is defined as the value taking action a in state s under a policy π. In this case, the

action-value function is equal to the reward plus the sum of maximum action-values of the child nodes under the policy π. We define the action-value function as follows:

$$Q^{\pi}(s,a)=\begin{cases}-E(s) & \text{if } a \text{ is a no-split action}\\ R(s,a)+\sum_{i=0}^{1}\max_{a'\in A(s_c^i)} Q^{\pi}(s_c^i,a') & \text{if } a \text{ is a split action}\end{cases}$$

where $E(s)$ is the impurity function at the node ϕ; s_c^i corresponds to the state vectors of the child nodes caused by the split a.

The split estimation cycle and tree growing cycle is shown in Figs. 8 and 9, respectively.

Split Estimation cycle
Repeat (for each episode):
$s \leftarrow S(\phi)$, ϕ is the node that contains all the data set
Repeat (for each step of episode):
Choose a from s using ε-greedy policy derived from Q function
If a is a split action,
Split the node ϕ and pick one of the child nodes as the next ϕ' with equal probability; $s' \leftarrow S(\phi')$
Else
s is marked as a terminal node; $s' \leftarrow s$
Endif

$$Q_{n+1}(s,a)\leftarrow\begin{cases}(1-\eta)Q_n(s,a)+\eta(-E(s)) & \text{if } a \text{ is a no-split action}\\ (1-\eta)Q_n(s,a)+\eta(R(s,a)+\sum_{i=0}^{1}\max_{a'\in A(s_c^i)} Q_n(s_c^i,a')) & \text{if } a \text{ is a split action}\end{cases}$$

$s \leftarrow s'$
until s' is a terminal node, or s is marked as a terminal node.

Fig. 8 Split estimation cycle

Tree Growing cycle
Initialize ϕ as the node that contains all the data
growTree(ϕ)
end
growTree (ϕ) procedure
$s \leftarrow S(\phi)$
$a \leftarrow \arg\max_{a'} Q(s,a')$
if a is a no-split action
ϕ is determined as a terminal node and is assigned a label value
else
Assign the split a for the node ϕ
growTree(ϕ_c) for all child nodes ϕ_c caused by the split
endif
end

Fig. 9 Tree growing cycle

C. Continuous State Problem

We solve the continuous state problem using the unsupervised adaptive clustering method for approximating the action-value function. We adopt FAST [6] to generate CMAC-like [7] coarse coding with multiple, overlapped grid tilings. Fig. 10 shows a FAST-based function approximator for action-value function.

The output of the function approximator is determined as

$$y(I)=\frac{1}{N_a}\sum_{i=1}^{N}x_i w_i$$

where $I \in R^m$ is the sensory input; N is the number of clusters; N_a is the number of activated clusters;

$$x_i=\begin{cases}1 & \text{if the cluster } i \text{ is activated by sensory input } I\\ 0 & \text{otherwise}\end{cases}$$

The update rule is as follows: $w_i(t+1)=w_i(t)+\alpha(y_d(I)-y(I))x_i$ where $y_d(I)$ is the desired output of the sensory input I; α, $0<\alpha<1$, is the learning rate.

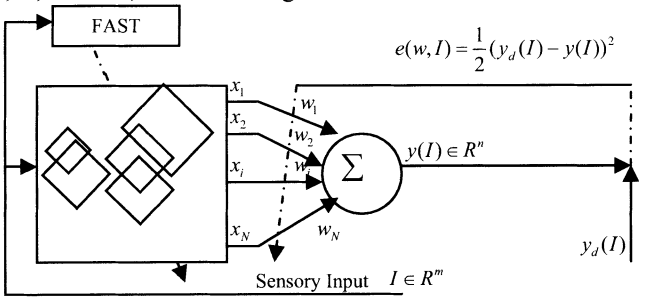


Fig. 10 A FAST-based Function Approximator

VI. Experiments

The experiment compares the performance of the proposed method and CART [8]. CART is a top-down heuristic method that consists of two cycles: tree growing and tree pruning.

A. Datasets

Both methods are evaluated on five datasets with only numerical attributes from the UCI Machine Learning Repository [9]. Table. 1 summarizes the properties of the datasets.

Table.1 The summary of the properties of the datasets

Dataset	Number of Classes	Number of attributes	Record Number (train/test)	Error rate
BUPA liver disorder	2	6	345/0	ten-fold
PIMA Indian diabetes	2	7	532/0	ten-fold
StatLog satellite image	6	36	4435/2000	Test set
Image segmentation	7	19	210/2100	Test set
StatLog vehicle silhouette	4	18	846/0	ten-fold

B. Experimental setup

For each dataset, we use two different ways to estimate the error rate of an algorithm. For datasets with independent test sets, the test set is used to estimate the error rate. The decision tree is constructed by the training set and then is tested on the test set. Two of the four datasets are analyzed this way. For the remaining three datasets, we use the ten-fold cross-validation procedure to estimate the error rate.

C. Results

We compare classification accuracy and sizes of trees of two methods. The size of a decision tree is the number of leaves of the tree. The results of five datasets are listed in Table. 2.

VII. TREE-BASED CRITIC-ACTOR MODEL

A. Critic-Actor Model with the decoder of a regression tree [3]

The objective of this chapter is to apply the tree induction method to the critic-actor model algorithm to achieve automatic state partitioning. The state space is partitioned into different sizes of boxes by a decision tree, which is constructed by the RL decision tree induction during learning procedure. Fig. 11 shows the architecture of a tree-based critic-actor model learning system. The sensory input from the environment is mapped to a box by the decision tree.

Table. 2 Experimental comparisons of RL method of CART

Datasets	RL method		CART	
	Error rates	Tree sizes	Error rates	Tree sizes
BUPA liver disorder	0.301	19	0.4	71
PIMA Indian diabetes	0.20703	40	0.27	112
StatLog satellite image	0.248	36	0.15	328
Image segmentation	0.1926	37	0.09	19
StatLog vehicle silhouette	0.368	32	0.38	118

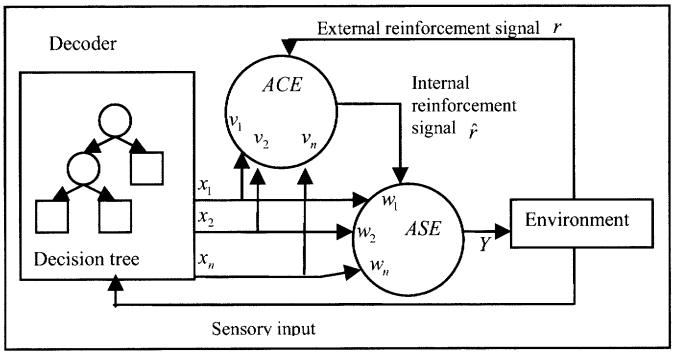


Fig. 11 AHC with the decoder of a regression tree

Like work of continuous U tree method [5], there are two concepts of states defined. One is the sensory input denoted as I and the other is the discrete state denoted as s. Discrete state is translated from sensory input by a decision tree. The method consists of two cycles: data gathering cycle and tree refinement cycle.

We make some changes to AHC algorithm as follows:

1) the weights w_i, v_i of AHC are modified only when the agent enters different boxes.
2) different way to calculations of the internal reinforcement signal.

In the beginning of the learning, the size of a box may be large. In this situation, the agent may spend several steps crossing a box to another neighbor box, shown in Fig.12 as an example. The agent enters the box b_n at time t and leaves at time $t+3$. box. Fig. 13 summarizes the tree-base critic-actor model algorithm. The data gathering cycle and tree refinement cycle are depicted in Fig.4 and Fig.15, respectively.

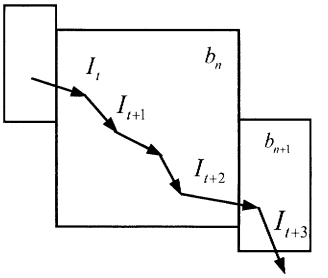


Fig. 12 The agent spends several steps crossing a box to another

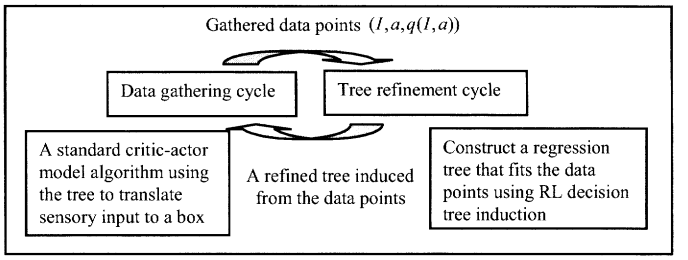


Fig. 13 Tree-based critic-actor model algorithm

VIII. SIMULATION RESULTS

The experiment is to learn to balance the pole without a priori knowledge of dynamics and state partitioning of a cart-pole system. The experiment consists of a sequence of trials. Each trial begins with the cart-pole state $x=0, \dot{x}=0, \theta=0, \dot{\theta}=0$, and ended with a failure signal when θ leaves $[-12^{\circ},12^{\circ}]$ or x leaves the interval $[-2.4,24]$. We reset all the trace variables to zero at the start of each trial. The results of our simulation are shown in Fig. 16. The arrows indicate the end of the epochs. An epoch consists of a data gathering cycle and a tree refinement cycle. The tree refinement cycle is executed after the data gathering cycle. Fig. 17 shows the average of weight changes in each trial. A cycle continues until the average of weight change in a trial is less than a certain threshold. The arrows mean the ends of the cycles.

IX. CONCLUSION

We propose a decision tree induction method which is based on reinforcement learning. We compare the performance of the proposed method with CART. Although we choose the split with the maximum long-term evaluation to avoid the greedy problem, inappropriate selection of the reward function may still induce a non-optimal decision tree. The continuous state problem is always an issue in reinforcement learning. We may try different function approximator methods to improve the performance.

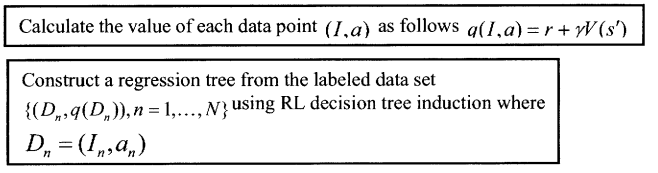


Fig.15 Tree refinement cycle

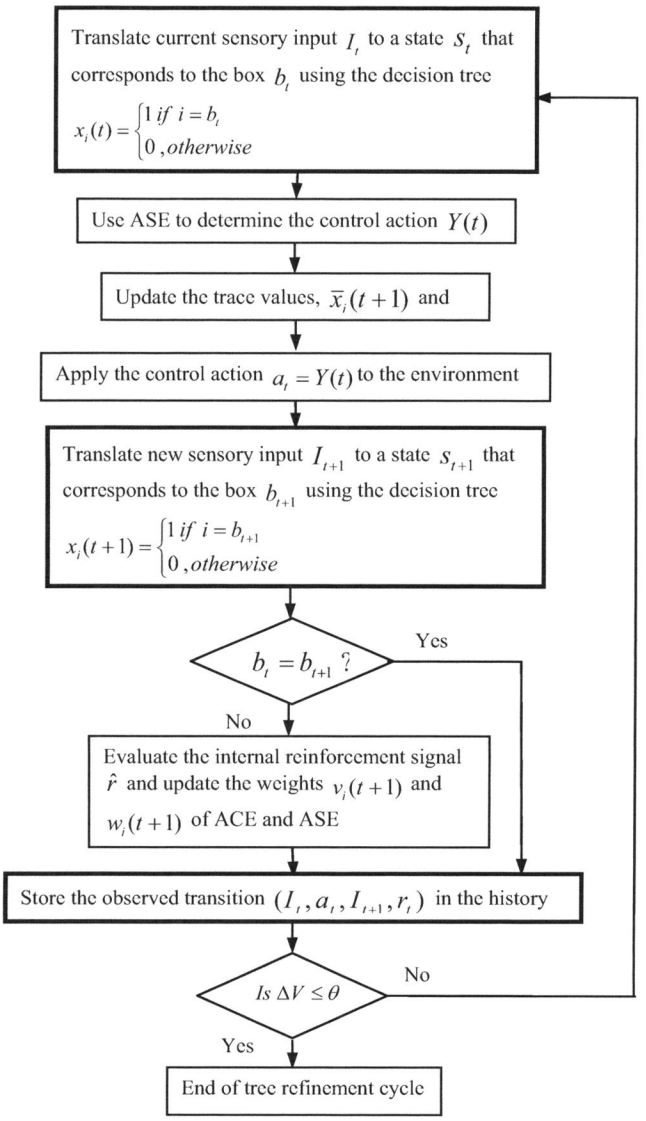


Fig.14 Data gathering cycle

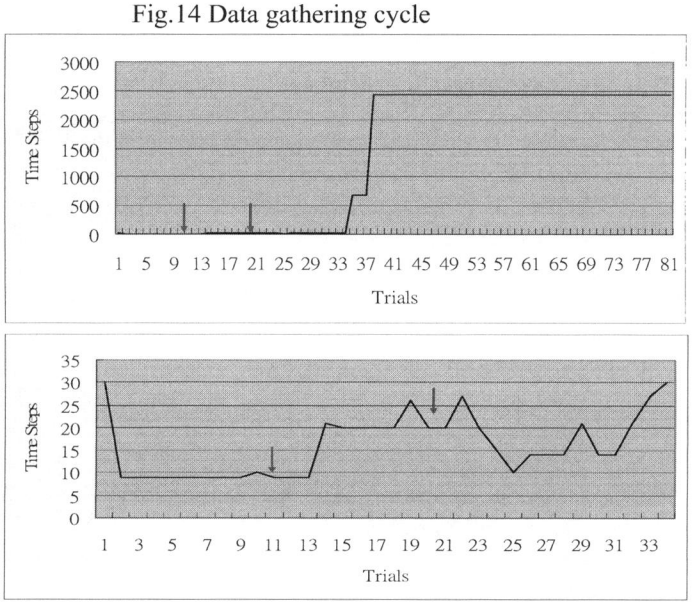


Fig. 16 Simulation results.

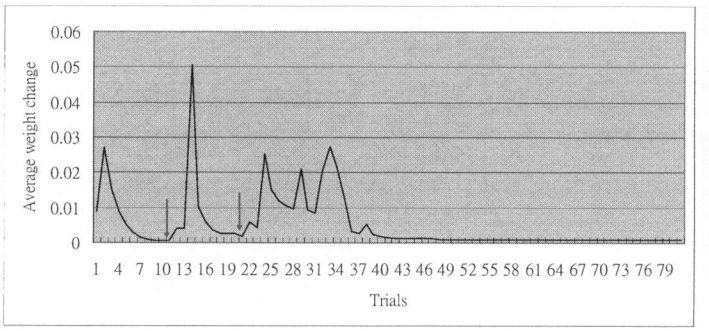


Fig. 17 Average of weight changes

REFERENCES

[1] J. S. R. Jang, C. T. Sun, and E. Mizutani, "Least-Squares Methods for System Identification," in *Neuro-Fuzzy and Soft Computing:a computational approach to learning and machine intelligence,* Prentice-Hall International, Inc, pp. 95-97, 1997.

[2] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is NP-complete," *Information Processing Letter*, vol. 5, 1976.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man and Cybernetics*, pp. 834-846, 1983.

[4] C. Watkins and P. Dayan, "Learning from delayed rewards," University of Cambridge, England, 1989.

[5] William T.B.Uther and Manuela M.Veloso, "Tree Based Discretization for Continuous State Space Reinforcement Learning," In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison,WI , 1998.

[6] A. Perez-Uribe and E. Sanchez, "The FAST architecture: A Neural Network with Flexible Adaptable-Size Topology," In *Proc. 5th International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, Lausanne, Switzerland, pp. 337-340, 1996.

[7] J. S. Albus, "A New Approach to Manipulate Control: The Cerebellar Model Articulation Controller (CMAC)," *Trans. ASME J. Dynamic Systems, Measurement, and Control*, vol. 97, pp. 220-227, 1975.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees,* Wadsworth. Inc., Belmont, California, 1984.

[9] C. J. Merz and P. M. Murphy, "UCI Repository of Machine Learning Databases," Department of Information and Computer Science, University of California, Irvine, CA, 1996.