A Learning Petri Net Model Based on Reinforcement Learning

Liang-Bing Feng, Masanao Obayashi, Takashi Kuremoto and Kunikazu Kobayashi

Graduate School of Science and Engineering, Yamaguchi University, 2-16-1 Tokiwadai, Ube, Yamaguchi, Japan

(Tel : 81-836-85-9518; Fax : 81-836-85-9501)
(n007we, m.obayas,wu,koba@yamaguchi-u.ac.jp)

Abstract: In this paper, a hybrid intelligent control system – Learning Petri Net (LPN) that combines the Petri net and reinforcement learning is presented. LPN is expanded High-level Time Petri nets, in which some transition's input arc weight function and transition delay time have a value item which records the reward from environment. Based on interaction with environment, LPN can adjust the arc weight function and transition delay time when it's modeling system is running. The arc weight function and transition delay time learning algorithm is based on Q-learning – a kind of Reinforcement Learning (RL). Finally, for the purpose of certification of the effectiveness of our proposed Learning Petri net, it is used to model a discrete event dynamic control system – Sony AIBO learning control system as an example. The result of the experiment shows this method is correct and effective.

Keywords: Petri Net, Reinforcement Learning, Discrete event dynamic system.

I. INTRODUCTION

Petri nets combine a well defined mathematical theory with a graphical representation of the dynamic behavior of systems. The theoretic aspect of Petri nets allows precise modeling and analysis of system behavior, while the graphical representation of Petri nets enable visualization of the modeled system state changes [1]. So, Petri net is widely used to model various dynamic systems. But the Petri net doesn't have the learning capability, all the parameters which describe the system characteristic need to be set individually and empirically when dynamic system is modeled. Recently, there are some researches for making the Petri net has learning capability. In paper [2], the global variables are used to record all state of colored Petri net when it is running. The global variables are optimized and colored Petri net is updated according to these global variables. Here, colored Petri net is only a tool of learning. A learning Petri net model which combines Petri net with neural network is proposed in [3]. This learning Petri net model can realize an input-output mapping through Petri net's weight function is adjusted just like an artificial neural network. And this learning method was applied to nonlinear system control. Paper [4] created a Time Interval Petri net for modeling real-time decision making. It makes the Petri net have learning capability through converting arc's type, adding a record to an output arc operator table. Based on these researches, a learning Petri net model based on reinforcement learning is proposed in this paper.

The rest of this paper is organized as followed. Section 2 gives the definition of LPN and illustrates the learning algorithm of LPN. Section 3 describes the application to discrete event dynamic system control and gives the simulation result. Finally, Section 4 summarizes the paper and discusses some directions for future work.

II. THE LEARNING PETRI NET AND LEARNING ALGORITHM

Learning Petri nets is Petri nets which have learning capability.

1. Definition of Learning Petri net

Learning Petri net is constructed based on High-Level Time Petri Net (HLTPN).

Definition 1: HLTPN has a structure $HLTPN = (NG, C, W, DT, M_0)$ [5], where

(i). $NG = (P, T, F)$ is called net graph with P that is a finite set of nodes, called Places. $ID{:}P \rightarrow N$ is a function marking P, $N = (1,2,\ldots)$ is the set of natural number. Using $P_1, P_2, \ldots, P_n$ represents the elements of P and n is the cardinality of set P;

T is a finite set of nodes, called Transitions, which disjoint from P, $P \cap T = \varnothing$; $ID{:}T \rightarrow N$ is a function marking T. Using $T_1, T_2, \ldots, T_m$ represents the elements of T, m is the cardinality of set T;

$F \subseteq (P \times T) \cup (T \times P)$ is a finite set of directional arcs, known as the flow relation;

(ii). C is a finite and non-empty color set for describing difference type data;

(iii). W: $F \rightarrow C$ is a weight function on F. If $F \subseteq (P \times T)$, the weight function W is W_{in} that decides which colored Token can go through the arc and enable T fire. This color tokens will be consumed when transition is fired. If $F \subseteq (T \times P)$, the weight function W is W_{out} that decides which colored Token will be generated by T and be input to P.

(iv). DT: $T \rightarrow N$ is a delay time function of a transition which has a *Time* delay for an enable transition fired.

(v). M_0: $P \rightarrow \mathrm{U}_{p \in P}\,\mu C(p)$ such that $\forall\, p \in P$, $M_0(p) \in \mu C(p)$ is the initial marking function which associates a multi-set of tokens of correct type with each place.

Now, we give the definition of LPN.

Definition 2: Learning Petri Net has a structure, *LPN= (HLTPN, VW, VT)*, where

(i). *HLTPN=* (*NG, C, W, DT,* M_0) is a High-Level Time Petri Net.

(ii). *VW* (the value of weight function): $W_{in} \rightarrow R$, is a function marking on W_{in}. A $F \subseteq (P \times T)$ has several W_{in} and every W_{in} has a reward value $R \in$real number.

(ii). *VT* (the value of delay time): $DT \rightarrow R$, is a function marking on *DT.* A transition has several *DT* and every *DT* has a reward value $R \in$real number.

Using LPN, a mapping of input-output Token is gotten. For example, we construct a LPN which is showed in Fig. 1. Colored Tokens C_{ij} $(i=1;j=1,2...n)$ are input to P_1 by T_{input}. There are n weight functions $W(<C_{ij}>, VW_{Cij,i,j})$ on a same $F_{i,j}$. Token C_{ij} obeys weight functions $W(<C_{ij}>, VW_{Cij,i,j})$ that is decided by the value $VW_{Cij,i,j}$. After Token C_{ij} passed through arc $F_{i,j}$ $(i=1;j=1,2...n)$, $T_{i,j}(i=1;\ j=\ 1,2\ ...n)$ fires and generates Tokens $C_{ij}(i=2;j=1,2...n)$ in P_2. After P_2 has color Token $C_{ij}(i=2;j=1,2...n)$, $T_{i,j}(i=2;\ j=\ 1,2\ ...n)$ fires and different colored Token $C_{ij}(i=3;j=1,2...n)$ is generated. Then, a reward will be gotten from environment according to whether it accords with system rule that C_{3j} generated by C_{1j}. These rewards are propagated to every $VW_{Cij,i,j}$ using algorithm 1.

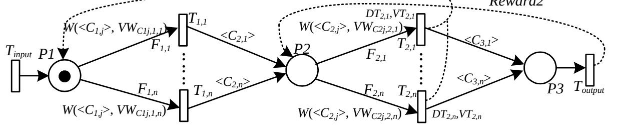

Fig.1. A Learning Petri Net.

Using LPN to model a dynamic system, the system state is modeled as Petri net place and the system state change is modeled as transition. Several characteristics parameters of system state and action can be expressed by Token number and color, arc weight function, transition delay time, and so on. When the system is modeled, some characteristics are unknown. So, these characteristics are set randomly and are gotten gradually and appropriately when the system runs. So, we use Learning Petri net to learn the system condition from real-time situation.

2. Learning algorithm of Learning Petri net

In Learning Petri net, RL is used to learn the *VM* and *VT* through interacting with environment. RL could learn the optimal policy of the dynamic system through environment state observation and improve its behavior through trial and error with the environment. RL agent senses the environment and takes actions. It receives numeric award and punishments from some reward function. The agent learns to choose actions to maximize a long term sum or average of the future reward it will receive [6].

In this paper, the arc weight function and transition delay time learning algorithm are based on Q-learning – a kind of RL. In arc weight function learning algorithm, $VW_{Cij,i,j}$ is randomly set firstly. So, the weight function on the arc is arbitrary. When the system runs, we use formula (1) to update $VW_{Cij,i,j}$.

$$VW_{Cij,i,j} = VW_{Cij,i,jj} + \alpha\,[r + \gamma\overline{(VW_{ci+1j,i+1,j}})\ -\ VW_{Cij,i,j}],\ (1)$$

where, (i). α is the step-size, γ is discount rate.

(ii). r is reward which $W(<C_{ij}>, VW_{Cij,i,j})$ gets when $T_{i,j}$ is fired by $<C_{ij}>$. Here, because environment only gives system reward at last step, so a feedback learning method is used. If $W(<C_{ij}>, VW_{Cij,i,j})$ through $T_{i,j}$ generated Token $<C_{i+1,j}>$ and $W(<C_{i+1j}>, VW_{Ci+1j,i+1,j})$ through $T_{i+1,j}$ generated Token $<C_{i+2,j}>$, $VW_{Ci+1j,i+1,j}$ gets an update value. And this value is feedback as $W(<C_{ij}>, VW_{Cij,i,j})$ next time reward r.

(iii). $\overline{(VW_{ci+1j,i+1,j})}$ is calculated from all $W(<C_{i+1j}>, VW_{Ci+1j,i+1,j})$ feedback value as formula (2).

$$\overline{(VW_{ci+1j,i+1,j})}_{t} = \gamma\overline{(VW_{ci+1j,i+1,j})}_{t-1} + r_t, \qquad (2)$$

where t is time for that $<C_{i+1j}>$ is generated by $W(<C_{ij}>, VW_{Cij,i,j})$.

In the transition delay time learning algorithm, the reward can learn immediately. $VT_{i,j}$ can be updated using formula (3).

$$VT_{i,j} = VT_{i,j} + \alpha\,[r + \gamma\overline{R} - VT_{i,j}]\ , \qquad (3)$$

where, r is reward that an action get after it acted. $\overline{R}$ is average of all the reward. α is the step-size, γ is discount rate.

Now, we found the algorithm of Learning Petri net which is listed in Table 1.

Table 1. Learning Algorithm for Learning Petri Net

```
Algorithm 1    Weight function learning algorithm
 Step 1. Initialization: Set all $VW_{ij}$ and $r$ of all input arc's
         weight function to zero.
 Step 2. Initialize learning Petri net. i.e. make the Petri net
         state as $M_0$.
   Repeat i) and ii) until system becomes end state.
     i) When a place gets a colored Token $C_{ij}$, there is a choice that which arc weight function is obeyed if the functions include this Token. This choice is according to selection policy which is ε greedy (ε is set according to execution environment by user, usually $0<\varepsilon<<1$).
     A: Select the function which has the biggest $VW_{cij,i,j}$ at probability $1-\varepsilon$;
     B: Select the function randomly at probability $\varepsilon$.
     ii)The transition which the function correlates is fired and reward is observed. Adjust the weight function value using $VW_{Cij,i,j} = VW_{Cij,i,jj} + a[r+\gamma(\overline{VW_{ci+1j,i+1,j}})$ - $VW_{Cij,i,j}]$. At the same time, $a[r+\gamma(\overline{VW_{ci+1j,i+1,j}})$ - $VW_{Cij,i,j}]$ is fed back to weight function with generate $C_{ij}$ as its reward for next time.

Algorithm 2    Delay time learning algorithm
 Step 1. Initialization: Set all $VT_{ij}$ of transition to zero.
 Step 2. Initialize learning Petri net. i.e. make the Petri net
         state as $M_0$.
   Repeat i) and ii) until system becomes end state.
     i) When a transition is fired, choose a delay time using selection policy which is ε greedy (ε is set according to execution environment by user, usually $0<\varepsilon<<1$).
        A: Select delay time which has the biggest $VT_{ij}$ of service at probability $1-\varepsilon$;
        B: Select the function randomly at probability $\varepsilon$.
     ii) After transition fires and reward is observed, the weight function value is adjusted using $VT_{i,j}= VT_{i,j}+ a[r+\gamma\overline{R}$ - $VT_{i,j}]$.
```

III. APPLICATION TO DISCRETE EVENT DYNAMIC SYSTEM CONTROL

In this section, we will apply the LPN to control a discrete event dynamic system.

1. Discrete event dynamic system and AIBO voice command recognition system

A discrete event dynamic system is a discrete-state, event-driven system of which the state evolution depends entirely on the occurrence of asynchronous discrete events over time [7]. Petri nets have been used to model various kinds of dynamic event-driven systems like computers networks, communication systems, and so on. In this paper, it is used to model Sony AIBO learning control system for the purpose of certification of the effectiveness of our proposed Learning Petri net.

AIBO (Artificial Intelligence roBOt) is a type of robotic pets designed and manufactured by Sony. AIBO is able to execute different actions, such as go ahead, move back, sit down, stand up and cry, and so on. And it can "listen" voice via microphone. A command and control system will be constructed for making AIBO understand several human voice commands by Japanese and English and take corresponding action.

The simulation system is developed on Sony AIBO's OPEN-R (Open Architecture for Entertainment Robot) [8]. The architecture of simulation system is showed in Fig. 2. Because there are English and Japanese voice commands for same AIBO action, the partnerships of voice and action are established in part (4). The lasted time of an AIBO action is learning in part (5). After an AIBO action finished, the rewards for action and action lasted time correctness are given by that different AIBO's sensors are touched.

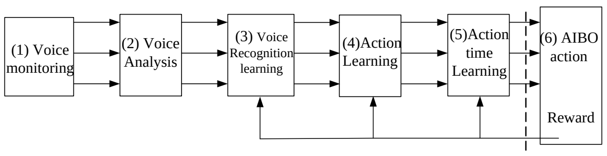

Fig.2 The system architecture of voice command recognition

2. Learning Petri net model for AIBO voice command recognition system

In the Learning Petri net model for AIBO voice command recognition system, AIBO action change, action time are modeled as transition, transition delay, respectively. The human voice command is modeled by different color Token. The LPN model is showed in Fig. 3. The meaning of every transition is listed below:

T_{input} change voice signal as colored Token which describe the voice characteristic.

T_{11}, T_{12} and T_{13} can analyze the voice signal. T_1 generates 35 different Token VL_1….VL_{35} according to the voice length. T_2 generates 8 different Token $E2_1$...$E2_8$ according to the front twenty voice sample energy characteristic. T_3 generates 8 different Token $E4_1$...$E4_8$ according to the front forty voice sample energy characteristic [9]. These three types Token are compounded into a compound Token $<VL_l>$ + $<VE2_m>$ + $<VE4_n>$ in P_2 [10].

T_{2j} generates the different voice Token. The input arc's weight function is $((<VL_l>+<VE_{2m}>+ <VE_{4n}>)$,

$VW_{Vlmn,2j}$) and the output arc's weight function is different voice Token. And voice Token will generate different action Token through T_{3j}.

When $P_4 - P_8$ has Token, AIBO's action will last. T_{4j} takes Token out from $P_4 - P_8$, and makes corresponding AIBO action terminates. T_{4j} has a delay time DT_{4i}, and every DT_{4i} has a value VT_{4i}. Transition adopts which delay time DT_{4i} according to VT_{4i}.

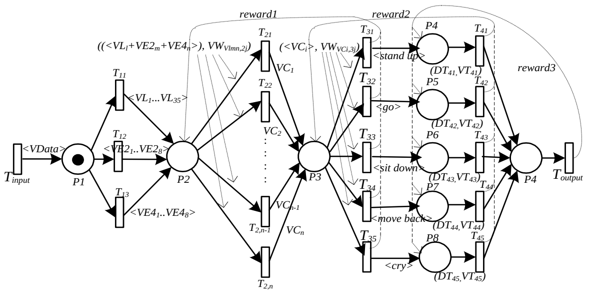

Fig. 3 The LPN model of voice command recognition

3. The results of simulation

When the system begins running, it can't recognize the voice commands. A voice command comes and it is changed into a compound Token in P_2. This compound Token will randomly generate a voice Token and puts into P_3. This voice Token randomly arouses an action Token. A reward for action correctness is gotten, then, *VW* and *VT* are updated. For example, a compound colored Token $(<VL_l> + <VE_{2m}> + <VE_{4n}>)$ fired T_{21} and colored Token VC_1 is put into P_3. VC_1 fires T_{32} and AIBO acts "go". A reward is gotten according to correctness of action. $VW_{VC1,32}$ is updated by this reward and $VW_{VC1,32}$ updated value is fed back to P_2 as next time reward value of $(<VL_l> + <VE_{2m}> + <VE_{4n}>)$ fired T_{21}. After an action finished, a reward for correctness of action time is gotten and *VT* is updated.

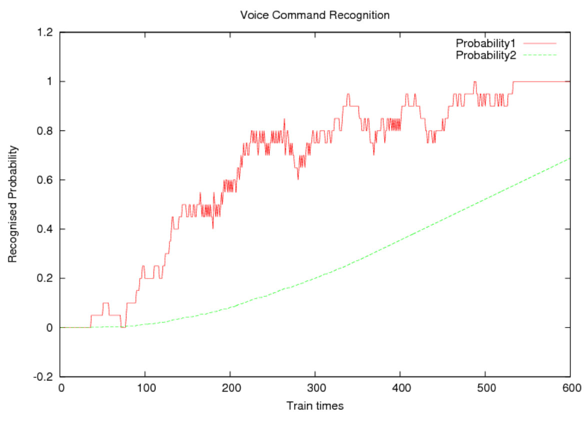

Fig. 4 Relation of training times and recognition probability

Fig. 4 shows the relation of training times and voice command recognition probability. Probability 1 shows the successful probability of recently 20 times training. Probability 2 shows the successful probability of total training times. From the result of simulation, we get that Learning Petri net is correct and effective using the AIBO voice command control system.

IV. CONCLUSION

In this paper, we proposed a hybrid intelligent control system – Learning Petri Net (LPN) that combines the Petri net and reinforcement learning. The definition of LPN is given and the learning algorithm is constructed. For the purpose of certification of the effectiveness of our proposed Learning Petri net, it is used to model Sony AIBO learning control system as an example. The result of the experiment shows this method is correct and effective.

We plan to use reinforcement learning algorithm to adjust other parameter of Petri net and extend our work to model other dynamic system in the future.

REFERENCES

[1]J. Wang (2007), Petri nets for dynamic event-driven system modeling. Handbook of Dynamic System Modeling, Ed: Paul Fishwick, CRC Press: 1-17
[2]V. Baranaushas, K. Sarkauskas (2006), Colored Petri Nets-Tool for control system Learning. Electronics and Electrical Engineering, 4(68):41-46
[3]Hirasawa K., Ohbayashi M., Sakai S, Hu J (1998), Learning Petri network and its application to nonlinear system control. IEEE Transactions on systems, man and cybernetics. Part B: Cybernetics, 28(6):781-789
[4]Vadim B., David C. W. (2005), Machine Learning for Time Interval Petri Nets. In Australian Joint Conference on Artificial Intelligence, Springer-Verlag: 959-965
[5]Guangming C., Minghong L., Xianghu W. (2006), The Definition of Extended High-level Time Petri Nets. Journal of Computer Science, 2(2):127-143
[6]R. S. Sutton, A. G. Batto (1998), Reinforcement learning: An Introduction. The MIT Press. Cambridge, Massachusetts, USA
[7]B. Hrúz, M.C. Zhou (2007), Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and Other Tools. Springer Press, London, UK
[8]OPEN-R programming group (2004), OPEN-R programming introduction. Sony corporation, Japan
[9]Frederick JelinekR (1999), Statistical Methods for Speech. The MIT Press. Cambridge, Massachusetts, USA
[10]H.S.Yan,J.Jian (1999), Agile concurrent engineering. Integrated Manufacturing Systems, 10(2): 103-113