We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 7

Construction and Application of Learning Petri Net

Liangbing Feng, Masanao Obayashi,
Takashi Kuremoto and Kunikazu Kobayashi

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48398

1. Introduction

Petri nets are excellent networks which have great characteristics of combining a well-defined mathematical theory with a graphical representation of the dynamic behavior of systems. The theoretical aspect of Petri nets allows precise modeling and analysis of system behavior, at the same time, the graphical representation of Petri nets enable visualization of state changes of the modeled system [32]. Therefore, Petri nets are recognized as one of the most adequate and sound tool for description and analysis of concurrent, asynchronous and distributed dynamical system. However, the traditional Petri nets do not have learning capability. Therefore, all the parameters which describe the characteristics of the system need to be set individually and empirically when the dynamic system is modeled. Fuzzy Petri net (FPN) combined Petri nets approach with fuzzy theory is a powerful modeling tool for fuzzy production rules-based knowledge systems. However, it is lack of learning mechanism. That is the significant weakness while modeling uncertain knowledge systems.

At the same time, intelligent computing is taken to achieve the development and application of artificial intelligence (AI) methods, i.e. tools that exhibit characteristics associated with intelligence in human behaviour. Reinforcement Learning (RL) and artificial neural networks have been widely used in pattern recognition, decision making, data clustering, and so on. Thus, if intelligent computing methods are introduced into Petri nets, this may make Petri nets have the learning capability, and also performance and the applicable areas of Petri nets models will be widely expanded. The dynamic system can be modeled by Petri nets with the learning capability and then the parameters of the system can be adjusted by online (data-driven) learning. At the same way, if the generalized FPNs are expanded by adding neural networks and their leaning

capability, then FPNs are able to realize self-adapting and self-learning functions. Consequently, it achieves automatic knowledge reasoning and fuzzy production rules learning.

Recently, there are some researches for making the Petri net have learning capability and making it optimize itself. The global variables are used to record all state of colored Petri net when it is running [22]. The global variables are optimized and colored Petri net is updated according to these global variables. A learning Petri net model which combines Petri net with a neural network is proposed by Hirasawa et al., and it was applied to nonlinear system control [10]. In our former work [5, 6], a learning Petri net model has been proposed based on reinforcement learning (RL). RL is applied to optimize the parameters of Petri net. And, this learning Petri net model has been applied to robot system control. Konar gave an algorithm to adjust thresholds of a FPN through training instances [1]. In [1], the FPN architecture is built on the connectionism, just like a neural network, and the model provides semantic justification of its hidden layer. It is capable of approximate reasoning and learning from noisy training instances. A generalized FPN model was proposed by Pedrycz et al., which can be transformed into neural networks with OR/AND logic neuron, thus, parameters of the corresponding neural networks can be learned (trained) [24]. Victor and Shen have developed a reinforcement learning algorithm for the high-level fuzzy Petri net models [23].

This chapter focuses on combining the Petri net and fuzzy Petri net with intelligent learning method for construction of learning Petri net and learning fuzzy Petri net (LFPN), respectively. These are applied to dynamic system controls and a system optimization. The rest of this paper is organized as follow. Section 2 elaborates on the Learning Petri net construction and Learning algorithm. Section 3 describes how to use the Learning Petri net model in the robots systems. Section 4 constructs a LFPN. Section 5 shows the LFPN is used in Web service discovery problem. Section 6 summarizes the models of Petri net described in the chapter and results of their applications and demonstrates the future trends concerned with Learning Petri nets.

2. The learning Petri net model

The Learning Petri net (LPN) model is constructed based on high-level time Petri net (HLTPN). The definition of HLTPN is given firstly.

2.1. Definition of HLTPN

HLTPN is one of expanded Petri nets.

Definition 1: HLTPN has a 5-tuple structure, $HLTPN = (NG, C, W, DT, M_0)$ [9], where

i. $NG = (P, Tr, F)$ is called "net graph" with P which called "Places". P is a finite set of nodes. ID: $P \rightarrow N$ is a function marking P, $N = (1, 2, \ldots)$ is the set of natural number. $p_1, p_2, \ldots, p_n$ represents the elements of P and n is the cardinality of set P;

Tr is a finite set of nodes, called "Transitions", which disjoints from P, $P \cap Tr = \varnothing$; $ID: Tr \rightarrow N$ is a function marking Tr. tr_1, tr_2, …, tr_m represents the elements of Tr, m is the cardinality of set Tr;

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

ii. C is a finite and non-empty color set for describing different type of data;
iii. $W: F \rightarrow C$ is a weight function on F. If $F \subseteq (P \times Tr)$, 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 $F \subseteq (Tr \times P)$, the weight function W is W_{out} that decides which colored Token will be generated by T and be input to P.
iv. DT: $Tr \rightarrow R$ is a delay time function of a transition which has a *Time* delay for an enable transition fired or the fire of a transition lasting time.
v. M_0: $P \rightarrow U_{p \in P} \mu C(p)$ 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.

2.2. Definition of LPN

In HLTPN, the weight functions of input and output arc for a transition decide the input and output token of a transition. These weight functions express the input-output mapping of transitions. If these weight functions are able to be updated according to the change of system, modeling ability of Petri net will be expanded. The delay time of HLTPN expresses the pre-state lasting time. If the delay time is able to be learnt while system is running, representing ability of Petri net will be enhanced. RL is a learning method interacting with a complex, uncertain environment to achieve an optimal policy for the selection of actions of the learner. RL suits to update dynamic system parameters through interaction with environment [18]. Hence, we consider using the RL to update the weight function and transition's delay time of Petri net for constructing the LPN. In another word, LPN is an expanded HLTPN, in which some transition's input arc weight function and transition delay time have a value item which records the reward from the environment.

Definition 2: LPN has a 3-tuple structure, $LPN = (HLTPN, VW, VT)$, where

i. $HLTPN = (NG, C, W, DT, M_0)$ is a High-Level Time Petri Net and $NG = (P, Tr, F)$.
ii. VW (value of weight function): $W_{in} \rightarrow R$, is a function marking on W_{in}. An arc $F \subseteq (P \times Tr)$ has a set of weight function W_{in} and each W_{in} has a reward value item $VW \in$real number.
iii. VT (value of delay time): DT→R, is a function marking on DT. A transition has a set of DT and each DT has a reward value item $VT \in$real number.

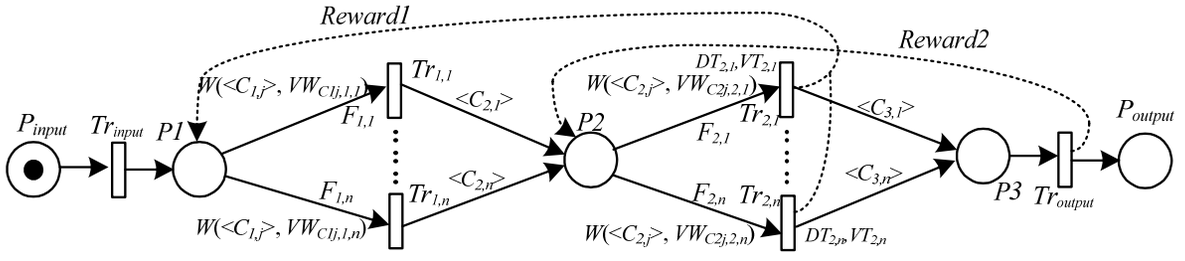


Figure 1. An example of LPN model

An example of LPN model is shown in Figure 1 Using LPN, a mapping of input-output tokens is gotten. For example, in Figure 1, colored tokens C_{ij} (i=1;j=1, 2, …, n) are input to P_1 by Tr_{input}. There are n weight functions $W(<C_{1j}>, VW_{C1j,1,j})$ on a same arc $F_{1,j}$. it is according to the value $VW_{Cij,i,j}$ that token C_{1j} obeys what weight functions in $W(<C_{ij}>, VW_{Cij,i,j})$ to fire a transition. After token C_{1j} passed through arc $F_{i,j}$ (i=1; j=1, 2, …, n), one of $Tr_{i,j}$ (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), $Tr_{i,j}$ (i=2; j=1, 2, …, n) fires and different colored Token C_{ij} (i=3; j=1, 2, …, n) is generated. Then, a mapping of $C_{1j} – C_{3j}$ is gotten. At the same time, 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 $VW_{Cij,i,j}$ and adjust the $VW_{Cij,i,j}$. After training, the LPN is able to express a correct mapping of input-output tokens.

Using LPN to model a dynamic system, the system state is modeled as Petri net marking which is marked for a set of colored token in all places of Petri net, and the change of the system state (i.e. the system action) is modeled as fired of transitions. Some parameters of system can be expressed as token number and color, arc weight function, transition delay time, and so on. For example, different system signals are expressed as different colored of token. When the system is modeled, some parameters are unknown or uncertain. So, these parameters are set randomly. When system runs, the system parameters are gotten gradually and appropriately through system acting with environment and the effect of RL.

2.3. Learning algorithm for LPN

In LPN, there are two kinds of parameters. One is discrete parameter –– the arc's weight function which describes the input and output colored tokens for transition. The other is continuous parameter –– the delay time for the transition firing. Now, we will discuss two kinds of parameters which are learnt using RL.

2.3.1. Discrete parameter learning

In LPN, RL is used to adjust *VW* and *VT* through interacting with environment. RL could learn the optimal policy of the dynamic system through environment state observation and improvement of 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.

The arc weight function learning algorithm is based on Q-learning – a kind of RL [18]. 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, formula (1) is used to update $VW_{Cij,i,j}$.

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

where,

i. *á* is the step-size,is a discount rate.

ii. *r* is reward which $W(<C_{ij}>, VW_{Cij,i,j})$ gets when $Tr_{i,j}$ is fired by $<C_{ij}>$. Here, because environment gives system reward at only last step, so a feedback learning method is used. If $W(<C_{ij}>, VW_{Cij,i,j})$ through $Tr_{i,j}$ generates Token $<C_{i+1,j}>$ and $W(<C_{i+1j}>, VW_{Ci+1j,i+1,j})$ through $Tr_{i+1,j}$ generates 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 feedback value of all $W(<C_{i+1j}>, VW_{Ci+1j,i+1,j})$ as formula (2).

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

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

When every weight function of input arc of the transition has gotten the value, each transition has a value of its action. The policy of the action selection needs to be considered. The simplest action selection rule is to select the service with the highest estimated state-action value, i.e. the transition corresponding to the maximum $VW_{Cij,i,j}$. This action is called a greedy action. If a greedy action is selected, the learner (agent) exploits the current knowledge. If selecting one of the non-greedy actions instead, agent intends to explore to improve its policy. Exploitation is to do the right thing to maximize the expected reward on the one play; meanwhile exploration may produce the greater total reward in the long run. Here, a method using near-greedy selection rule called ε-greedy method is used in action selection; i.e., the action is randomly selected at a small probability ε and selected the action which has the biggest $VW_{cij,i,j}$ at probability 1−ε. Now, we show the algorithm of LPN which is listed in Table 1.

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 the 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<ε<<1).
   A: Select the function which has the biggest $VW_{cij,i,j}$ at probability1-ε;
   B: Select the function randomly at probability *ε*.

ii. The transition which the function correlates fires and reward is observed. Adjust the weight function value using $VW_{Cij,i,j} = VW_{Cij,i,jj} + \alpha[r + \gamma(\overline{VW_{ci+1j,i+1,j}}) - VW_{Cij,i,j}]$. At the same time, $\alpha[r + \gamma(\overline{VW_{ci+1j,i+1,j}}) - VW_{Cij,i,j}]$ is fed back to the weight function with generated C_{ij} as its reward for next time.

Table 1. Weight function learning algorithm in LPN

2.3.2. Continuous parameter learning

The delay time of transition is a continuous variable. So, the delay time learning is a problem of RL in continuous action spaces. Now, there are several methods of RL in continuous spaces: discretization method, function approximation method, and so on [4]. Here, discretization method and function approximation method are used in the delay time learning in LPN.

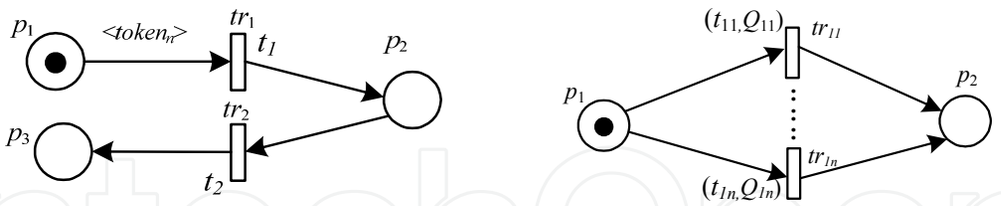
Discretization method

As shown in Figure 2 (i), the transition tr_1 has a delay time t_1. When p_1 has a token $<token_n>$, the system is at a state that p_1 has a Token. This time transition tr_1 is enabled. Because tr_1 has a delay time t_1, tr_1 doesn't fire immediately. After passing time t_1 and tr_1 fires, the token in p_1 is taken out and this state is terminated. Then, during the delay time of tr_1, the state that p_1 has a token continues.

Because the delay time is a continuous variable, the different delay time is discretized for using RL to optimize the delay time. For example, tr_1 in Figure 2 (i) has an undefined delay time t_1. Tr_1 is discretized into several different transitions which have different delay times (shown in Figure 2 (ii)) and every delay time has a value item Q. After Tr_1 fired at delay time t_{1i}, it gets a reward r immediately or after its subsequence gets rewards. The value of Q is updated by formula (3).

$$Q(P,Tr) \leftarrow Q(P,Tr) + \alpha[r + \gamma Q(P',Tr') - Q(P,Tr)] \tag{3}$$

where, $Q(P, Tr)$ is value of transition Tr at Petri net state P. $Q(P',Tr')$ is value of transition T' at next state P' of P. α is a step-size, γ is a discount rate.



(i) The high-level time Petri net model (ii) The discretization learning model for the delay time

Figure 2. Transformation form from high-level Petri net to the learning model

After renewing of Q, the optimal delay time will be selected. In Figure 2 (ii), when $tr_{11},\ldots,tr_{1n}$ get value $Q_{11},\ldots,Q_{1n}$,respectively, the transition is selected by the soft-max method according to a probability of Gibbs distribution.

$$\Pr\{t_t=t \mid p_t=p\} = \frac{e^{\beta Q(p,t)}}{\sum_{b\in A} e^{\beta Q(p,b)}} \tag{4}$$

where, $\Pr\{t_t=t \mid p_t=p\}$ is a probability selecting of transition t at state p, $\hat{a}$ is a positive inverse temperature constant and A is a set of available transitions.

Now, we found the learning algorithm of delay time of LPN using the discretization method. And it is listed in Table 2.

Transition's delay time learning algorithm 1 (Discretization method):
Step 1. Initialization: discretize the delay time and set $Q(p,t)$ of every transition's delay time to zero.
Step 2. Initialize Petri net, i.e. make the Petri net state as P_1.
Repeat (i) and (ii) until system becomes end state.
i. Select a transition using formula (4).
ii. After transition fired and reward is observed, value of $Q(p,t)$ is adjusted using formula (3).
Step 3. Step 3. Repeat Step2 until t is optimal as required.

Table 2. Delay time learning algorithm using the discretization method

Function approximation method

First, the transition delay time is selected randomly and executed. The value of the delay time is obtained using formula (3). When the system is executed m times, the data $(t_i, Q_i(p,t_i))$ $(i = 1, 2, \ldots, m)$ is yielded. The relation of value of delay time Q and delay time t is supposed as $Q = F(t)$. Using the least squares method, $F(t)$ will be obtained as follows. It is supposed that F is a function class which is constituted by a polynomial. And it is supposed that formula (5) hold.

$$f(t) = \sum_{k=0}^{n} a_k t^k \in F \tag{5}$$

The data $(t_i, Q_i(p,t_i))$ are substituted in formula (5). Then:

$$f(t_i) = \sum_{k=0}^{n} a_k t_i^k \quad (i = 1, 2, \ldots, m\ ;\ m \geq n) \tag{6}$$

Here, the degree m of data $(t_i, Q_i(p,t_i))$ is not less than data number n of formula (5). According to the least squares method, we have (2.7).

$$||\delta||^2 = \sum_{i=1}^{m} \delta_i^2 = \sum_{i=1}^{m} [\sum_{k=0}^{n} a_k t_i^k - Q_i]^2 \Rightarrow \min \tag{7}$$

In fact, (7) is a problem which evaluates the minimum solution of function (8).

$$||\delta||^2 = \sum_{i=1}^{m} [\sum_{k=0}^{n} a_k t_i^k - Q_i]^2 \tag{8}$$

So, function (9), (10) are gotten from (8).

$$\frac{\partial ||\delta||^2}{\partial a_j} = 2\sum_{i=1}^{m}\sum_{k=0}^{n} (a_k t_i^k - Q_i) t_i^j = 0 \quad (j = 0, 1, \ldots, n) \tag{9}$$

$$\sum_{i=1}^{m}(\sum_{k=0}^{n} t_i^{j+k})a_k = \sum_{i=1}^{m} t_i^j Q_i \quad (j = 0, 1, \ldots, n). \tag{10}$$

Solution of Equation (10) $a_0, a_1, \ldots, a_n$ can be deduced and $Q = f(t)$ is attained. The solution t^*_{opt} of $Q = f(t)$ which makes maximum Q is the expected optimal delay time.

$$\frac{\partial f(t)}{\partial t} = 0 \tag{11}$$

The multi-solution of (11) $t = t_{opt}$ (opt = 1, 2, …, n-1) is checked by function (5) and a $t^*_{opt} \in t_{opt}$ which makes $f(t^*_{opt}) = \max f(t_{\text{opt}})$ (opt = 1, 2, …, n-1) is the expected optimal delay time. t^*_{opt} is used as delay time and the system is executed and new $Q(p, t^*_{opt})$ is gotten. This (t^*_{opt}, $Q(p, t^*_{opt})$) is used as the new and the least squares method can be used again to acquire more precise delay time.

After the values of actions are gotten, the soft-max method is selected as the actions selection policy. And then, we found the learning algorithm of delay time of Learning Petri net using the function approximation method. And it is listed in Table 3.

Transition's delay time learning algorithm 2 (Function approximation method):
Step 1. Step 1. Initialization: Set $Q(p, t)$ of every transition's delay time to zero.
Step 2. Step 2. Initialize Petri net, i.e. make the Petri net state as P_1.
Repeat (i) and (ii) until system becomes end state.
i. Randomly select the transition delay time t.
ii. After transition fires and reward is observed, the value of $Q(p, t)$ is adjusted using formula (3).
Step 3. Step 3. Repeat Step 2 until adequacy data are gotten. Then, evaluate the optimal t using the function approximation method.

Table 3. Delay time learning algorithm using the function approximation method

3. Applying LPN to robotic system control

3.1. Application for discrete event dynamic robotic system control

A discrete event dynamic system is a discrete-state, event-driven system in which the state evolution depends entirely on the occurrence of asynchronous discrete events over time [2]. Petri nets have been used to model various kinds of dynamic event-driven systems like computers networks, communication systems, and so on. In this Section, it is used to model Sony AIBO learning control system for the purpose of certification of the effectiveness of the proposed LPN.

AIBO voice command recognition system

AIBO (Artificial Intelligence roBOt) is a type of robotic pets designed and manufactured by Sony Co., Inc. 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 "listens" 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) [19]. The architecture of the simulation system is showed in Figure 3. 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 correctness of action and action lasted time are given by the touch of different AIBO's sensors.

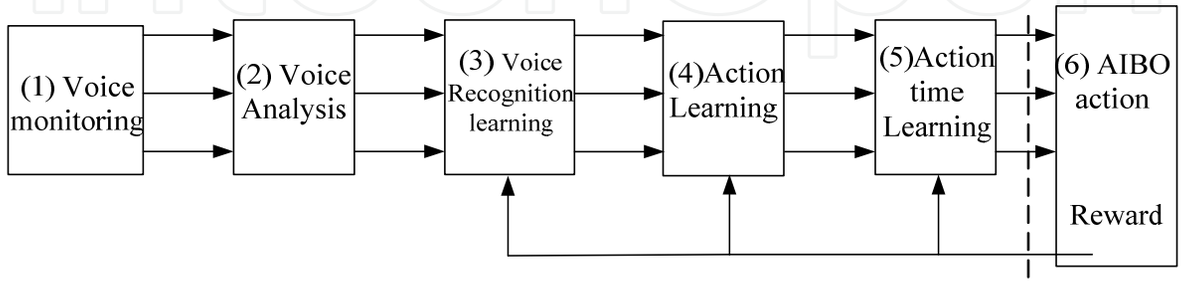


Figure 3. System architecture of voice command recognition

LPN model for AIBO voice command recognition system

In the LPN 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 the different color Token. The LPN model is showed in Figure 4. The meaning of every transition is listed below: Tr_{input} changes voice signal as colored Token which describe the voice characteristic. Tr_{11}, Tr_{12} and Tr_{13} can analyze the voice signal. Tr_1 generates 35 different Token $VL_1 \ldots VL_{35}$ according to the voice length. Tr_2 generates 8 different Token $E2_1 \ldots E2_8$ according to the front twenty voice sample energy characteristic. Tr_3 generates 8 different Token $E4_1 \ldots E4_8$ according to the front forty voice sample energy characteristic [8]. These three types of the token are compounded into a compound Token $<VL_l> + <VE2_m> + <VE4_n>$ in p_2 [12].

Tr_{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 Tr_{3j}. When $Pr_4 - Pr_8$ has Token, AIBO's action will last. Tr_{4j} takes Token out from $p_4 - p_8$, and makes corresponding AIBO action terminates. Tr_{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}.

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 Tr_{21} and colored Token

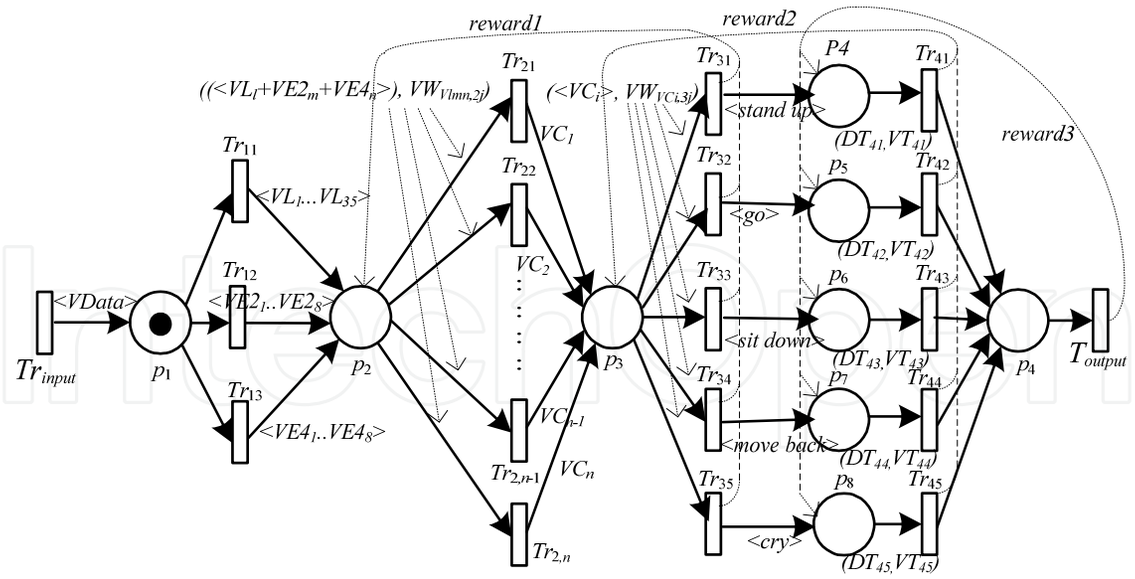


Figure 4. LPN model of voice command recognition

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 Tr_{21}. After an action finished, a reward for correctness of action time is gotten and VT is updated.

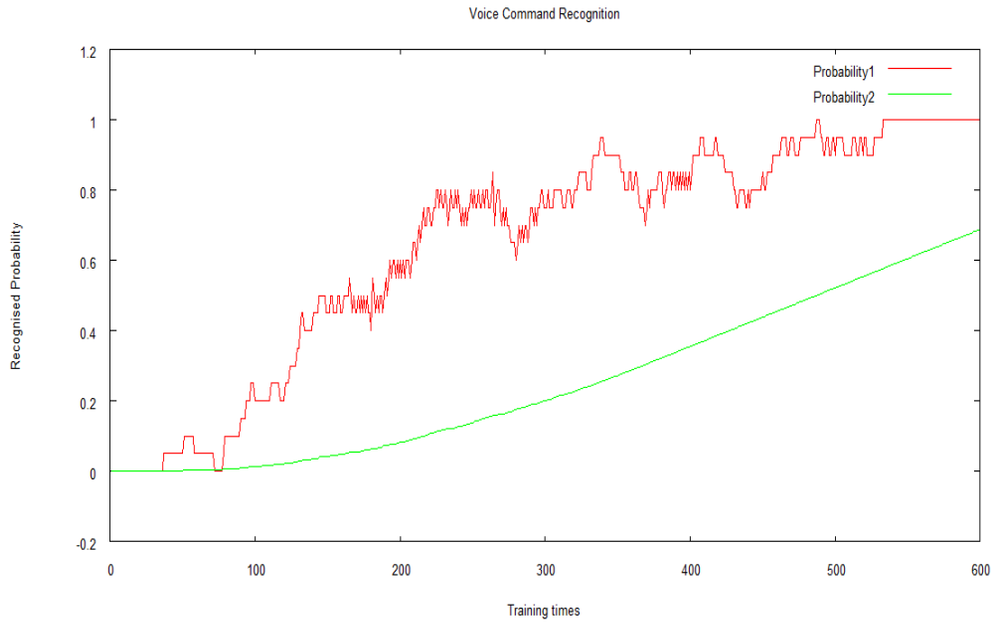


Figure 5. Relation between training times and recognition probability

Figure 5 shows the relation between 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 confirmed that LPN is correct and effective using the AIBO voice command control system.

3.2. Application for continuous parameter optimization

The proposed system is applied to guide dog robot system which uses RFID (Radio-frequency identification) to construct experiment environment. The RFID is used as navigation equipment for robot motion. The performance of the proposed system is evaluated through computer simulation and real robot experiment.

RFID environment construction

RFID tags are used to construct a blind road which showed in Figure 6. There are forthright roads, corners and traffic light signal areas. The forthright roads have two group tags which have two lines RFID tags. Every tag is stored with the information about the road. The guide dog robot moves, turns or stops on the road according to the information of tags. For example, if the guide dog robot reads corner RFID tag, then it will turn on the corner. If the guide dog robot reads either outer or inner side RFID tags, it implies that the robot will deviate from the path and robot motion direction needs adjusting. If the guide dog robot reads traffic control RFID tags, then it will stop or run unceasingly according to the traffic light signal which is dynamically written to RFID.

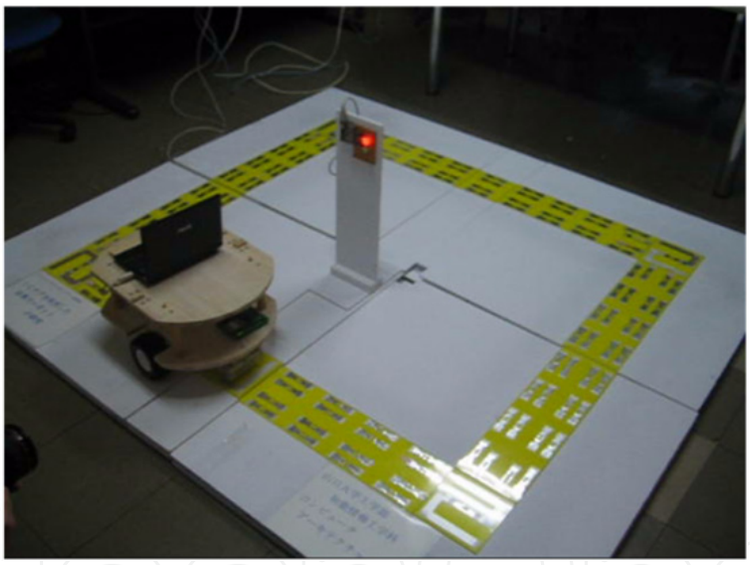


Figure 6. The real experimental environment

LPN model for the guide dog

The extended LPN control model for guide dog robot system is presented in Figure 7. The meaning of place and transition in Figure 7 is listed below:

P1	System starting state	P2	Getting RFID information
P3	Turning corner state	P4	Left adjusting state
P5	Right adjusting state	Tr1	Reading of the RFID environment
Tr2	Stop of the guide dog	Tr3	Guide dog runs
Tr4	Start of the turning corner state	Tr5	Start of left adjusting state
Tr6	Start of the right adjusting state	Tr7	Stop of the turning corner state
Tr8	Stop of the left adjusting state	Tr9	Stop of the right adjusting state

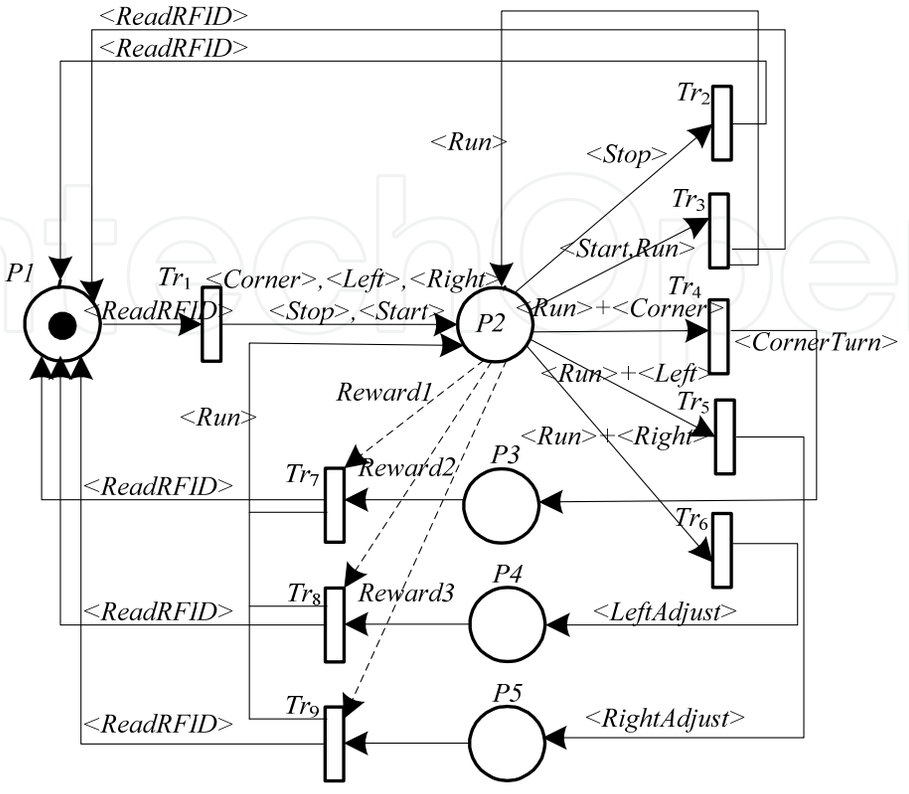


Figure 7. The LPN model for the guide dog robot

When the system begins running, it firstly reads RFID environment and gets the information, Token puts into P_2. These Tokens fire one of transition from Tr_2 to Tr_6 according to weight function on P_2 to Tr_2, …, Tr_6. Then, the guide dog enters stop, running, turning corner, left adjusting or right adjusting states. Here, at P_3, P_4, P_5 states, the guide dog turns at a specific speed. The delay time of Tr_7-Tr_9 decide the correction of guide dog adjusting its motion direction.

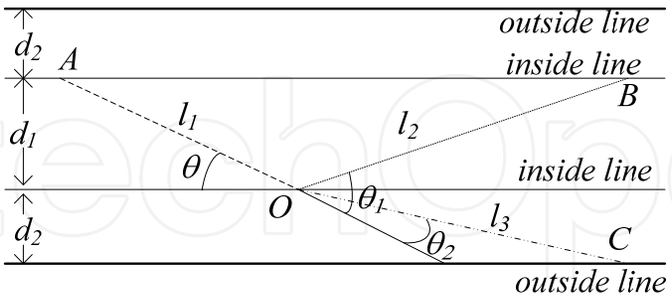
Reward getting from environment

When Tr_7, Tr_8 or Tr_9 fires, it will get reward r as formula (12-b) when the guide dog doesn't get Token <Left> and <Right> until getting Token <corner> i.e. the robot runs according correct direction until arriving corner. It will get reward r as formula (12-a), where t is time from transition fire to get Token <Left> and <Right>. On the contrary, it will get punishment -1 as (12-c) if robot runs out the road.

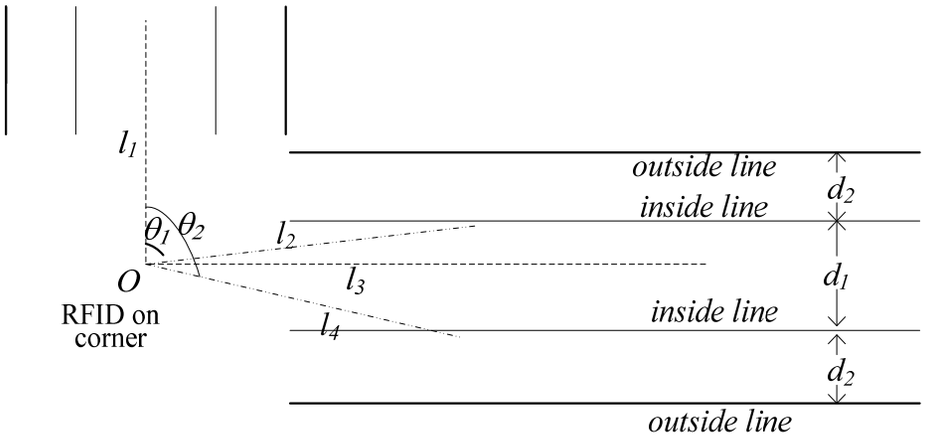
$$r = \begin{cases} 1/e^{t} & \text{(a)} \\ 1 & \text{(b)} \\ -1 & \text{(c)} \end{cases}. \tag{12}$$

Computer simulation and real robot experiment

When robot reads the <Left>, <Right> and <corner> information, it must adjust the direction of the motion. The amount of adjusting is decided by the continuing time of the robot at the state of P_3, P_4 and P_5. So, the delay time of Tr_7, Tr_8 and Tr_9 need to learn.



(i) Direction adjustment of the robot motion on the forthright road



(ii) Direction adjustment of the robot motion at the corner

Figure 8. Direction adjustment of the guide dog robot motion

Before the simulation, some robot motion parameter symbols are given as:

v velocity of the robot
ω angular velocity of the robot
t_{pre} continuous time of the former state
t adjusting time
t_{post} last time of the state after adjusting

v, ω, t_{pre}, t_{post} can be measured by system when the robot is running. The delay time of Tr_7, Tr_8 and Tr_9, i.e. the robot motion adjusting time, is simulated in two cases.

1. As shown in Figure 8 (i), when the robot is running on the forthright road and meets inside RFID line, its deviation angle θ is:

$$\theta = \arcsin(d_1/l_1) = \arcsin(d_1/(t_{pre} \bullet v)). \quad (13)$$

where d_1 and l_1 are width of area between two inside lines and moving distance between two times reading of the RFID, respectively (See Figure 8).

Robot's adjusting time (transition delay time) is t.

If $\omega t - \theta \geq 0$, then

$$t_{post} = \frac{d_1}{v \sin(\omega t - \theta)}, \tag{14}$$

else

$$t_{post} = \frac{d_2}{v \sin(\omega t - \theta)}. \tag{15}$$

Here, t_{post} is used to calculate reward r using formula (12). In the same way, the reward r can be calculated when the robot meets outside RFID line.

When the robot is running on the forthright road and meets the outside RFID line, the deviation angle θ is

$$\theta = \arcsin(d_2/(v \bullet t_{pre})), \tag{16}$$

Robot's adjusting time (transition delay time) is t.

If $\omega t - \theta \geq 0$, then

$$t_{post} = \frac{d_2}{v \sin(\omega t - \theta)}, \tag{17}$$

else the robot will runs out the road. And the reward r is calculated using formula (12).

2. As shown in Figure 8 (ii), when the robot is running at the corner, it must adjust θ=90°. If $\theta \neq 90°$, the robot will read <Left>, <Right> after it turns corner. Now, the case which the robot will read inner line <Left>, <Right> will be considered. If robot's adjusting time is t. If $\omega t - \theta \geq 0$, then

$$t_{post} = \frac{d_1}{2v \sin(\omega t - \theta)}, \tag{18}$$

else

$$t_{post} = \frac{d_2}{2v \sin(\omega t - \theta)} \tag{19}$$

Same to case (1), t_{post} is used to calculate reward r using formula (12). In the same way, the reward r can calculate when the robot meets outside RFID line. The calculation of reward, which is calculated from t, for other cases of direction adjustment of the robot is considered as the above two cases.

In this simulation, the value of the delay time has only a maximum at optimal delay time point. The graph of relation for the delay time and its value is parabola. So, when

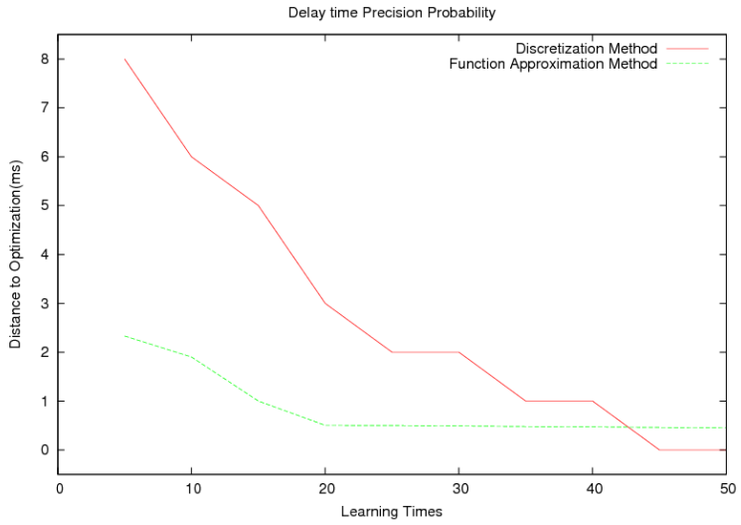
transition's delay time learning by function approximation method which states in section 2.2.3, the relation of the delay time and its value is assumed as:

$$Q = a_2 t^2 + a_1 t + a_0 . \tag{20}$$

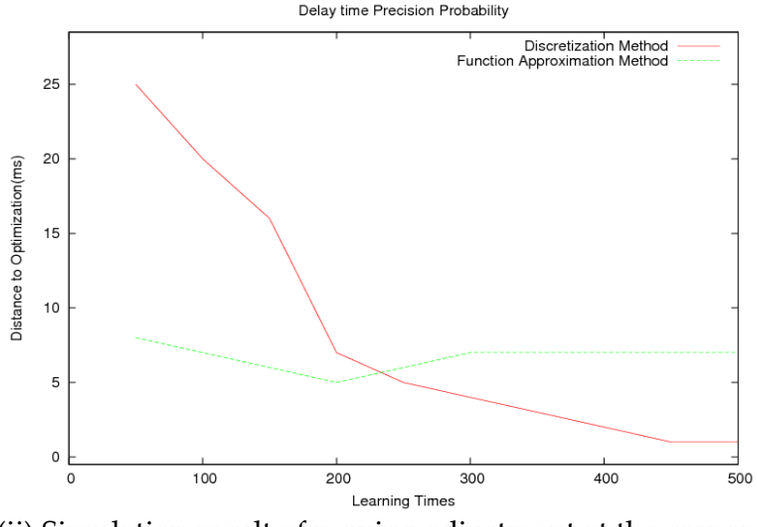
Computer simulations of Transition's delay time learning algorithms were executed in the all cases of the robot direction adjusting. In the simulation of algorithm of discretization, the positive inverse temperature constant β is set as 10.0. After the delay time of different cases was learnt, it is recorded in a delay time table. Then, the real robot experiment was carried out using the delay time table which was obtained by simulation process.

Result of simulation and experiment

The simulation result of transition's delay time learning algorithm in two cases is shown in Figure 9.



(i) Simulation result of moving adjustment on the forthright road



(ii) Simulation result of moving adjustment at the corner

Figure 9. Result of simulation for the guide dog robot

The simulation result of θ=5° for the robot moving adjustment on forthright road is shown in Figure 9 (i). The simulation result of robot moving adjustment at the corner is shown in Figure 9 (ii). From the result, it is found that the function approximation method can quickly approach optimal delay time than the discretization method, but the discretization method can approach more near optimal delay time through long time learning.

4. Construction of the learning fuzzy Petri net model

Petri net (PN) has ability to represent and analyze concurrency and synchronization phenomena in an easy way. PN approach can also be easily combined with other techniques and theories such as object-oriented programming, fuzzy theory, neural networks, etc. These modified PNs are widely used in the fields of manufacturing, robotics, knowledge based systems, process control, as well as other kinds of engineering applications [15]. Fuzzy Petri net (FPN), which combines PN and fuzzy theory, has been used for knowledge representation and reasoning in the presence of inexact data and knowledge based systems. But traditional FPN lacks of learning mechanism, it is the main weakness while modeling uncertain knowledge systems [25]. In this section, we propose a new learning model tool — learning fuzzy Petri net (LFPN) [7]. Contrasting with the existing FPN, there are three extensions in the new model: 1) the place can possess different tokens which represent different propositions; 2) these propositions have different degrees of truth toward different transitions; 3) the truth degree of proposition can be learned through the arc's weight function adjusting. The LFPN model obtains the capability of fuzzy production rules learning through truth degree updating. The artificial neural network is gotten learning ability through weight adjusting. The LFPN learning algorithm which introduces network learning method into Petri net update is proposed and the convergence of algorithm is analyzed.

4.1. The learning fuzzy Petri net model

Petri net is a directed, weighted, bipartite graph consisting of two kinds of nodes, called places and transitions, where arcs are either from a place to a transition or from a transition to a place. Tokens exist at different places. The use of the standard Petri net is inappropriate in situations where systems are difficult to be described precisely. Consequently, fuzzy Petri net is designed to deal with these situations where transitions, places, tokens or arcs are fuzzified.

The definition of fuzzy Petri net

A fuzzy place associates with a predicate or property. A token in the fuzzy place is characterized by a predicate or property belongs to the place, and this predicate or property has a level of belonging to the place. In this way, we may get a fuzzy proposition or conclusion, for example, *speed is low.* A fuzzy transition may correspond to an *if-then* fuzzy production rule for instance and is realized by truth values such as fuzzy inference algorithms [11, 20, 26].

Definition1 *FPN* is a 8-tuple, given by $FPN=<P, Tr, F, D, I, O, \alpha, \beta>$

where:

$P = \{p_1, p_2, \dots, p_n\}$ is a finite set of places;
$Tr = \{tr_1, tr_2, \dots, tr_m\}$ is a finite set of transitions;
$F \subseteq (P \times Tr) \cup (Tr \times P)$ is a finite set of directional arcs;
$D = \{d_1, d_2, \dots, d_n\}$ is a finite set of propositions, where proposition d_i corresponds to place p_i;
$P \cap Tr \cap D = \varnothing$; cardinality of ($P$) = cardinality of ($D$);
I: $tr \rightarrow P^{\infty}$ is the input function, representing a mapping from transitions to bags of (their input) places, noting as $*tr$;
O: $tr \rightarrow P^{\infty}$ is the output function, representing a mapping from transitions to bags of (their output) places, noting as $tr*$;
α: $P \rightarrow [0, 1]$ and β: $P \rightarrow D$. A token value in place $p_i \in P$ is denoted by $\alpha(p_i) \in [0, 1]$. If $\alpha(p_i)=y_i$, $y_i \in [0, 1]$ and $\beta(p_i)= d_i$,, then this states that the degree of truth of proposition d_i is y_i.

A transition tr_k is enabled if for all $p_i \in I(tr_k)$, $\alpha(p_i) \geq th$, where th is a threshold value in the unit interval. If this transition is fired, then tokens are moved from their input place and tokens are deposited to each of its output places. The truth values of the output tokens are $y_i \bullet u_k$, where u_k is the confidence level value of tr_k. FPN has capability of modeling fuzzy production rules. For example, the fuzzy production rule (21) can be modeled as shown in Figure 10.

IF d_i THEN d_j (with Certainty Factor (CF) u_k) (21)

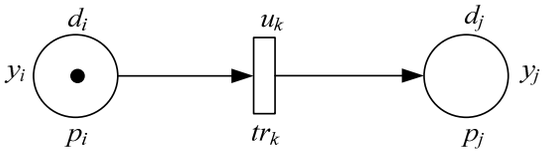


Figure 10. A fuzzy Petri net model (FPN)

The definition of LFPN

In a FPN, a token in a place represents a proposition and a proposition has a degree of truth. Now, three aspects of extension are done at the FPN and learning fuzzy Petri net (LFPN) is constructed. First, a place may have different tokens (Tokens are distinguished with numbers or colors) and the different tokens represent different propositions, i.e. a place has a set of propositions. Second, a place has a special token, i.e. there is a specified proposition. This proposition may have different degrees of truth toward different transitions tr which regard this place as input place $*tr$. Third, the weight of each arc is adjustable and used to record transition's input and output information.

Definition 3 *LFPN* is a 10-tuple, given by $LFPN= <P, Tr, F, D, I, O, Th, W, \alpha, \beta>$ (A LFPN model is shown in Figure 11).

where: Tr, F, I, O are same with definition of FPN.

P={ p_1, p_2,…, p_i,…, p_n,…, p'_1, p'_2,…, p'_i,…, p'_r} is a finite set of places, where p_i is input place and p'_i is output places.

D = {d_{11}, … , d_{1N} ; d_{21}, …, d_{2N} ; …, d_{ij}, …; d_{n1}, … , d_{nN} ; d'_{11}, … , d'_{1N} ; $d'_{21,}$ …, d'_{2N} ; …, d'_{ij}, …; d'_{r1}, … , d'_{rN}} is a finite set of propositions, where proposition d_{ij} is j-th proposition for input place p_i and proposition d'_{ij} is j-th proposition for output place p'_i.

W ={w_{11}, w_{12}, …, w_{1k}, …, w_{1m}; …; w_{i1}, w_{i2}, …, w_{ik}, …, w_{im}; … ; w_{n1}, w_{n2}, …, w_{nm}; w'_{11}, w'_{12}, …, w'_{1r}; …; w'_{k1}, w'_{k2}, …, w'_{kj} , …, w'_{kr}; … ; w'_{m1}, w'_{m2}, …, w'_{mr}} is the set of weights on the arcs, where w_{ik} is a weight from i-th input place to k-th transition and w'_{kj} is a weight from k-th transition to j-th output place.

W ={w_{11}, w_{12}, …, w_{1k}, …, w_{1m}; …; w_{i1}, w_{i2}, …, w_{ik}, …, w_{im}; … ; w_{n1}, w_{n2}, …, w_{nm}; w'_{11}, w'_{12}, …, w'_{1r}; …; w'_{k1}, w'_{k2}, …, w'_{kj} , …, w'_{kr}; … ; w'_{m1}, w'_{m2}, …, w'_{mr}} is the set of weights on the arcs, where w_{ik} is a weight from i-th input place to k-th transition and w'_{kj} is a weight from k-th transition to j-th output place.

$\alpha(d_{ij}, tr_k) \rightarrow [0, 1]$ and β: $P \rightarrow D$. When $p_i \in P$ has a special $token_{ij}$ and $\beta(token_{ij}, p_i)=d_{ij}$, the degree of truth of proposition d_{ij} in place p_i toward to transition tr_k is denoted by $\alpha(d_{ij}, tr_k) \in [0, 1]$. When tr_k fires, the probability of proposition d_{ij} in p_i is $\alpha(d_{ij}, tr_k)$.

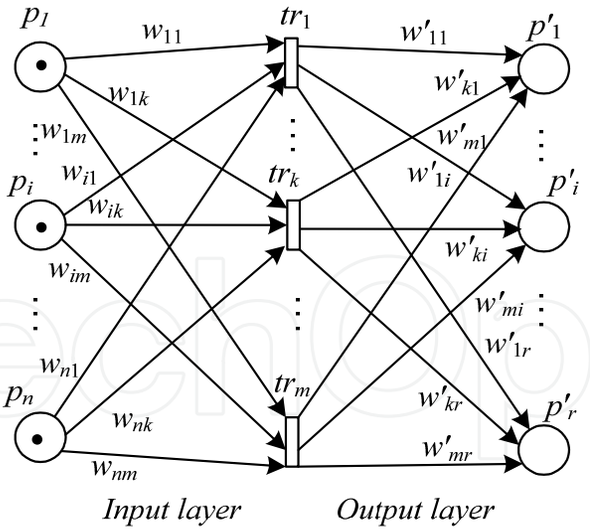


Figure 11. The model of learning fuzzy Petri net (LFPN)

$\alpha(d_{ij}, tr_k) \rightarrow [0, 1]$ and β: $P \rightarrow D$. When $p_i \in P$ has a special $token_{ij}$ and $\beta(token_{ij}, p_i)=d_{ij}$, the degree of truth of proposition d_{ij} in place p_i toward to transition tr_k is denoted by $\alpha(d_{ij}, tr_k) \in [0, 1]$. When tr_k fires, the probability of proposition d_{ij} in p_i is $\alpha(d_{ij}, tr_k)$.

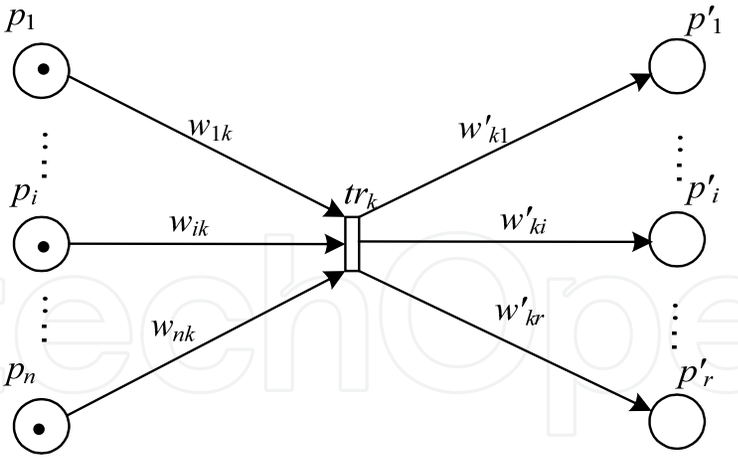


Figure 12. A LFPN model with one transition

$Th = \{th_1, th_2, \ldots, th_k, \ldots, th_m\}$ represents a set of threshold values in the interval [0, 1] associated with transitions (tr_1, tr_2, …, tr_k, …, tr_m), respectively; If all $p_i \in I(tr_k)$ and $\alpha(d_{ij}, tr_k) \geq th_k$, tr_k is enable.

As showed in Figure 12, when p_i has a $token_{ij}$, there is proposition d_{ij} in p_i. This proposition d_{ij} has different truth to tr_1, tr_2, …, tr_k, …, tr_m. When a transition tr_k fired, tokens are put into p'_1, …, p'_r according to weight w'_{k1}, …, w'_{kr} and each of p'_1, …, p'_r gets a proposition.

Figure 11 shows a LFPN which has n-input places, m-transitions and r-output places. To explain the truth computing, transition fire rule, token transfer rule and fuzzy production rules expression more clearly, a transition and its relation arcs, places are drawn-out from Figure 11 and shown in Figure 12.

Truth computing As shown in Figure 12, w_{ik} is the perfect value for $token_{ij}$ when tr_k fires. When a set of $tokens = (token_{1j}, token_{2j}, \ldots, token_{ij}, \ldots, token_{nj})$ are input to all places of $*tr_k$, $\beta(token_{1j}, p_1) = d_{1j}, \ldots, \beta(token_{nj}, p_n) = d_{nj}$. $\alpha(d_{ij}, tr_k)$ is computed using the degree of similarity between $token_{ij}$ and w_{ik} and calculation formula is shown in formula (22).

$$\alpha(d_{ij}, tr_k) = 1 - \frac{\left| w_{ik} - token_{ij} \right|}{\max(\left| w_{ik} \right|, \left| token_{ij} \right|)} \tag{22}$$

According to LFPN models for different systems, the token and weight value may have different data types. There are different methods for computing $\alpha(d_{ij}, tr_k)$ according to data type. If value types of token and weight are real number, $\alpha(d_{ij}, tr_k)$ is computed as formula (2). In Section 4, $\alpha(d_{ij}, tr_k)$ will be discussed for a LFPN model which has the textual type token and weight.

Transition fire rule As shown in Figure 12, when a set of $tokens = (token_{1j}, token_{2j}, \ldots, token_{nj})$ are input to all places of $*tr_k$, and $\beta(token_{1j}, p_1) = d_{1j}, \ldots, \beta(token_{nj}, p_n) = d_{nj}$. If all $\alpha(d_{ij}, tr_k)$ (i=1, 2, …, n) $\geq th_k$ is held, tr_k is enabled. Maybe, several transitions are enabled at same time. If formula (23) is held, tr_k is fired.

$$\begin{aligned}&\alpha(d_{1j},tr_k)\cdot\alpha(d_{2j},tr_k)\cdot\ldots\cdot\alpha(d_{nj},tr_k)\\&=\max(\alpha(d_{1j},tr_h)\cdot\alpha(d_{2j},tr_h)\cdot\ldots\cdot\alpha(d_{nj},tr_h)_{1\le h\le m})\end{aligned}\tag{23}$$

Token transfer rule As shown in Figure 12, after tr_k fired, token will be taken out from p_1~p_n. The token take rule is:

If $token_{ij} \le w_{ik}$ is held, $token_{ij}$ in p_i will be taken out.
If $token_{ij} \ge w_{ik}$ is held, *token* which equates $token_{ij}-w_{ik}$ will be left in p_i.

Thus, after a transition tr_k fired, maybe the enable transitions still exist in LFPN. An enable transition will be selected and fired according to formula (23) until there isn't any enable transition.

After tr_k fired, the token according w'_{ki} will be put into p'_i. For example, if the weight function of arc tr_k to p'_i is w'_{ki}, then *token* which equates w'_{ki} will be put into p'_i.

Fuzzy production rules expression A LFPN is capable of modeling for fuzzy production rules just as a FPN. For example, as a case which states in **Transition fire rule** and **Token transfer rule**, when tr_k is fired, the below production rule is expressed:

IF d_{1j} AND d_{2j} AND … AND d_{nj} THEN d'_{1k} AND d'_{2k} AND … AND d'_{rk}

$$(\text{CF}=\alpha\ (d_{1i},\ tr_k)\bullet\alpha(d_{2i},\ tr_k)\bullet\ldots\bullet\alpha(d_{ni},\ tr_k)) \tag{24}$$

The mathematical model of LFPN

In this section, the mathematical model of LFPN will be elaborated. Firstly, some conceptions are defined. When a $token_{ij}$ is input to a place p_i, it is defined event p_{ij} occurs, i.e. the proposition d_{ij} is generated and probability of event p_{ij} is $Pr(p_{ij})$. The fired tr_k is defined as event tr_k and probability of event tr_k occurrence is $Pr(tr_k)$. Secondly, we assume that each transition tr_1, tr_2, …, tr_k, …, tr_m has the same fire probability in whole event space, then

$$Pr(tr_k)=\frac{1}{m} \tag{25}$$

And when event tr_k occurs, the conditional probability of p_{ij} occurrence is defined as $Pr(p_{ij} \mid tr_k)$, i.e. $\acute{a}(d_{ij},\ tr_k)$ which is the probability of proposition d_{ij} generation when tr_k fires.

When p_1, p_2 , …, p_n have $token_{1j}$, $token_{2j}$…$token_{nj}$ and events p_{1j} , p_{2j} , …, p_{nj} occur. Then, $Pr(tr_k \mid p_{1j}, p_{2j}, \ldots, p_{nj})$ is:

$$Pr(tr_k \mid p_{1j},p_{2j},\ldots,p_{nj})=\frac{Pr(p_{1j},p_{2j},\ldots,p_{nj} \mid tr_k)Pr(tr_k)}{\sum_{h=1}^{m}Pr(tr_h)Pr(p_{1j}p_{2j},\ldots,p_{nj})} \tag{26}$$

When events p_{1j}, p_{2j}, …, p_{nj} occurred, there is one of transitions tr_1, tr_2, …, tr_k, …, tr_m which will be fired, therefore

$$\sum_{h=1}^{m} Pr(tr_h) Pr(p_{1j}, p_{2j}, ..., p_{nj}) = 1 \tag{27}$$

From (25), (26) and (27), (28′) is gotten by the formula of full probability and Bayesian formula.

$$Pr(tr_k \mid p_{1j}, p_{2j}, ..., p_{nj}) = \frac{1}{m} Pr(p_{1j}, p_{2j}, ..., p_{nj} \mid tr_k)$$

$$= \frac{1}{m} Pr(p_{1j} \mid tr_k) \times Pr(p_{2j} \mid tr_k) \times ... \times Pr(p_{nj} \mid tr_k) \tag{28′}$$

$$= \frac{1}{m} \alpha(d_{1j}, tr_k) \cdot \alpha(d_{2j}, tr_k) \cdot ... \cdot \alpha(d_{nj}, tr_k) \tag{28}$$

The transformation from (28′) to (28) is according to definition of $\alpha(d_{ij}, tr_k)$. As shown in Figure 11, when p_1, p_2 , …, p_n have $token_{1j}$, $token_{2j}$…$token_{nj}$, the occurring probability of transition tr_1, …, tr_k, …, tr_m are $\alpha(d_{1j}, tr_1) \bullet \alpha(d_{2j}, tr_1) \bullet ... \bullet \alpha(d_{nj}, tr_1)/m$, …, $\alpha(d_{1j}, tr_k) \bullet \alpha(d_{2j}, tr_k) \bullet ... \bullet \alpha(d_{nj}, tr_k)/m$, …, $\alpha(d_{1j}, tr_m) \bullet \alpha(d_{2j}, tr_m) \bullet ... \bullet \alpha(d_{nj}, tr_m)/m$. Thus, the transition tr_k, which has maximum of $\alpha(d_{1j}, tr_k) \bullet \alpha(d_{2j}, tr_k) \bullet ... \bullet \alpha(d_{nj}, tr_k)$, is selected and fired according to formula (23).

4.2. Learning algorithm for learning fuzzy Petri net

Learning algorithm

The learning fuzzy Petri net (LFPN) can be trained and made it learn fuzzy production rules. When a set of data input LFPN, a set of propositions are produced in each input place. For example, when token vectors ($token_{1j}$, $token_{2j}$, …, $token_{nj}$) (j=1, 2, …, N) input to p_1~p_n, propositions d_{1j}, d_{2j}, …, d_{nj} (j=1, 2, …, N) are produced. To train a fuzzy production rule which is IF d_{1j} AND d_{2j} AND … AND d_{nj} THEN d'_{1k} AND d'_{2k} AND … AND d'_{nk}, there are two tasks:

1. $\alpha(d_{1j}, tr_k) \bullet \alpha(d_{2j}, tr_k) \bullet ... \bullet \alpha(d_{nj}, tr_k)$ ($k \in \{1,2,...m\}$) need to be updated to hold formula (23);
2. 2) The output weight function of tr_k need to be updated for putting correct token to p'_1~p'_r. Then, $\beta(p'_1) = d'_{1k}$, $\beta(p'_2) = d'_{2k}$, …, $\beta(p'_r) = d'_{rk}$.

To accomplish these two tasks, the weights w_{1k}, w_{2k}, …, w_{nk} and w'_{k1}, w'_{k2}, …, w'_{kr} are modified by a learning algorithm of LFPN. Firstly, we define the training data set as {($\boldsymbol{X}_1$, $\boldsymbol{Y}_1$), ($\boldsymbol{X}_2$, $\boldsymbol{Y}_2$), …, ($\boldsymbol{X}_N$, $\boldsymbol{Y}_N$)}, where $\boldsymbol{X}$ is input token vector, $\boldsymbol{Y}$ is output token vector and $\boldsymbol{X}_j$, $\boldsymbol{Y}_j$ is defined as $\boldsymbol{X}_j$=(x_{1j}, x_{2j}, …, x_{nj})^T, $\boldsymbol{Y}_j$=(y_{1j}, y_{2j}, …, y_{rj})^T, respectively. Thus,

$\boldsymbol{X}$=($\boldsymbol{X}_1$, $\boldsymbol{X}_2$, …, $\boldsymbol{X}_j$, …, $\boldsymbol{X}_N$,), $\boldsymbol{Y}$=($\boldsymbol{Y}_1$, $\boldsymbol{Y}_2$, …, $\boldsymbol{Y}_j$, …, $\boldsymbol{Y}_N$) , i.e.

$$\boldsymbol{X}=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1j} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2j} & \cdots & x_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nj} & \cdots & x_{nN} \end{bmatrix} \quad \boldsymbol{Y}=\begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1j} & \cdots & y_{1N} \\ y_{21} & y_{22} & \cdots & y_{2j} & \cdots & y_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{r1} & y_{r2} & \cdots & y_{rj} & \cdots & y_{rN} \end{bmatrix}$$

Secondly, the weight $\boldsymbol{W}_k=(w_{1k}, w_{2k}, \ldots, w_{nk})^T$ is the weight on arcs from $*tr_k$ to tr_k and $\boldsymbol{W}'_k=(w'_{k1}, w'_{k2}, \ldots, w'_{kr})^T$ is the weight on arcs from tr_k to tr_k*. $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_k, \ldots, \boldsymbol{W}_m$ and $\boldsymbol{W}'_1, \ldots, \boldsymbol{W}'_k, \ldots, \boldsymbol{W}'_m$ are the input and output arcs weight for $tr_1, \ldots, tr_k, \ldots, tr_m$. Thus,

$\boldsymbol{W}=(\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_k, \ldots, \boldsymbol{W}_m)$, $\boldsymbol{W}'=(\boldsymbol{W}'_1, \boldsymbol{W}'_2, \ldots, \boldsymbol{W}'_k, \ldots, \boldsymbol{W}'_m)$, i.e.

$$\boldsymbol{W}=\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2k} & \cdots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nk} & \cdots & w_{nm} \end{bmatrix} \quad \boldsymbol{W}'=\begin{bmatrix} w'_{11} & w'_{21} & \cdots & w'_{k1} & \cdots & w'_{m1} \\ w'_{12} & w'_{22} & \cdots & w'_{k2} & \cdots & w'_{m2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w'_{1r} & w'_{2r} & \cdots & w'_{kr} & \cdots & w'_{mr} \end{bmatrix}$$

Lastly, in the learning algorithm, when tr_k is fired, the truth of $d'_{1j}, d'_{2j}, \ldots, d'_{rj}$ to tr_k are defined as $\alpha(d'_{1j}, tr_k)=1-|y_{1j}-w'_{k1}|/\max(|w'_{k1}|, |y_{1j}|)$, $\alpha(d'_{2j}, tr_k)=1-|y_{2j}-w'_{k2}|/\max(|w'_{k2}|, |y_{2j}|)$, …, $\alpha(d'_{rj}, tr_k)=1-|y_{rj}-w'_{kr}|/\max(|w'_{kr}|, |y_{rj}|)$ according to definition 3.The learning algorithm of learning fuzzy Petri net is shown in Table 4.

Learning Algorithm of LFPN:

Step 1. $\boldsymbol{W}$ and $\boldsymbol{W}'$ are selected randomly.

Step 2. For every training data set $(\boldsymbol{X}_j, \boldsymbol{Y}_j)(j=1, 2, \ldots, N)$, subject propositions $d_{1j}, d_{2j}, \ldots, d_{nj}$ in $p_1 \sim p_n$ and propositions $d'_{1j}, d'_{2j}, \ldots, d'_{rj}$ in $p'_1 \sim p'_r$ are produced. Then do step 3 to step 7;

Step 3. For $i=1$ to n

For $h=1$ to m do

Compute $\alpha(d_{ij}, tr_h)$ according to formula (2);

Step 4. Compute maximum truth of transition

4.1 $Max=\alpha(d_{1j}, tr_1) \bullet \alpha(d_{2j}, tr_1) \bullet \ldots \bullet \alpha(d_{nj}, tr_1)$; $k=1$;

4.2 For $h=1$ to m do

If $\alpha(d_{1j}, tr_h) \bullet \alpha(d_{2j}, tr_h) \bullet \ldots \bullet \alpha(d_{nj}, tr_h) > Max$

Then { $Max=\alpha(d_{1j}, tr_h) \bullet \alpha(d_{2j}, tr_h) \bullet \ldots \bullet \alpha(d_{nj}, tr_h)$;

$k=h$; }

Step 5. Fire tr_k;

Step 6. Make $d_{1j}, d_{2j}, \ldots, d_{nj}$ have bigger truth to tr_k,

$$\boldsymbol{W}_k^{(new)} = \boldsymbol{W}_k^{(old)} + \gamma(\boldsymbol{X}_j - \boldsymbol{W}_k^{(old)}) \quad (29)$$

($\boldsymbol{W}_k^{(new)}$ is the vector $\boldsymbol{W}_k$ after update and $\boldsymbol{W}_k^{(old)}$ is the vector $\boldsymbol{W}_k$ before updated. $\gamma \in (0,1)$ is learning rate.)

Step 7. Make $d'_{1j}, d'_{2j}, \ldots, d'_{rj}$ have bigger truth to tr_k,

$$\boldsymbol{W}'^{(new)}_k = \boldsymbol{W}'^{(old)}_k + \gamma(\boldsymbol{Y}_j - \boldsymbol{W}'^{(old)}_k) \quad (30)$$

($\boldsymbol{W}'^{(new)}_k$ is the vector $\boldsymbol{W}'_k$ after update and $\boldsymbol{W}'^{(old)}_k$ is the vector $\boldsymbol{W}'_k$ before updated. $\gamma \in (0,1)$ is learning rate.)

Step 8. Repeat step 2-7, until the truth of $\alpha(d_{1j}, tr_k)$, $\alpha(d_{2j}, tr_k)$, …, $\alpha(d_{nj}, tr_k)$ meet the requirement.

Table 4. Learning algorithm of learning fuzzy Petri net

Some details in the algorithm need to be elaborated further.

1. About the net construction: The number of input and output places can be easily set according to a real problem. It is difficult to decide a number of transitions when the net is initialized. When LFPN is used to solve a special issue, the number of transitions is initially set according to practical situation experientially. Then, transitions can be dynamically appended and deleted during the training. If an input data $\boldsymbol{X}_j$ has a maximal truth to tr_k but one or several $\alpha(d_{ij}, tr_k)(1 \le i \le n)$ are less than th_k (threshold of tr_k), transition tr_k cannot fire according to definition 3. Thus, data $\boldsymbol{X}_j$ cannot fire any existed transition. This case means that $\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_k, \ldots, \boldsymbol{W}_m$ cannot describe the vector characteristic of $\boldsymbol{X}_j$. Then, a new transition tr_{m+1} and the arcs which connect tr_{m+1} with input and output place are constructed. $\boldsymbol{X}_j$ can be set as weight $\boldsymbol{W}_{m+1}$ directly. Second, during a training episode, if there is no data in $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N$ that can fire transition tr_d, it means that $\boldsymbol{W}_d$ cannot describe the vector characteristic of any data $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N$. Then, the transition tr_d and the arcs which connect tr_d with input and output place will be deleted.
2. About $\boldsymbol{W}$ and $\boldsymbol{W}'$ initialization: for promoting training efficiency at the first stage of training, $\boldsymbol{W}$ and $\boldsymbol{W}'$ are set randomly in $[\boldsymbol{X}_{\min}, \boldsymbol{X}_{\max}]$, $[\boldsymbol{Y}_{\min}, \boldsymbol{Y}_{\max}]$ ($\boldsymbol{X}_{\min}$ is a vector which every components is minimal component of vector set $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N$; $\boldsymbol{X}_{\max}$ is a vector which every components is maximal component of vector set $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N$; $\boldsymbol{Y}_{\min}$, $\boldsymbol{Y}_{\max}$ are same meaning with $\boldsymbol{X}_{\min}$, $\boldsymbol{X}_{\max}$).
3. Training stop condition of the learning algorithm: According to application case, $th_1, th_2, \ldots, th_k, \ldots th_m$ are generally set a same value th. When training begins, the threshold th is set low (for example 0.2), th increases as training time increasing. A threshold value th_{last} (for example 0.9) is set as training stop condition and algorithm is run until $\alpha(d_{1j}, tr_k) > th_{last}$, $\alpha(d_{2j}, tr_k) > th_{last}$ $\ldots \alpha(d_{nj}, tr_k) > th_{last}$. From transition appending analysis, we understand that number of transitions will near to the number of training data if the threshold of transition sets near to 1. In this case, results will be obtained more correctly but the training time and LFPN running time will increase.

Analysis for convergence of LFPN learning algorithm

In this section, the convergence of the proposed algorithm will be analyzed. In step 6 of the LFPN learning algorithm, the formula (29) is used for making Wk (new) approach Xj than Wk (old) when Xj fired a transition trk. It is proved as follows.

$$\begin{aligned} W_k^{(\text{new})} &= W_k^{(\text{old})} + \gamma(X_j - W_k^{(\text{old})}) = W_k^{(\text{old})} + \gamma \mathbf{X}_j - \gamma \mathbf{W}_k^{(\text{old})} \\ X_j - W_k^{(\text{new})} &= X_j - [W_k^{(\text{old})} + \gamma(X_j - W_k^{(\text{old})})] \\ &= X_j - W_k^{(\text{old})} - \gamma \mathbf{X}_j + \gamma \mathbf{W}_k^{(\text{old})} \\ &= (1-\gamma)\,(X_j - W_k^{(\text{old})}) \end{aligned} \tag{31'}$$

Formula (11′) is rewritten as a scalar type and the scalar type of $(\boldsymbol{X}_j - \boldsymbol{W}_k^{(\text{old})})$ is used to divide both sides of formula (11′). We get formula (11).

$$\left(\frac{\boldsymbol{x}_{ij}-\boldsymbol{w}_{kj}^{(new)}}{\boldsymbol{x}_{ij}-\boldsymbol{w}_{kj}^{(old)}}\right)_{0\le i\le n}=\left(\frac{\left(1-\gamma\right)\cdot\left(\boldsymbol{x}_{ij}-\boldsymbol{w}_{kj}^{(old)}\right)}{\boldsymbol{x}_{ij}-w_{kj}^{(old)}}\right)_{0\le i\le n}=1-\gamma \tag{31}$$

Hence, $\boldsymbol{W}_k$ will converge to $\boldsymbol{X}_j$ after enough training times.

In LFPN learning algorithm, there may be a class of training data $\boldsymbol{X}_j$ which are able to fire same transition tr_k. In this case, $\boldsymbol{W}_k$ approaches to a class of data $\boldsymbol{X}_j$ and converges to a point in the class of data $\boldsymbol{X}_j$ according to formula (31).

Now, we will discuss the point in the class of data $\boldsymbol{X}_j$ where $\boldsymbol{W}_k$ converges to. Supposing, there are b_1 data which are in $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, …, $\boldsymbol{X}_j$, …, $\boldsymbol{X}_N$ and fire a certain transition tr_k at the first training episode. At the second training episode, there are b_2 data which fire tr_k, and so on. If the total training times is ep and the total number of data which fire tr_k is t, $t=\sum_{i=1}^{ep}b_i$. According to the order of the data fired tr_k, these t data are rewritten as $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{kt}$. The average of training data $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{kt}$ is noted as $\overline{\boldsymbol{X}}_k$. To record the updated process of $\boldsymbol{W}_k$ simply, the updated order of $\boldsymbol{W}_k$ is recorded as $\boldsymbol{W}_{k1}$,$\boldsymbol{W}_{k2}$…. $\boldsymbol{W}_{kt}$.

The learning rate γ ($0<\gamma<1$) will decrease according to training time increasing, and it approaches to 0 at last because every training data cannot effect $\boldsymbol{W}_k$ too much in the last stage of training, else $\boldsymbol{W}_k$ will shake at the last stage of training. If learning rate γ is set as $1/(q+1)$ ($q>0$) when training begin, $1/(q+2)$, $1/(q+3)$, …, $1/(q+t)$ are set as learning rate γ when tr_k is fired at 2, 3, …, t time. Here, the initial values of $\boldsymbol{W}_k$ is set as $\boldsymbol{W}_{k0}=\boldsymbol{W}^{(0)}\times 1/q$, every component of $\boldsymbol{W}^{(0)}\times 1/q$ is a random value in $[\boldsymbol{X}_{\min}, \boldsymbol{X}_{\max}]$. According to formula (29), we get

$$\boldsymbol{W}_{k0}=\frac{1}{q}\boldsymbol{W}^{(0)}$$

$$\boldsymbol{W}_{k1}=\boldsymbol{W}_{k0}+\frac{1}{q+1}(\boldsymbol{X}_{k1}-\boldsymbol{W}_{k0})=\frac{1}{q}\boldsymbol{W}^{(0)}-\frac{1}{q+1}\times\frac{1}{q}\boldsymbol{W}^{(0)}+\frac{1}{q+1}\boldsymbol{X}_{k1}=\frac{1}{q+1}(\boldsymbol{W}^{(0)}+\boldsymbol{X}_{k1})$$

$$\boldsymbol{W}_{k2}=\boldsymbol{W}_{k1}+\frac{1}{q+2}(\boldsymbol{X}_{k2}-\boldsymbol{W}_{k1})=\frac{1}{q+1}\boldsymbol{W}^{(0)}+\frac{1}{q+1}\boldsymbol{X}_{k1}$$

$$-\frac{1}{q+1}\times\frac{1}{q+2}\boldsymbol{W}^{(0)}-\frac{1}{q+1}\times\frac{1}{q+2}\boldsymbol{X}_{k1}+\frac{1}{q+2}\boldsymbol{X}_{k2}$$

$$=\frac{1}{q+2}(\boldsymbol{W}^{(0)}+\boldsymbol{X}_{k1}+\boldsymbol{X}_{k2})$$

$$\boldsymbol{W}_{kt}=\boldsymbol{W}_{k,t-1}+\frac{1}{q+t}(\boldsymbol{X}_{kt}-\boldsymbol{W}_{k,t-1})=\frac{1}{q+t-1}\boldsymbol{W}^{(0)}-\frac{1}{q+t}\times\frac{1}{q+t-1}\boldsymbol{W}^{(0)}+$$

$$\frac{1}{q+t-1}\boldsymbol{X}_{k1}-\frac{1}{q+t}\times\frac{1}{q+t-1}\boldsymbol{X}_{k1}+\ldots+\frac{1}{q+t-1}\boldsymbol{X}_{k,t-1}-\frac{1}{q+t}\times\frac{1}{q+t-1}\boldsymbol{X}_{k,t-1}+\frac{1}{q+t}\boldsymbol{X}_{kt}$$

$$= \frac{1}{q+t}(\boldsymbol{W}^{(0)} + \boldsymbol{X}_{k1} + ... + \boldsymbol{X}_{k,t-1} + \boldsymbol{X}_{kt}) \tag{32}$$

When the training time increases, the training data set $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{kt}$ can be looked as very large, i.e. t is large.

$$\lim_{t\to\infty} \boldsymbol{W}_{kt} = \lim_{t\to\infty} \frac{1}{q+t}(\boldsymbol{W}^{(0)} + \boldsymbol{X}_{k1} + ... + \boldsymbol{X}_{k,t-1} + \boldsymbol{X}_{kt}) \tag{33}$$

Generally, q is a small positive constant and t is large. Then,

$$\lim_{t\to\infty} \boldsymbol{W}_{kt} \approx \lim_{t\to\infty} \frac{1}{t}(\boldsymbol{W}^{(0)} + \boldsymbol{X}_{k1} + ... + \boldsymbol{X}_{k,t-1} + \boldsymbol{X}_{kt})$$

$$= \lim_{t\to\infty} \frac{1}{t}\boldsymbol{W}^{(0)} + \lim_{t\to\infty} \frac{1}{t}(\boldsymbol{X}_{k1} + ... + \boldsymbol{X}_{k,t-1} + \boldsymbol{X}_{kt})$$

$$\approx \lim_{t\to\infty} \frac{1}{t}(\boldsymbol{X}_{k1} + ... + \boldsymbol{X}_{k,t-1} + \boldsymbol{X}_{kt}) = \overline{\boldsymbol{X}_k} \tag{34}$$

From formula (14) will be gotten:

$$\boldsymbol{W}_k \rightarrow \overline{\boldsymbol{X}}_k \tag{35}$$

In the same way, $\boldsymbol{W}_k \rightarrow \overline{\boldsymbol{X}}_k$ (k=1, 2, …, m) and $\boldsymbol{W'}_k \rightarrow \overline{\boldsymbol{Y}}_k$ (k=1, 2, …, m) can be proved. Consequently, the learning algorithm of LFPN converges.

Now, we will analyze the convergence process and signification of convergence.

1. $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{kt}$ fire a certain transition tr_k at training time. As the training time increase, there are almost same data which fire the transition tr_k in every training time. These data belong to a class k. We suppose that these data are $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$. When training begins, supposing, there is data $\boldsymbol{X}_u$ which does not belong to $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$ but fires tr_k. But, when training times increase, $\boldsymbol{W}_k$ will approach to $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$ and the probability which $\boldsymbol{X}_u$ fires tr_k will decrease. Hence, this type data $\boldsymbol{X}_u$ is very small part of $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{kt}$. $\boldsymbol{X}_u$ little affects to $\boldsymbol{W}_k$. On the other hand, when training begins, there is $\boldsymbol{X}_{ke}$ which belongs to $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$ but doesn't fire transition tr_k. But, when training times increase, the probability which $\boldsymbol{X}_{ke}$ fires tr_k increases, then, $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$ can be approximately looked firing tr_k according to the training. $\overline{\boldsymbol{X}}_k$ is denoted as the average of training data $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$.
2. In the convergence demonstration, we use a special series of learning rate γ. Form the analysis in 1), $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$ can be looked as a class data which fires one transition tr_k. The data series $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{kt}$ can be looked as iterations of $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$. $\boldsymbol{W}_k$ can converge to a point near $\overline{\boldsymbol{X}}_k$ with any damping learning rate series γ.
3. After training, $\boldsymbol{W}_k$=(w_{1k}, w_{2k}, …, w_{nk}) comes near to the average of data which belong to class k, i.e., $\boldsymbol{W}_k \approx \overline{\boldsymbol{X}}_k$ =($\overline{x}_{1k}$, $\overline{x}_{2k}$, …, $\overline{x}_{nk}$). When a data $\boldsymbol{X}_{kj}$ belong to class k comes, $\boldsymbol{X}_{kj}$ will

have same vector characteristic with $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$, i.e. $x_{1,kj}$, $x_{2,kj}$, …, $x_{n,kj}$ are near to w_{1k}, w_{2k}, …, w_{nk}. Then, each component $x_{i,\,kj}$ ($1 \le i \le n$) of this data $\boldsymbol{X}_{kj}$ will have bigger similarity to w_{ik} ($1 \le i \le n$) than i-th components of other weight $\boldsymbol{W}$ according to formula (2). $\boldsymbol{X}_{kj}$ will have biggest truth to tr_k according to formula (2). Thus, when data $\boldsymbol{X}_{kj}$ which belongs to class of $\boldsymbol{X}_{k1}$, $\boldsymbol{X}_{k2}$, …, $\boldsymbol{X}_{ks}$ inputs to LFPN, it will fire tr_k correctly and product correct output.

5. Web service discovery based on learning fuzzy Petri net model

Web services are used for developing and integrating highly distributed and heterogeneous systems in various domains. They are described by Web Services Description Language (WSDL). Web services discovery is a key to dynamically locating desired Web services across the Internet [16]. It immediately raises an issue, i.e. to evaluate the accuracy of the mapping in a heterogeneous environment when user wants to invoke a service. There are two aspects which need to evaluate. One is functional evaluation. The service providing function should be completely matched with user's request; another aspect is non-functional evaluation, i.e. Quality of Service (QoS) meets user's requirement. UDDI (Universal Description, Discovery and Integration) is widely used as a kind of discovery approach for functional evaluation. But, as the number of published Web services increases, discovering proper services using the limited description provided by the UDDI standard becomes difficult [17]. And UDDI cannot provide the QoS information of service. To discover the most appropriate service, there are necessary to focus on developing feasible discovery mechanisms from different service description methods and service execution context. Segev proposed a service function selection method [21]. A two-step, context based semantic approach to the problem of matching and ranking Web services for possible service composition is elaborated. The two steps for service function selection are Context extraction and Evaluation for Proximity degree of Service. Cai proposed service performance selection method [3]. The authors used a novel Artificial Neural Network-based service selection algorithm according to the information of the cooperation between the devices and the context information. In this paper, we aim at analyzing different context of services and constructing a services discovery model based on the LFPN. Firstly, different service functional descriptions are used to evaluate service function and an appropriate service is selected. Secondly, context of QoS is used to predict QoS and a more efficient service is selected. Data of QoS is real number and LFPN learning algorithm is directly used. But service function description is literal. Therefore, a Learning Fuzzy Petri Net for service discovery model is proposed for keyword learning based on LFPN.

5.1. Web services discovery model based on LFPN

To map a service's function accurately, free textual service description, WSDL description, Web service's operation and port parameters which are drawn from WSDL are used as input data here. Because the input data type is keyword, the proposed LFPN cannot deal with this type of data. Consequently, a Learning Fuzzy Petri Net for Web Services Discovery

model (LFPNSD) is proposed. LFPNSD is a 10-tuple, given by LFPNSD = <*P*, *Tr*, *F*, *W*, *D*, *I*, *O*, *Th*, *α*, *β* > (as shown in Figure 13.)

where: *Tr*, *I*, *O*, *Th*, *β* are same with definition of LFPN.

$P = \{P_{input}\} \cup \{P_{output}\} = \{P_{11}, P_{12}, P_{13}\} \cup \{P_{21}, P_{22}, P_{23}, P_{24}\}$

$F \subseteq (P_{input} \times Tr) \cup (Tr \times P_{output})$

$W = F \rightarrow Keywords^{+}$, where weight function on $P_{input} \times Tr$ are different keywords of service description and weight function on $Tr \times P_{output}$ are different service invoking information.

$D = \{d_{11,a}, d_{12,b}, d_{13,c}\} \cup \{d_{21,e}, d_{22,f}, d_{23,g}, d_{24,h}\}$ is a finite set of propositions, where proposition $d_{11,a}$ is that P_{11} has a service description tokens; proposition $d_{12,b}$ is that P_{12} has a free textual description tokens; proposition $d_{13,c}$ is that P_{13} has a service operation and port parameters tokens. And the propositions $d_{21,e}$, $d_{22,f}$, $d_{23,g}$, $d_{24,h}$ are that P_{21}, P_{22}, P_{23}, P_{24} have different invoking information tokens of services.

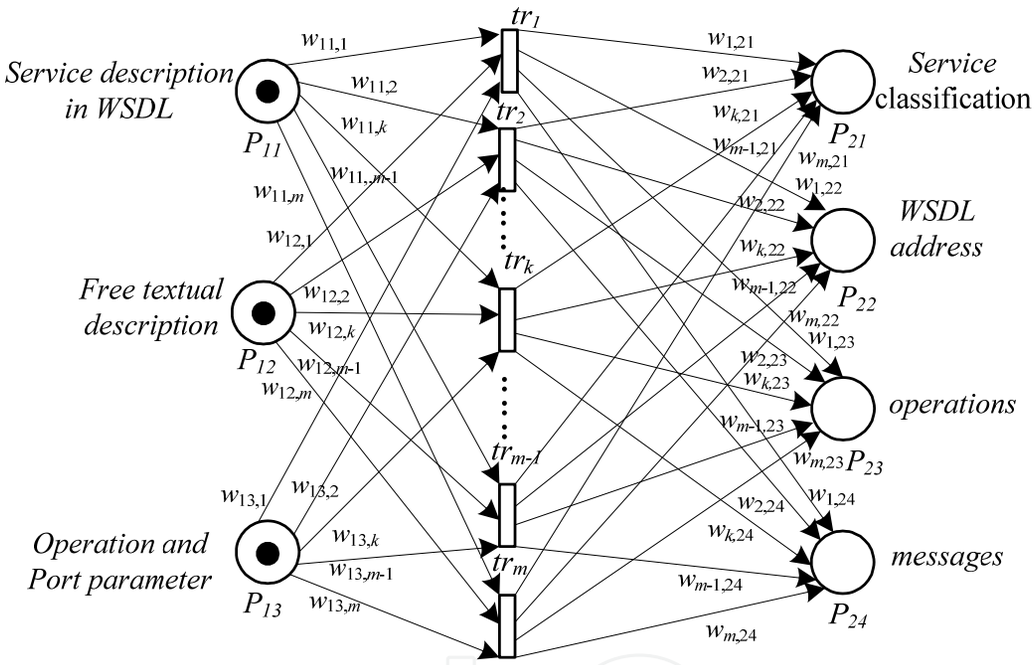


Figure 13. The learning fuzzy Petri net for Web service discovery (LFPNSD)

$\alpha(d_{ij}, tr_k) \rightarrow [0, 1]$. $\alpha(d_{ij}, tr_k) = y_i \in [0, 1]$ is the degree of truth of proposition d_{ij} to tr_k. $\alpha(d_{ij}, tr_k)$ is computed by bellow rules: if input description has n keywords and the w_{ik} on arc P_i to tr_k has s same keywords, the degree of similarity between weight keywords and input description keywords is expressed as:

$$\alpha\left(d_{ij}, tr_k\right) = 1 - \frac{|n - s|}{\max(n, s)} \tag{36}$$

The fire rule of transition: if $\alpha(d_{11,a}, tr_k) \bullet \alpha(d_{12,b}, tr_k) \bullet \alpha(d_{13,c}, tr_k) = \max((\alpha(d_{11,a}, tr_i) \bullet \alpha(d_{12,b}, tr_i) \bullet \alpha(d_{13,c}, tr_i))_{1 \le i \le m})$ and all of $\alpha(d_{11,a}, tr_k)$, $\alpha(d_{12,b}, tr_k)$, $\alpha(d_{13,c}, tr_k)$ are bigger than a threshold value th, then tr_k fires, the tokens in P_{11}~P_{13} are taken out and tokens which according to $w_{k,21}$, $w_{k,22}$, $w_{k,23}$, $w_{k,24}$ are put into P_{21}~P_{24}.

As shown in Figure 13, service free textual description, WSDL description and operation and port information are used as input vector in the learning algorithm. And, service classification, WSDL address, all of service operation names and service SOAP messages are used as output vector. Because the training data type is the keyword, the learning algorithm of LFPN is developed into a learning algorithm of LFPNSD. The learning algorithm of learning fuzzy Petri net for Web service discovery is shown in the table 5.

Learning Algorithm of LFPNSD:

Step 1. Make all weights on arcs be ◎;

Step 2. For every service in training data set,

Repeat:

2.1 Get free textual description; Draw out WSDL description and operation and port name from WSDL;

2.2 Set service textual description, WSDL description, operation and port information as input vector;

2.3 Compare the input with the keywords on the weight of input arc:

If every keyword in weight is in the input data, then compute $\alpha(d_{ij}, tr_k)$ according to formula (16), else set $\alpha(d_{ij}, tr_k)$ =0.

If each of $\alpha(d_{ij}, tr_1)$, $\alpha(d_{ij}, tr_2)$, …, $\alpha(d_{ij}, tr_{m-1})$ equates 0 and the weight of tr_m is ◎, then set $\alpha(d_{ij}, tr_m)$ =1.

If each of $\alpha(d_{ij}, tr_1)$, $\alpha(d_{ij}, tr_2)$, …, $\alpha(d_{ij}, tr_{m-1})$ equates 0◎and tr_m doesn't exist, a new transition tr_m and the arcs which connect tr_m with input and output place are constituted, set weight of arcs to be ◎◎and $\alpha(d_{ij}, tr_m)$ =1.

2.4 If $\alpha(d_{11,a}, tr_k) \bullet \alpha(d_{12,b}, tr_k) \bullet \alpha(d_{13,c}, tr_k) = \max((\alpha(d_{11,a}, tr_i) \bullet \alpha(d_{12,b}, tr_i) \bullet \alpha(d_{13,c}, tr_i))_{1 \le i \le m})$, then tr_k fires.

2.5 If the tr_k fired, get a keyword in service description but not in the weight, and add it into the weight.

If training time is t and the weight is ◎, t keywords in service description are gotten and they are added into the weight.

2.6 If the tr_k fired, compare out training data (service classification, WSDL address, service operation and message) with the weight of $w_{k,21}$, $w_{k,22}$, $w_{k,23}$, $w_{k,24}$, and calculate and record the correct rate of output.

2.7 Update $w_{k,21}$, $w_{k,22}$, $w_{k,23}$, $w_{k,24}$ according to output of training data.

Step 3. Repeat step 2, until each $\alpha(d_{11,a}, tr_k)$, $\alpha(d_{12,b}, tr_k)$, $\alpha(d_{13,c}, tr_k)$ meets the requirement value th_k.

Table 5. Learning algorithm of learning fuzzy Petri net for Web service discovery

Discussion:

1. We discuss about the learning rate γ in the learning algorithm of LFPNSD. In the algorithm, the keyword is learned and added into weights one by one. Hereby, $\boldsymbol{X}_j - \boldsymbol{W}_k^{(new)} = 1$ and $\boldsymbol{X}_j - \boldsymbol{W}^{(old)}$ equates the difference between the number of input data keywords and the number of keywords on arc weight. Because $\boldsymbol{X}_j - \boldsymbol{W}^{(old)}$ is not constant, the learning rate γ is different at each learning episode. For example, when input data has 10 keywords and arc weight has 6 keywords firstly, one keyword is learnt from input data and added into weight. In this case, the learning rate is 1/(10-6)=0.25.
2. If keyword isn't learning one by one, the keywords on $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, …, $\boldsymbol{W}_k$, …, $\boldsymbol{W}_m$ will do not balance at beginning stage of training. Then, the similar but different description

services have unbalance probability to fire transition at beginning stage of training. This makes the similar but different description services improperly fire a transition which has more keywords on its weight. It makes training efficiency lower.

3. In step 2.3 of algorithm, when each of $\alpha(d_{ij}, tr_1)$, $\alpha(d_{ij}, tr_2)$, …, $\alpha(d_{ij}, tr_{m-1})$ equates 0, it means all weights on transition tr_1~ tr_{m-1} cannot describe this service. Therefore, it is a new type service. If there is a transition which has weight arc, it is used to record the new type service; else a new transition needs to be constructed.

5.2. The result of simulation

The two simulations are carried out. One is a more efficient service selection through QoS prediction using LFPN. The other is a service selection for appropriate function using LFPNSD.

Simulation for more efficient Web service selection

During the process of Web services discovery, there are maybe several services which have same function. One service which has the best QoS needs to be select. Hereby, the service performance context is used to predict the QoS value for next execution of service. If the prediction is precise enough, an appropriate service maybe selected.

In this simulation, LFPN is used as learning model for predicting service execution time which is main part of QoS. There are 11 inputs and 1 output in this model. 11 inputs include 10 data which are last 10 times execution time of a service and one data which is reliability of the service. The output is a prediction for execution time of service's next execution. 10 transitions of LFPN is set when initialization.

A Web service performance dataset is employed for simulation. This dataset includes 100 publicly available Web services located in more than 20 countries. 150 service users executed about 100 invocations on each Web service. Each service user recorded execution time and invocation failures in dataset [27]. We selected one use's invocation data as training data. Last 10 times execution time and reliability of each service was set as input and next time execution time was set as output. 20 sets of training data were selected for each of 100 services.

The initial threshold is selected as 0.2 and the threshold is increased 0.001 at every training episode. The initial learning rate is set as 1/1.1 for every transition. The learning rate is $1/(0.1+t)$ when a transition fired t times. Prediction result and training output data are noted as $Output_{predict}$ and $Output_{training}$. Prediction precision probability Pre_{pro} is used to evaluate the precision result. And the precision probability is computed using:

$$Pre_{pro}=1-(|Output_{predict}-Output_{training}|/Output_{training}).$$

Three different training stop conditions are set as that three threshold values equal to 0.7, 0.8, and 0.9. The simulation result is listed in Table 6. Here, the number of service, which their execution time is precisely predicted, increased with the training threshold value increasing.

In the paper [3], the authors improved the traditional BP algorithm based on three-term method consisting of a learning rate, a momentum factor and a proportional factor for predicting service performance according to service context information. In this paper, this model is used to predict service execution time. The training data is same to LFPN's. And the learning rate is 0.6, momentum factor 0.9, proportional factor 1 and training times is 10,000. We compared the simulation result of the method of [3], i.e. the conventional method, with that of LFPN in Table 7. From Table 7, it is shown that Web service number of high precision in LFPN's prediction is bigger than the number of BP algorithm's prediction and Web service number of low precision in LFPN's prediction is smaller that BP algorithm's prediction. Hereby, the result of LFPN is better than result of three term's BP algorithm.

Precision	0.99~1	0.98~0.99	0.95~0.98	0.9~0.95	0.8~0.9	0.7~0.8	0.6~0.7	0~0.6
Number of Web services (*th*= 0.9)	21	14	17	15	10	8	9	6
Number of Web services (*th*= 0.8)	17	12	14	11	10	12	10	14
Number of Web services (th= 0.7)	10	10	16	8	8	11	19	18

Table 6. Prediction ability of LFPN

Precision	0.99~1	0.98~0.99	0.95~0.98	0.9~0.95	0.8~0.9	0.7~0.8	0.6~0.7	0~0.6
Number of Web services using the LFPN(th=0.9)	21	14	17	15	10	8	9	6
Number of Web services using the conventional method	6	7	15	18	20	12	10	12

Table 7. Prediction ability compares for two methods

Simulation for selection of Web service's function

In this simulation, LFPNSD is used as leaning model. The benchmark Web services which listed at www.xmethods.net are used as training data. Each service of these 260 services has a textual description and its WSDL address. And, we can get WSDL description, operation and port parameters from the WSDL. We want to classify the Web service into four classes: 1) business, 2) finance, 3) nets and 4) life services. After training, Web services are invoked by natural language request [14]. The natural language is decompounded into three inputs of this model. For example, we want to get a short message service (SMS) for sending a message to a mobile phone. The nature language of this discovery is input and decomposed into three parts: 1) WSDL description: send a message to a mobile phone; 2) free textual service description: sending a message to a mobile phone through the Internet; 3) operation and port parameters maybe have operation names: send messages, send message multiple recipients, and so on; port names send service SOAP, and so on.

In this simulation, we firstly set 100 transitions for LFPNSD model. The training stop condition is th_k ($1 \le k \le m$) ≥ 0.6. The service selection precision is recorded after every time of training. As shown in Figure 14 and 15, using LFPNSD model and its learning algorithm described in Section 5.1, every service class precision probability raised to more than 0.9 when the training time reaches to 10.

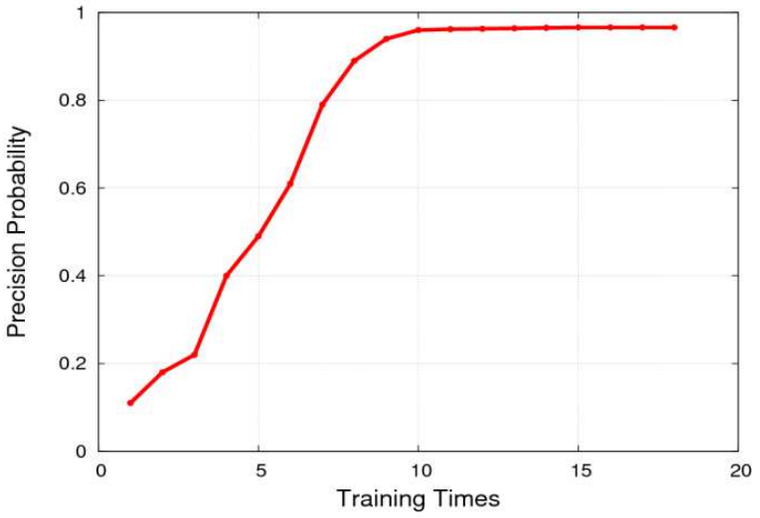


Figure 14. The results of simulation using LFPNSD and its learning algorithm− Discovery Precision Probability for total services

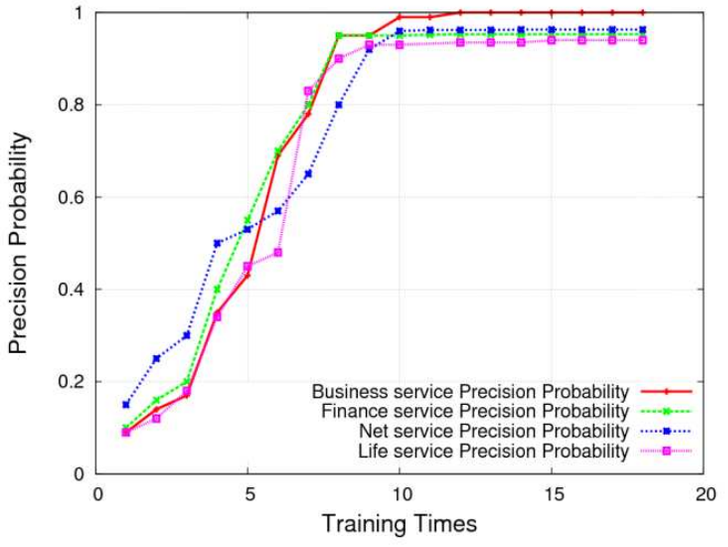


Figure 15. The results of simulation using LFPNSD and its learning algorithm − Discovery Precision Probability for classification services

A method for evaluating the proximity of services is proposed [21]. In the method, WSDL document is represented as $D^{wsdl}=\{t_1, t_2, \ldots, t_{wsdl}\}$ and $D^{desc}=\{t_1, t_2, \ldots, t_{desc}\}$ represents the textual description of the service. Because there is another descriptor of operation and port parameters in LFPNSD model, we add this descriptor as $D^{op\&port}=\{t_1, t_2, \ldots, t_{op\&port}\}$ in order to compare two methods. Here, t_{wsdl}, t_{desc} and $t_{op\&port}$ are last keyword of WSDL, textural description and operation and port parameters. In the proximity of services method, the descriptor of natural language request which is provided by a user is D_{user} and descriptor of invoked service is D_{inv}. The three Context Overlaps (CO) are defined as same keywords between $D^{wsdl}{}_{user}$, $D^{desc}{}_{user}$, $D^{op\&port}{}_{user}$ and $D^{wadl}{}_{inv}$, $D^{desc}{}_{inv}$, $D^{op\&port}{}_{inv}$. The proximity of user

requested service and invoked service is defined as a root of sum of three CO's squares. When a user invoking comes, it is compared with all services in services repository. Then, one service in D_{inv}, which has the biggest proximity value with D_{user}, was selected. We compared the discovery precision probability of this method (conventional method) with the proposed LFPNSD. The simulation results are shown in Figure 16. The LFPNSD method yielded higher precision probabilities than the conventional method proposed in [21]. Especially when the service number of Web services' repository becomes more than 88, the difference is much more significant. Here, a correct service is selected in 14 services, 24 services, 37 services, 54 services, 88 services, 151 services just as they were used in [21].

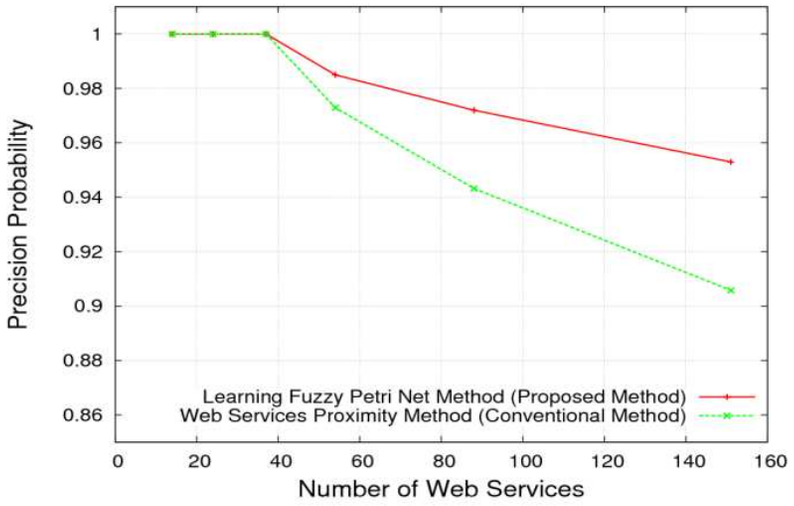


Figure 16. Comparison of two discovery methods

6. Conclusion

In this chapter, Learning Petri net (LPN) was constructed based on High-level Time Petri net and reinforcement learning (RL). The RL was used for adjusting the parameter of Petri net. Two kinds of learning algorithm were proposed for Petri net's discrete and continuous parameter learning. And verification for LPN was shown. LPN model was applied to dynamical system control. We had used the LPN in three robot systems control - the AIBO, Guide Dog. The LPN models were found and controlled for these robot systems. These robot systems could adjust their parameters while system was running. And the correctness and effectiveness of our proposed model were confirmed in these experiments. LPN model was improved to the hierarchical LPN model and this improved hierarchical LPN model was applied to QoS optimization of Web service composition. The hierarchical LPN model was constructed based on stochastic Petri net and RL. When the model was used, the Web service composition was modeled with stochastic Petri net. A Web service dynamical composing framework is proposed for optimizing QoS of web service composition. The neural network learning method was used to Fuzzy Petri net. Learning fuzzy Petri net (LFPN) was proposed. Contrasting with the existing FPN, there are three extensions in the new model: the place can possess different tokens which represent different propositions;

these propositions have different degrees of truth toward different transitions; the truth degree of proposition can be learnt through adjusting of the arc's weight function. The LFPN model obtains the capability of fuzzy production rules learning through truth degree updating. The LFPN learning algorithm which introduced network learning method into Petri net update was proposed and the convergence of the algorithm was analyzed. The LFPN model was used into discovery of Web service. Using the LFPN model, different service functional descriptions are used to evaluate service function and an appropriate service is selected firstly, Secondly, context of QoS is used to predict QoS and a more efficient service is selected.

In the future, the different intelligent computing methods will be used into Petri net for constructing different type of LPN. The efficient different types of LPN used in different special area will be compared and an efficient LPN model for solving various problems will be founded.

Author details

Liangbing Feng, Masanao Obayashi, Takashi Kuremoto and Kunikazu Kobayashi
Division of Computer Science & Design Engineering, Yamaguchi University, Ube, Japan

Liangbing Feng
Shenzhen Institutes of Advanced Technology, Shenzhen, China

7. References

[1] Konar A., Chakraborty U. K. and Wang P. P. Supervised Learning on a Fuzzy Petri Net. Information Sciences 2005; Vol.172, No.3-4, 397-416.

[2] Hrúz B., Zhou M.C. Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and Other Tools. Springer Press. London, UK, 2007.

[3] Cai H., Hu X., Lu Q. and Cao Q. A Novel Intelligent Service Selection Algorithm and Application for Ubiquitous Web Services Environment. Expert Systems with Applications 2009; Vol. 36, No. 2, 2200-2212.

[4] Doya, K. Reinforcement Learning in Continuous Time and Apace, Neural Computation, 2000; Vol.12, No.1, 219–245.

[5] Feng L. B, Obayashi M., Kuremoto T. and Kobayashi K. A Learning Petri Net Model Based on Reinforcement Learning. Proceedings of the 15th International Symposium on Artificial Life and Robotics (AROB2010); 290-293.

[6] Feng L. B., Obayashi M., Kuremoto T. and Kobayashi K. An Intelligent Control System Construction Using High-Level Time Petri Net and Reinforcement Learning. Proceedings of International Conference on Control, Automation, and Systems (ICCAS 2010); 535 – 539.

[7] Feng L. B., Obayashi M., Kuremoto T. and Kobayashi K. A learning Petri net Model. IEEJ Transactions on Electrical and Electronic Engineering 2012; Volume 7, Issue 3, pages 274–282.

[8] Frederick J. R. Statistical Methods for Speech. The MIT Press. Cambridge, Massachusetts, USA, 1999.

[9] Guangming C., Minghong L., Xianghu W. The Definition of Extended High-level Time Petri Nets. Journal of Computer Science 2006; 2(2):127-143.
[10] Hirasawa K., Ohbayashi M., Sakai S., Hu J. Learning Petri Network and Its Application to Nonlinear System Control. IEEE Transactions on Systems, Man and Cybernetic, Part B: Cybernetics, 1998; 28(6), 781-789.
[11] VIRTANEN H. E. A Study in Fuzzy Petri Nets and the Relationship to Fuzzy Logic Programming, Reports on Computer Science and Mathematics, No. 162, 1995.
[12] Yan H. S., Jian J. Agile concurrent engineering. Integrated Manufacturing Systems 1999; 10(2): 103-113.
[13] Wang J. Petri nets for dynamic event-driven system modeling. Handbook of Dynamic System Modeling 2007; Ed: Paul Fishwick, CRC Press, 1-17.
[14] Lim J. H., Lee. K. H. Constructing Composite Web Services from Natural Language Requests, Web Semantics: Science, Services and Agents on the World Wide Web 2010; Vol. 8, No.1, 1-13.
[15] Li X., Yu W., and Rsano F. L. Dynamic Knowledge Inference and Learning under Adaptive Fuzzy Petri Net Framework, IEEE Transactions on System, Man, and Cybernetics-Part C 2000; Vol. 30, No.4, 442-450.
[16] Papazoglou M.P., Georgakopoulos D. Service-Oriented Computing, Communications of the ACM, 2003; 46(10), 25–65.
[17] Platzer C. and S. Dustdar. A Vector Space Search Engine for Web Services, Proc. Third European Conf. Web Services (ECOWS'05) 2005, 62-71.
[18] Sutton R. S., Batto A. G. Reinforcement learning: An Introduction. The MIT Press, Cambridge, Massachusetts, USA, 1998.
[19] Sony OPEN-R programming group, OPEN-R programming introduction. Sony Corporation, Japan, (2004).
[20] Tzafesta S.G., Rigatos G.G. Stability analysis of an adaptive fuzzy control system using Petri nets and learning automata. Mathematics and computers in Simulation 2000; Vol. 51. No. 3. 315-339.
[21] Segev A. and E. Toch. Context-Based Matching and Ranking of Web Services for Composition, IEEE Transaction on Services computing 2009;, Vol.2, No.3, 210-222.
[22] Baranaushas V., Sarkauskas K. Colored Petri Nets-Tool for control system Learning. Electronics and Electrical Engineering 2006; 4(68):41-46.
[23] Victor R. L. Shen. Reinforcement Learning for High-level Fuzzy Petri Nets, IEEE Transactions on System, Man, and Cybernetics-Part B 2003; Vol.33, No.2, 351-361.
[24] Pedrycz W. and Gomide F. A Generalized Fuzzy Petri Net Model, IEEE Transaction on Fuzzy System 1994; Vol.2, No.4, 295-301.
[25] Xu H., Wang Y., and Jia P. Fuzzy Neural Petri Nets, Proceedings of the 4th International Symposium on Neural Networks: Part II--Advances in Neural Networks, 2007; 328 – 335.
[26] Ding Z. H., Bunke H., Schneider M. and A. Kandel. Fuzzy Time Petri net Definitions, Properties, and Applications, Mathematical and Computer Modeling, 2005, Vol. 41, No. 2-3, 345-360.
[27] Zheng Z. B., Lyu M. R. Collaborative Reliability Prediction for Service -Oriented Systems, Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering (ICSE2010), 35 – 44.