# Learning Automata-Based Adaptive Petri Net and Its Application to Priority Assignment in Queuing Systems With Unknown Parameters

S. Mehdi Vahidipour, Mohammad Reza Meybodi, and Mehdi Esnaashari

***Abstract*—In this paper, an adaptive Petri net (PN), capable of adaptation to environmental changes, is introduced by the fusion of learning automata and PN. In this new model, called learning automata-based adaptive PN (APN-LA), learning automata are used to resolve the conflicts among the transitions. In the proposed APN-LA model, transitions are portioned into several sets of conflicting transitions and each set of conflicting transitions is equipped with a learning automaton which is responsible for controlling the conflicts among transitions in the corresponding transition set. We also generalize the proposed APN-LA to ASPN-LA which is a fusion between LA and stochastic PN (SPN). An application of the proposed ASPN-LA to priority assignment in queuing systems with unknown parameters is also presented.**

***Index Terms*—Adaptive Petri net (APN), conflict resolution, learning automata, Petri nets (PNs).**

## I. Introduction

Petri nets (PNs) are graphical and mathematical modeling tools which have been applied to many different systems. They are used to describe and study information processing systems with concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic characteristics [1].

The evolution of a PN system can be described in terms of its initial marking and a number of firing rules [2]. At any given time during the evolution of a PN system, current marking of the system evolves to a new marking by applying the firing rules. Firing rules of an ordinary PN system, for any marking M, consist of three steps [2]: 1) determining the set of enabled transitions from available transitions; 2) selecting a transition from the set of enabled transitions for firing; and 3) generating a new marking by firing the selected transition.

Mechanisms which determine the set of enabled transitions from the available transitions in PNs try to reduce the number of enabled transitions by introducing concepts such as inhibitor arc [3], priority [4], [5], time [6], and color [7].

Manuscript received August 19, 2014; revised November 17, 2014; accepted January 9, 2015. Date of publication March 9, 2015; date of current version September 14, 2015. This work was supported by a grant of the Iran Telecommunications Research Center. This paper was recommended by Associate Editor M. Zhou.

S. M. Vahidipour and M. R. Meybodi are with the Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran 158754413, Iran (e-mail: vahidipour@kashanu.ac.ir).

M. Esnaashari is with the Department of Information Technology, Iran Telecommunications Research Center, Tehran 1439955471, Iran.

Digital Object Identifier 10.1109/TSMC.2015.2406764

To select a transition for firing among the enabled transitions, some mechanisms are reported in [3] and [8]–[10]. Among these mechanisms, random selection [3] is the most used mechanism. Random selection is a good mechanism if there is no conflict among enabled transitions. But, if two or more enabled transitions are in conflict, that is, firing one transition results in disabling the other(s), random selection will not be an appropriate mechanism to resolve conflicts. A set of conflicts changes when marking changes and hence the random selection for selecting the transition to be fired is inappropriate when PN is used to model the dynamics of real world problems [5], [6]. This is mainly due to the fact that random selection does not consider the dynamics of the environment under which it operates and hence, it cannot adapt itself appropriately. To tackle this problem, one approach is to add a mechanism to PN to adaptively resolve conflicts among the enabled transitions.

Learning automaton (LA) is adaptive agent used for making decisions [11]. In this paper, we propose an adaptive PN (APN) in which the LA is used as a conflict resolution mechanism. The proposed APN, called learning automata-based APN (APN-LA) is obtained from the fusion between PNs and the LA. In this model, at first, transitions are partitioned into several clusters of conflicting transitions. Then, each cluster is equipped with an LA which is responsible for controlling the conflicts among transitions in the corresponding transition cluster. Each LA has a number of actions each of which corresponds to the selection of one of the enabled transitions. During the evolution of the APN-LA, unlike standard PN, a cluster, instead of a transition, is selected for firing at each marking. A cluster can be selected for firing only if it has an enabled transition. Then, the LA associated with the cluster is activated to select one of the enabled transitions within that cluster. The selected transition will be fired and a new marking will be generated. Upon the next activation of this LA, a reinforcement signal will be generated using the sequence of markings generated between the two activations of the LA. Using the reinforcement signal produced by the environment, the internal structure of the LA will be updated to reflect the markings' changes.

We also present a generalization of APN-LA and call it ASPN-LA. The ASPN-LA is a fusion between the LA and stochastic PN (SPN). The ASPN-LA will be applied to solve the problem of priority assignment (PA) in a queuing system under unknown characteristics. The queuing system consists of

2168-2216 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

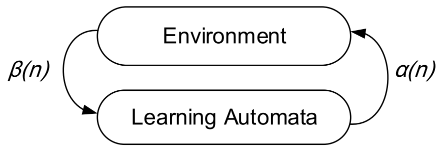


Fig. 1. Interaction between learning automata and the environment.

a server and two queues. Jobs are processed from the queues in such a way that the average waiting time of the system will be minimized. Finally, we give a solution based on the ASPN-LA to solve the problem of PA in a queuing system with $m$ queues and a server.

The rest of this paper is organized as follows. Section II gives the basic definition. Section III presents the related work. In Section IV, the proposed APN and in Section V, the proposed adaptive SPN are described. In Section VI, two applications of the ASPN-LA to the PA in queuing systems are given. Section VII is the conclusion.

## II. Learning Automata and PNs

In this section, we first briefly review learning automata and then give a short review of PNs. A LA is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment [12]. Fig. 1 shows the interaction between learning automata and the environment. An action is chosen at random as a sample realization of action probability distribution. The chosen action is then taken in the environment. The environment responds to the taken action in turn with a reinforcement signal. The action probability vector is then updated based on the reinforcement feedback from the environment.

*Definition 1:* Environment is shown by a triple $E = \{\alpha, \beta, c\}$, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents the finite set of inputs, $\beta = \{\beta_1, \beta_2, \ldots, \beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c = \{c_1, c_2, \ldots, c_m\}$ denotes the set of the penalty probabilities, where the element $c_i$ is associated with the given action $\alpha_i$.

If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a nonstationary environment. The environment depending on the nature of the reinforcement signal $\beta$ can be classified into P-, Q-, and S-models. An environment in which the reinforcement signal can take only two binary values 0 and 1 is denoted by P-model environment. In the Q-model environment, a finite number of values in the interval [0, 1] can be taken by the reinforcement signal. In an S-model environment, the reinforcement signal lies in the interval [a, b] [13].

An LA can be classified into two main families [13]: 1) fixed structure LA and 2) variable structure LA. In what follows variable structure LA used in this paper will be briefly described.

*Definition 2:* LA with variable structure is represented by $\{\alpha,\ \beta,\ q,\ L\}$, where $a = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of actions, $\beta = \{\beta_1,\ \beta_2, \ldots, \beta_m\}$ is the set of inputs, $q = \{q_1, q_2, \ldots, q_r\}$ is the action probability set, and $L$ is the learning algorithm. $r$ is the number of actions that can be chosen by the automaton.

The learning algorithm $L$ is a recurrence relation which is used to modify the action probability vector. Let $\alpha(n)$ and $q(n)$ denote the action chosen at instant $n$ and the action probability vector, respectively. The recurrence equation, shown by (1) and (2), is a linear learning algorithm using which the action probability vector $q$ is updated. Let $\alpha(n)$ be the action chosen by the automaton at instant $n$

$$q_j(n+1) = \begin{cases} q_j(n) + a(n)\big[1 - q_j(n)\big], & j = i \\ (1 - a(n))q_j(n), & \forall j \neq i \end{cases} \quad (1)$$

when the taken action is rewarded by the environment (i.e., $\beta(n) = 0$) and

$$q_j(n+1) = \begin{cases} q_j(n) + a(n)\big[1 - q_j(n)\big], & j = i \\ (1 - a(n))q_j(n), & \forall j \neq i \end{cases} \quad (2)$$

when the taken action is penalized by the environment (i.e., $\beta(n) = 1$). In (1) and (2), $a(n) \geq 0$ and $b(n) \geq 0$ denote the reward and penalty parameters that determine the amount of increases and decreases in the action probabilities, respectively. If $a(n) = b(n)$, the recurrence equations (1) and (2) are called linear reward–penalty ($L_{R-P}$) algorithm; if $a(n) \gg b(n)$, the given equations are called linear reward-$\varepsilon$ penalty ($L_{R-\varepsilon P}$); and finally if $b(n) = 0$, they are called linear reward–inaction ($L_{R-I}$). In $L_{R-I}$, the action probability vector remains unchanged when the taken action is penalized by the environment.

At the time instant $n$, an LA operates as follows: 1) the LA randomly selects an action $\alpha(n)$ based on the action probability set $q(n)$; 2) the LA performs $\alpha(n)$ on the environment and receives the environment's response $\beta(n)$; and 3) the LA updates its action probability vector using the learning algorithm. The time needed by activation of an LA with $r$ actions is $O(r)$.

Leaning automata have been successfully used in many applications such as combinatorial optimization problems [14], solving NP-complete problems [15], [16], computer networks [17], neural networks engineering [18], [19], wireless networks [20], social networks [21], and cloud computing [22], to mention a few.

In [14]–[25], two types of learning automata are presented in which the number of actions available for selection may change over time. In LA introduced in [23], when the selection probability of any action becomes zero, the action is removed from the available set of actions from that time on. In LA introduced in [24] and [25], called LA with a variable set of actions, the set of available actions can change with time and can be formally defined as below.

*Definition 3:* An LA with a variable set of actions is the one that has a set of available actions at each instance. $A = \{A_1, A_2, \ldots, A_{2^r-1}\}$ denotes the set of action subsets and $A(n)$ is the subset of all the available actions which can be chosen by the LA, at instant $n$. $\widehat{q_i}(n)$ denotes the probability of choosing action $\alpha_i$, conditioned on the event that the action subset $A(n)$ has already been selected as well as $\alpha_i \in A(n)$. The scaled probability $\widehat{q_i}(n)$ is defined as

$$\widehat{q_i}(n) = \text{prob}[\alpha(n) = \alpha_i \mid A(n),\ a_i \in A(n)] = q_i(n)/K(n) \quad (3)$$

where $K(n) = \sum_{\alpha_i \in A(n)} q_i(n)$ is the sum of the probabilities of actions in subset $A(n)$, and $q_i(n) = \text{prob}[\alpha(n) = \alpha_i]$.

The procedure of choosing an action and updating the action probabilities in an LA with a variable set of actions at instance $n$, can be described as follows [24]: 1) the probabilities of all the actions in the available set are scaled as defined in (3); 2) the automaton randomly selects one of its possible actions according to the scaled action probability vector $\hat{q}(n)$; 3) depending on the response received from the environment, the LA updates its scaled action probability vector $\hat{q}$ while the probability of the available actions is only updated; and 4) the probability vector of the actions of the chosen subset is rescaled as defined in (4) for all $\alpha_i \in A(n)$

$$q_i(n+1) = \widehat{q_i}(n+1) * K(n) \text{ if } a_i \in A(n). \tag{4}$$

*Definition 4:* PN is a triple $\{P,\ T,\ W\}$, where $P$ is a nonempty finite set of places, $T$ is a nonempty finite set of transitions, and $W{:}((P \times T) \cup (T \times P)) \to \mathbb{N}$ defines the interconnections of both sets of $P$ and $T$.

For each element $x$, either a place or a transition, its preset is defined as $x = \{y \in P \cup T \mid W(y, x) > 0\}$ and its post-set is defined as $x = \{y \in P \cup T \mid W(x, y) > 0\}$. A marking $M$ is $|P|$-vector and $M(i)$ is the nonnegative number of tokens in place $P_i$. A transition $t$ is defined to be enabled in marking $M$ (denoted by $[M, t >)$, if for every place $p_i \in t$, $M(i)$ is equal to or greater than $W(p, t)$. A transition $t$ can fire if $[M, t >$. The firing operation, denoted as $M[t > M'$, means that $[M, t >$ and that $M'$ is the next marking in the PN evolution. A PN along with an initial marking $M_0$ creates a PN system.

The set of markings reached from $M_0$ is called a reachability set and represented by a reachability graph (RG) [1], [2], [6].

*Definition 5:* SPN is a PN which is defined by a six-tuple $\{P,\ T,\ W,\ R, \omega, M_0\}$, where

1) $P,\ T,\ W, M_0$ are defined as in Definition 4;
2) $\forall t \in T, R_t \in \mathbb{R}^+ \cup \{\infty\}$ is the rate of exponential distribution for the firing time of transition $t$. If $R_t = \infty$, the firing time of $t$ is zero; such a transition is called immediate. On the other hand, a transition $t$ with $R_t < \infty$ is called timed transition. A marking $M$ is said to be vanishing if there is an enabled immediate transition in this marking; otherwise, $M$ is said to be tangible [26];
3) $\forall t \in T, \omega_t \in \mathbb{R}^+$ is the weight assigned to the firing of the enabled transition $t$, whenever its rate $R_t$ is equal to $\infty$. The firing probability of transition $t$ enabled in a vanishing marking $M$ is computed as $\omega_t / \sum_{[M,t'>} \omega_{t'}$.

It should be noted that the above definition of SPN coincides with the definition of Generalized Stochastic Petri Nets (GSPNs) given in [6].

## III. Related Work

Control for concurrent processes may be needed with respect to synchronization and conflict resolution. For PNs, control can be considered as a mechanism that specifies the order by which transitions are fired so that probable conflicts among transitions can be resolved. The most used controlling mechanism in the literature is the random selection which is inappropriate in conflicting situations [3]. The queue regime [8] is another set of mechanisms used to control the order of firing in the PN. In this mechanism, enabled transitions are put in a queue. To select a transition from the queue of transitions, different approaches have been addressed. None of these approaches is appropriate to conflicting situations when the environment is dynamic.

In priority net in which a priority level is assigned to each transition. In any marking, an enabled transition with the highest priority will be fired [4], [27]. The priority net can be either static [4] or dynamic [5], [28]. In the static priority net, the priority level of a transition does not change, whereas in the dynamic priority net, the priority level of a transition, even though different for different markings, for a specific marking is fixed and does not change as priority net evolve. For this reason, neither static priority net nor dynamic priority net is a good candidate to control firing of transitions when the environment is dynamic.

A PN in which firing of transitions is controlled by a set of finite automata is introduced in [29]. In this PN, transitions are partitioned into a number of disjoint sets. Each automaton is responsible for controlling transition firing in the corresponding set of transitions. This mechanism is able to handle conflicts among transitions, but due to the deterministic nature of finite automaton, it cannot adapt itself to the environmental changes.

Controlled PNs (CtlPNs) are a class of PNs with a new kind of places called control places [30]. A control place allows an external controller to influence the evolution of the CtlPN. In most cases, the external controller must control the evolution of CtlPN so that the system always remains in a specified set of allowed states, or equivalently, a set of forbidden markings are never reached by the CtlPN. Such a control policy can be designed by general approaches such as linear integer programming [31] and path-based approaches [32]. CtlPNs, within the framework of supervisory control [33], [34] is used as a discrete event model [35], [36]. A drawback of these mechanisms is the lack of learning capability which prevents them from being adapted to the changes in the environment.

In [37], several approaches are used in the literature to make PNs adaptive by fusing intelligent techniques such as neural networks and fuzzy logic systems with PNs [38]–[40]. But none of these mechanisms is adaptive and cannot be used to resolve conflicts among transitions when the environment is dynamic. In the following paragraphs some of these APNs are briefly described.

In [41], a feed-forward neural PN (NPN) obtained by fusing PNs with neural networks is introduced. The NPN is constructed using the basic concepts of PNs (place, transition, and arc). Places in the NPN are decomposed into three layers: 1) input; 2) hidden; and 3) output. In order to construct connections between places in hidden and output layers, threshholding transitions and arcs with trainable weights are introduced. A threshholding transition is enabled when the summation of its weighted inputs is equal to or greater than its predefined threshold. Using a set of input-output training examples, the weights of arcs are trained according to the back propagation rules. Another example of fusion between PNs and neural networks has been recently reported [42].

In this PN, a special transition, called adaption transition (A-transition), is introduced which is associated with a multilayer neural network. The inputs of the neural network are the markings of its input places whereas the outputs are the markings of its output places. The inputs of all A-transitions create an environment for the system; thus, when the environmental changes happen, A-transitions can adapt themselves to the changes. Based on the markings of the output places of a fired A-transition, a sub-graph of the RG will be traversed. Other types of fusion between PNs and neural networks are also reported in [43] and [44].

An APN obtained from the fusion of PNs and fuzzy logic systems is reported in [45] where a PN is used to represent a fuzzy rule-based system. In this PN, places represent propositions and tokens represent states of those propositions. A value between 0 and 1 is assigned to each token representing the truth degree of a proposition state. Firing a transition expresses a rule reasoning process [46], [47]. A combination of the LA with color fuzzy PN has been recently reported [48]. The LA is used to adjust adaptively the membership functions defined on the input parameters.

## IV. Proposed APN Based on LA (APN-LA)

In this section, an APN based on learning automata (APN-LA) will be proposed. The APN-LA uses learning automata to resolve conflicts among the transitions of the PN. First the APN-LA will be formally defined and second, a procedure used by the designer to construct an APN-LA will be given. Finally, to describe the evolution of the APN-LA, its firing rules will be presented. Before we give the formal definition of the APN-LA, three definitions are given.

*Definition 6:* Potential conflict is a set of transitions $\{t_1, t_2, \dots, t_n\}$ iff every $t_k \in \{t_1, t_2, \dots, t_n\}$ shares at least one input place with one $t_j \in \{t_1, t_2, \dots, t_n\}\backslash\{t_k\}$. The potential conflict only depends on the structure of the PN graph.

*Definition 7:* Maximal potential conflict is a set of transitions $\rho = \{t_1, t_2, \dots, t_n\}$ iff the following conditions are held.

1) $\{t_1, t_2, \dots, t_n\}$ are in potential conflicts.
2) For every $t \in T\backslash\rho$, $\{t\} \cup \rho$ are not in potential conflict.

We refer to a maximal potential conflict set as a cluster hereafter.

*Definition 8:* An updating transition $t^u \in T^u$, is an immediate transition, except when it fires, the action probability vector of the LA that fused with the PN is updated.

### A. Formal Definition

An APN-LA can be formally defined as follows.

*Definition 9:* An APN-LA is a five-tuples $(\hat{P}, \hat{T}, \hat{W}, S, L)$, where $\hat{P}$ is a finite set of places, $\hat{T} = T \cup T^U$ is a finite set of ordinary transitions and updating transitions $T^U = \{t_1^u, \dots, t_n^u\}$, $\hat{W}:((\hat{P} \times \hat{T}) \cup (\hat{T} \times \hat{P})) \to \mathbb{N}$ defines the interconnection of $\hat{P}$ and $\hat{T}$, $L = \{\text{LA}_1, \dots, \text{LA}_n\}$ is a set of learning automata with varying number of actions, and $S = \{s_0, s_1, \dots, s_n\}$ denotes a set of clusters, each consists of a set of transitions.

1) $s_i$, $i = 1, \dots, n$ are sets of clusters, each is a maximal potential conflict set. Each cluster $s_i$ is equipped with a LA, denoted by $\text{LA}_i$. Number of actions of $\text{LA}_i$ is equal to the number of transitions in $s_i$; each action corresponds to a transition.
2) $s_0$ is the set of remaining transitions in $\hat{T}$.

*Definition 10:* An APN-LA system is a triple $(\hat{N}, M_0, \hat{F})$, where $\hat{N}$ is an APN-LA, $M_0$ is the initial marking, $\hat{F} = \{f_1, \dots, f_n\}$ is the set of reinforcement signal generator functions. $f_i : M \to \beta_i$ is the reinforcement signal generator function related to $\text{LA}_i$. Sequence of markings in the APN-LA is the input of $f_i$ and the output of $f_i$ is reinforcement signal $\beta_i$. Upon the generation of $\beta_i$, $\text{LA}_i$ updates its action probability vector using the learning algorithm given in (3) and (4).

### B. Construction Procedure

An APN-LA is constructed by fusing a PN with learning automata according to the following steps.

1) Determine all sets $s_i$, $i = 1, \dots, n$ of maximal potential conflicts which call them clusters in the PN. The remaining transitions in this PN, which are not in any of the maximal potential conflicts sets $s_i$, $i = 1, \dots, n$, form a cluster called $s_0$.
2) For any cluster $s_i$, $i = 1, \dots, n$ do the following steps.
3) Assign $\text{LA}_i$ with a variable number of actions to $s_i$. Each action of $\text{LA}_i$ corresponds to a transition in $s_i$.
4) Insert a newly updating transition $t_i^u$ with one input and one output place.
5) Connect the newly inserted output place to $t_i^u$ with an inhibitor arc.
6) Add the newly inserted output place to the preset of all transitions in cluster $s_i$.
7) Find all shared input places of transitions in cluster $s_i$. Let the union of the presets of these places be the preset of the newly inserted input place.
8) Put a token in newly inserted output place if at least one token exists in one of the shared input places of transitions in cluster $s_i$.

### C. Evolution of APN-LA System

The evolution of a PN system can be described in terms of its initial marking and a number of firing rules [2]. At any given time during the evolution of a PN system, the current marking of the system evolves to a new marking by applying the firing rules. Firing rules of an ordinary PN system, for any marking $M$, are defined as follows [2].

1) A transition $t \in T$ is said to be enabled if each input place $p$ of $t$ is marked with at least $W(p, t)$ tokens.
2) An enabled transition may or may not fire (only one enabled transition can fire in each marking).
3) A firing of an enabled transition $t$ removes $W(p, t)$ tokens from each input place $p$ of $t$, and adds $W(t, p)$ tokens to each output place $p$ of $t$.

Firing rules of an APN-LA system differs from those of an ordinary PN due to the fusion with the LA. In the remaining of this section, we will introduce the firing rules for the APN-LA. We first give a number of definitions which will be used later for defining the firing rules of the APN-LA.

*Definition 11:* Cluster $s_i$ is enabled in marking $M$ when at least one transition in $s_i$ is enabled.

*Definition 12:* Fired cluster is an enabled cluster which is selected as the fired cluster, from which a transition will be fired in marking $M$.

*Definition 13:* $\text{LA}_i$ is activated in marking $M$ iff $s_i$ is selected as the fired cluster in marking $M$.

*Definition 14 (Effective Conflict):* A subset $E_M^s$ of a cluster $s$ is said to be in effective conflict in marking $M$ iff these conditions are held: $\forall t \in E_M^s$ is enabled in $M$ and $\{\forall t' \in s | t' \notin E_M^s,\ [M, t' \not>\}$.

Considering the above definitions, the firing rules of an APN-LA in any marking $M$ can be described as follows.

1) A transition $t \in \hat{T} = T \cup T^U$ is said to be enabled if each input place $p$ of $t$ is marked with at least $\hat{W}(p, t)$ tokens.
2) A cluster $s_i$ is said to be enabled if at least one transition in $s_i$ is enabled.
3) If $s_0$ is enabled, then $s_0$ is selected as fired with probability $|t' \in s_0, [M, t' > |/|t \in \hat{T},\ [M, t > |$, where $|\ |$ stands for norm operator; otherwise, an enabled cluster $s_i,\ i = 1, \dots, n$ is selected as fired with probability $|E_M^{s_i}|/|t \in \hat{T},\ [M, t > |$ (only one enabled cluster can be selected as the fired cluster in each marking).
4) Only one enabled transition $t \in \hat{T}$ from the fired cluster can fire in each marking $M$.
5) If $s_0$ is the fired cluster, then an enabled transition $t$ is randomly selected from $s_0$ for firing; otherwise, an LA associated with the fired cluster is activated. The available action set of the activated LA consists of the actions corresponding to the enabled transitions ($E_M^{s_i}$) in the fired cluster. The activated LA selects an action from its available action set according to its action probability vector and then the corresponding transition is selected for firing.
6) Firing of a transition $t \in \hat{T}$ removes $\hat{W}(p, t)$ tokens from each input place $p$ of $t$, and adds $\hat{W}(t, p)$ tokens to each output place $p$ of $t$.
7) The reinforcement signal generator function $f_i \in \hat{F}$ is executed when the updating transition $t_i^u \in T^u$ fires.
8) The reinforcement signal for the activated LA will be generated at the next activation of that LA.
9) Using the generated reinforcement signal, the internal structure of the LA will be updated to reflect the markings' changes.

It is worth mentioning that two or more transitions may be enabled at the same time without sharing any input places. For example, two transitions $t_2$ and $t_3$ shown in Fig. 2, are enabled at the same time without sharing any input places. Although these transitions are in actual conflict in marking $\{p_2 + p_3\}$, the potential conflict condition in Definition 6 does not hold for these transitions; our proposed APN-LA does not resolves this actual conflict. As a consequence, the proposed APN-LA, as defined above is only applicable for conflict resolution among the effective conflicts.

## V. Generalization of the APN-LA

In this section, we generalize the proposed APN-LA and obtain ASPN-LA which is a fusion between the LA and

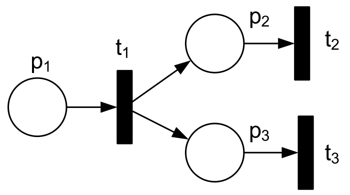


Fig. 2. Two transitions $t_2$ and $t_3$ can be enabled at the same time without sharing any input places.

the SPN. The most important difference between PN and SPN is the addition of timed transitions. A timed transition has a countdown timer, which is started whenever the transition becomes enabled. This timer is set to a value that is sampled from the negative exponential probability density function associated with the transition. The transition is fired whenever its timer reaches zero. Thus, no special mechanism is necessary for the resolution of timed conflicts: the temporal information provides a metric that used in the conflict resolution mechanism [6]. As such, to generalize the APN-LA to the ASPN-LA, it is sufficient to put all timed transitions of the SPN into a single cluster, denoted by $s_{-1}$, in which the temporal information is used to select an enabled timed transition for firing. Formally, an ASPN-LA can be defined as follows.

*Definition 15:* An ASPN-LA is a seven-tuples $(\hat{P}, \hat{T}, \hat{W}, \hat{S}, L, R, \omega^0)$, where

1) $\hat{P}, \hat{T}, \hat{W},\ L$ are defined as in Definition 9 and $R$ is defined as in Definition 5;
2) $\hat{S} = \{s_{-1},\ s_0,\ s_1, \dots, s_n\}$ denotes *a* set of clusters, each contains of a set of transitions:
    a) $s_{-1}$ contains all timed transitions in the ASPN-LA;
    b) $s_i,\ i = 1, \dots, n$ are defined as in Definition 9;
3) $\forall t \in s_0, \omega_t^0 \in \mathbb{R}^+$ is the weight assigned to an enabled transition $t$ in cluster $s_0$.

Note that the definition of $\omega_t$ in the SPN is changed into $\omega_t^0$ in the ASPN-LA. In other words, in the ASPN-LA, weights are only assigned to the transitions in cluster $s_0$. In the ASPN-LA, conflicts among immediate transitions in $s_i, i = 1, \dots, n$ are resolved by LAs; hence, no weigh is required for these sets of transitions.

Firing rules of an ASPN-LA system differs from those of an APN-LA because of the addition of the cluster $s_{-1}$. Therefore, the following rules must be added to the firing rules of the APN-LA.

1) Cluster $s_{-1}$ is said to be enabled if at least one timed transition in $s_{-1}$ is enabled and none of other clusters $s_i, i \geq 0$ is enabled.
2) Cluster $s_{-1}$ is selected as fired cluster if it is enabled.
3) If $s_{-1}$ is the fired cluster, the temporal information associated with the enabled transitions within $s_{-1}$ is used to select a transition for firing.

In addition, if $s_0$ is selected as the fired cluster, then instead of selecting a transition at random for firing, an enabled transition $t$ is selected for firing with probability $({\omega^0}_t)/(\sum_{[M,t'>} {\omega^0}_{t'})$.

It is worth mentioning that in an SPN, if several immediate transitions are enabled, a metric will be necessary to identify

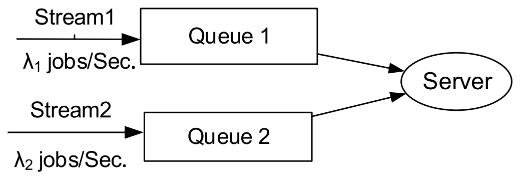


Fig. 3. Queuing system with one server and two queues.

which transition produces the marking's change. However, this is required only in those cases in which a conflict must be resolved; if the enabled transitions are concurrent, they can be fired in any order [6]. In conflicting situations, weights assigned to immediate transitions are used to determine which immediate transition will actually fire. A practical approach for assigning weights to conflicting transitions is given in [6]. The main difference between the proposed ASPN-LA and the GSPN is that, the conflict resolution mechanism in the ASPN-LA is adaptive to the environmental changes, whereas weights and priorities, assigned to immediate transitions in the GSPN, are constant.

## VI. Application of ASPN-LA

In this section, we use the proposed ASPN-LA to solve the PA problem in a queuing system with a server and multiple classes of jobs. We first construct an ASPN-LA model for a queuing system with two classes of jobs and then study its steady state behavior. Finally, a solution for the PA problem in a queuing system with $m$ queues and a server will be given.

### A. PA Problem in Queuing System With Two Queues

In a queuing system with a server and two queues (Fig. 3), there are two classes of jobs. Jobs from class $i$ arrive into queue $Q_i$ at constant Poisson rate $\lambda_i, i = 1, 2$. The service times of these classes are independent and identically distributed with exponential distribution having mean $\mu_i^{-1}, i = 1, 2$.

When the server becomes idle, it has to select a new job from either of the two queues. In other words, the server must prioritize one class of jobs to another class when both queues have jobs, waiting for service. This prioritization affects the total waiting time of the system [49] and is denoted by PA problem. Thus, the PA problem in a queuing system with two queues is the problem of how to select jobs from two queues so that the total waiting time of the system be minimized.

In a queuing system with two queues $Q_1$ and $Q_2$, if $\mu_1 > \mu_2$, then it means that the average service time of the jobs arriving into $Q_1$ is shorter than that of arriving into $Q_2$. In this queuing system, a lower total waiting time of the system is obtained when the server selects the jobs from $Q_1$ with a higher priority [49], [50]. If $\mu_1$ and $\mu_2$ are known to the server, then the PA problem can be solved by a simple mechanism such as strict priority [51]; the server always has to select the jobs from $Q_1$ with a higher priority. But, if $\mu_1$ and $\mu_2$ are unknown, then such a simple mechanism is inadequate. One way to solve the PA problem in this case, is by adopting a probabilistic priority (PP) mechanism [52]. In this mechanism, the priority of each class is defined by a probability with which its corresponding queue is selected. Here, the shorter total waiting time of the system is achieved when the server selects jobs from $Q_1$ with a higher probability (PP).

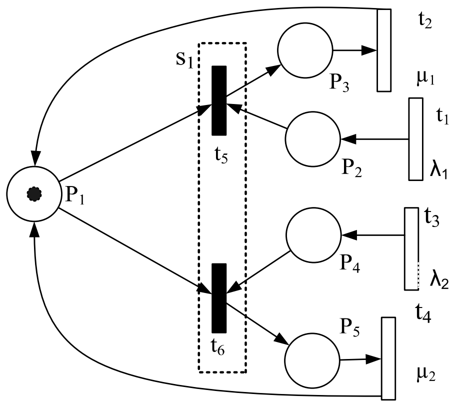


Fig. 4. Model of PP mechanism to solve the PA problem using an SPN.

One simple PP mechanism to solve the PA problem can be described using the SPN, denoted by SPN-PP (Fig. 4), where server selects jobs from either of the queues with equal probability of 0.5. In this SPN-PP, $p_1$ represents the server and $p_2$ and $p_4$ represent $Q_1$ and $Q_2$, respectively. If the server is idle ($M(p_1) = 1$) and both queues have jobs, ($M(p_2) > 1$ and $M(p_4) > 1$), then one of the jobs is selected randomly for execution. In other words, in this SPN-PP, no PA scheme is used in order to reduce the total waiting time of the system. In the following section, an adaptive PP mechanism based on the APN-LA will be described.

### B. Modeling the PA Problem With ASPN-LA

An ASPN-LA is constructed based on the SPN illustrated in Fig. 4. Note that the set $\{t_5, t_6\}$ is a maximal potential conflict and hence, forms cluster $s_1$. The transitions $t_5$ and $t_6$ are in effective conflict in marking $M = p_1 + p_2 + p_4$. Hence, we assign LA$_1$ with two actions to cluster $s_1$; each action corresponds to firing of one of the two transitions. The updating transition $t_1^u$, input place $p_6$ and output place $p_7$ are inserted into the constructed ASPN-LA an then output place $p_7$ is connected to updating transition $t_1^u$ with an inhibitor arc. The place $p_7$ is added to the preset of and $t_6$. Since place $p_1$ is the shared input place of transitions $t_5$ and $t_6$, we add preset of this place to preset of $p_6$. Since a token exists in place $p_1$ (shared input place of the transitions $t_5$ and $t_6$) a token will be placed in $p_7$. Transitions $t_1, t_2, t_3$, and $t_4$ create cluster $s_{-1}$ and $t_1^u$ creates cluster $s_0$. The ASPN-LA in which the above modifications are made is called ASPN-LA-PP which is shown in Fig. 5.

To obtain the ASPN-LA-PP system, the reinforcement signal generator function set $\hat{F} = \{f_1\}$ is needed. $f_1$ is executed upon the firing of $t_1^u$ and generates the reinforcement signal $\beta_1$ for LA$_1$. To specify how $f_1$ generates $\beta_1$, we first note that when $t_1^u$ fires for the $n$th time, the following parameters are known to the system.

1) The queue from which the last job (job $n$) was selected for execution.
2) The execution time of job $n$, denoted by $\delta(n)$.

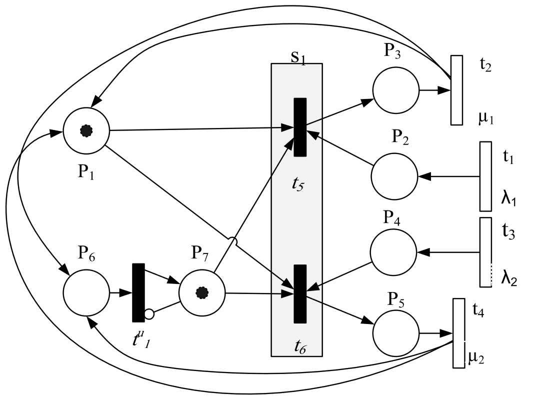


Fig. 5. ASPN-LA-PP models the PA problem in a queuing system with two queues and a server.

3) Number of jobs from $Q_i$ which have been given service so far, denoted by $k_i(\sum_i k_i = n)$.
4) Total service time for the jobs in $Q_i$ which have been given service so far, denoted by $\Delta_i(n)$.
5) Average service time for the jobs in $Q_i$ which have been given service so far can be calculated as $\Gamma_i(n) = 1/k_i \times \Delta_i(n)$.

Considering the above parameters, $f_1$ can be described using

$$\begin{cases} \beta_1 = 1;\ \delta(n) \geq \frac{1}{2} \times [\Gamma_1(n) + \Gamma_2(n)] \\ \beta_1 = 0;\ \delta(n) < \frac{1}{2} \times [\Gamma_1(n) + \Gamma_2(n)] \end{cases} \tag{5}$$

where $\beta_1 = 1$ is the penalty signal and $\beta_1 = 0$ is the reward signal. Using $\beta_1$, generated according to (5), $\text{LA}_1$ updates its available action probability vector, $\hat{q}(n)$, using the $L_{R-I}$ learning algorithm described in (1). Then the probability vector of the actions of the chosen subset is rescaled according to (4).

## C. Analysis of the ASPN-LA-PP System

Since the fusion of the LA with PNs does not change the RG of the underlying PNs, any method capable of analyzing the underlying PNs is also applicable to analyze the fused version of that PN with the LA. This is why we have been able to analyze an ASPN-LA using its underlying Continuous-Time Markov Chain (CTMC), which is used to analyze a SPN. Thus, in what follows, we first describe how a CTMC can be derived from the proposed ASPN-LA-PP system and then, we use this CTMC to study the steady-state behavior of the system.

To analyze the learning ability of the ASPN-LA-PP, we will show that $\text{LA}_1$ associated with cluster $s_1$ gradually learns to assign higher probabilities to the first class of jobs in comparison to the second class, if $\mu_1 > \mu_2$. To this end, we will partition the states of the yielded CTMC into three groups: 1) a job is selected from $Q_1$ while another job is waiting in $Q_2$; 2) a job is selected from $Q_2$ while another job is waiting in $Q_1$; and 3) remaining states. We show that $\text{LA}_1$ learns when the summation of the steady-state probabilities of states in the first group is more than that of the second group. We will show that this can happen, if the learning algorithm $L_{R-I}$ is used to update the action probability vector $q(n)$ of $\text{LA}_1$ in the ASPN-LA-PP system.

*1) Deriving CTMC From the ASPN-LA-PP:* The ASPN-LA-PP is an SPN which contains immediate and timed transitions and hence, an extended RG (ERG) can be generated from it [51]. Each node in this ERG is a reachable marking of the ASPN-LA-PP, and each directed arc from $M_i$ to $M_j$ represents the probability or rate of reaching to $M_j$ from $M_i$. ERG can be transformed into a reduced RG by eliminating the vanishing marking and the corresponding transitions [51]. A CTMC will be developed from an RG [6].

Let $L_i$ denote the number of tokens in place $p_i$ and $M_j = (L_1, \ldots, L_7)$ denote the $j$th marking of the ASPN-LA-PP. The ERG, obtained from the initial marking $M_0 = (1, 1, 0, 1, 0, 1, 0)$, is shown in Fig. 6(a). To reduce the number of markings in this ERG, we assume that places $p_2$ and $p_4$ are one-bounded [1]. This ERG contains seven vanishing markings $\{M_0, M_2, M_6, M_7, M_8, M_{10}, M_{12}\}$ represented by rectangles and thirteen tangible markings $\{M_1, M_3 - M_5, M_9, M_{11}, M_{13} - M_{15}, M_{16}, M_{17} - M_{19}\}$ represented by ovals.

Fig. 6(b) shows the corresponding finite and irreducible CTMC of Fig. 5. In this CTMC, each state j corresponds to the tangible marking $M_j$ of the ERG given in Fig. 6(a).

*2) Steady-State Analysis of the ASPN-LA-PP:* Fig. 6(a) shows that in two tangible markings, a job is selected from $Q_1$ while another job is waiting in $Q_2$, i.e., marking $M_1$ and $M_4$ which are shown in Fig. 6(a) by bold ovals. In other words, in these two markings, the ASPN-LA-PP system assigns a higher probability to the first class than to the second class. The set of markings in which higher PP is assigned to the first class is $S^{(1)} = \{s_1, s_4\}$ which are shown in Fig. 6(b) by bold circles. On the other hand, in the set of markings $S^{(2)} = \{s_3, s_5\}$, the ASPN-LA-PP system gives PP to the second class.

With CTMC in Fig. 6(b), the steady-state probability vector $\pi$ can be derived using equation $\pi\mathbb{Q} = 0$ where $\mathbb{Q}$ is the infinitesimal generator matrix [53]. In this section, we define $P^{(1)} = \pi_1 + \pi_4$ and $P^{(2)} = \pi_3 + \pi_5$, where $\pi_i$ denotes the steady-state probability of the CTMC being in state $i$ and $P^{(i)}$ denotes the steady-state probability of marking set $S^{(i)}$, $i = 1, 2$.

Assuming $\mu_1 > \mu_2$, we analyze our proposed ASPN-LA-PP system by comparing the values of $P^{(1)}$ and $P^{(2)}$ when the system reaches the steady state; if $P^{(1)} > P^{(2)}$ then the PP mechanism assigns the higher probability to the first class of jobs.

In this section, we design two theorems to prove that the ASPN-LA-PP reaches the steady states in which $P^{(1)} > P^{(2)}$. In Theorem 1, using the steady-state probabilities, we calculate a general lower bound value based on $\mu_1$ and $\mu_2$ such that if the value of $q_1(n)$ passes this value, then the ASPN-LA-PP gives higher PP to the first queue. In Theorem 2, we show that if $\text{LA}_1$ uses an $L_{R-I}$ algorithm to update its action probability vector, then $P^{(1)} > P^{(2)}$ holds when $n$ goes to infinity. In order to follow CTMC analysis by ordinary methods, the stochastic information attached to the arcs should be fixed values. This is why we use fixed values $q_1^*$ and $q_2^*$ instead of $q_1(n)$ and $q_1(n)$, respectively.

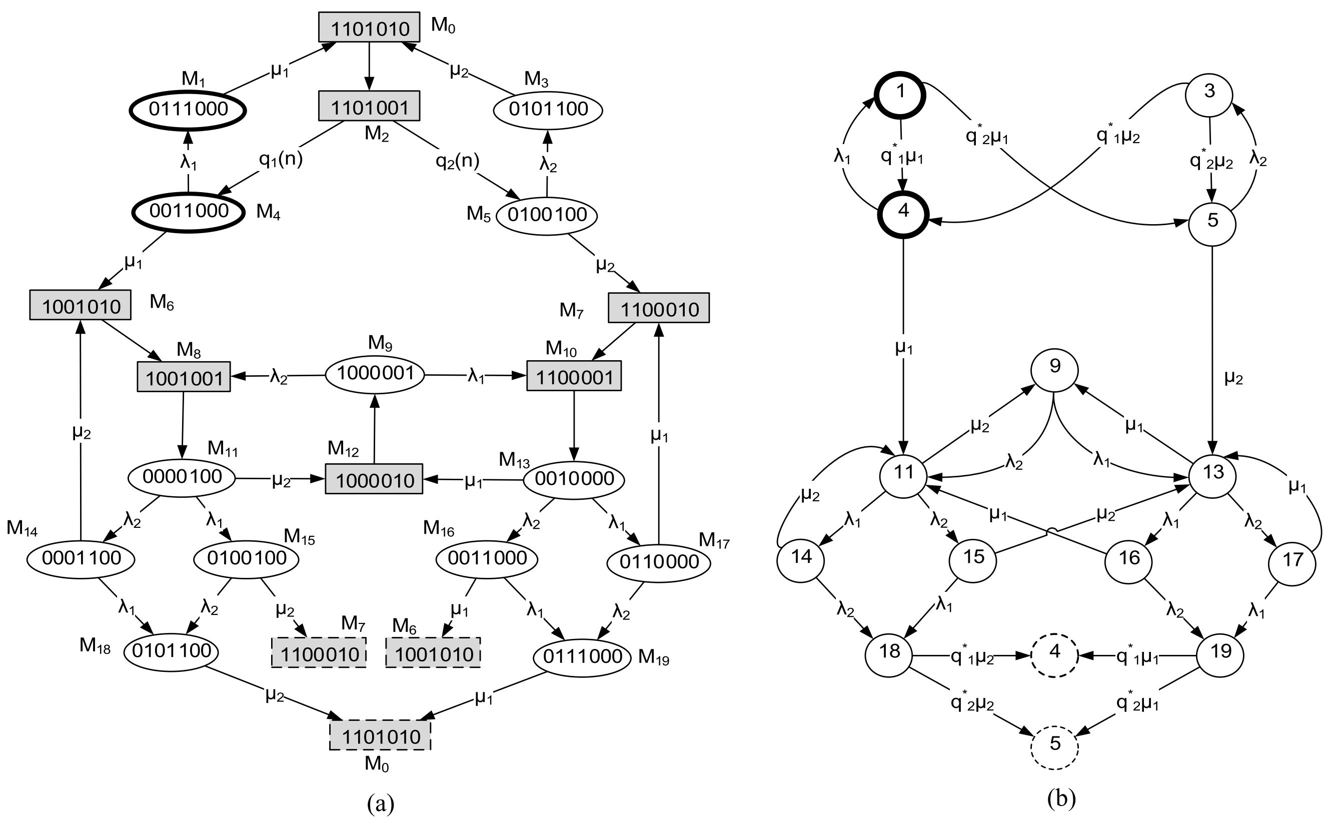


Fig. 6. (a) ERG of the ASPN-LA-PP. (b) CTMC for the ASPN-LA-PP. In the highlighted states (markings), the ASPN-LA-PP system assigns a higher probability to the first class than to the second class. For simplicity, some states (markings) has been duplicated (indicated by dash-line).

*Theorem 1:* In order for the ASPN-LA-PP to assign higher PP to the first class of jobs, the inequality $q_1^* > \mu_1/\mu_1 + \mu_2$ must be held.

*Proof:* Appendix A. ■

*Corollary 1:* Assuming $\mu_1 > \mu_2$, lower total waiting time in the ASPN-LA-PP is obtained when the assigned PP to the first queue passes $\mu_1/\mu_1 + \mu_2$ value. In other words, the values of $q_1(n)$ must be adjusted to a value upper than $\mu_1/\mu_1 + \mu_2$ by LA$_1$.

*Theorem 2:* Assuming $\mu_1 > \mu_2$ in the ASPN-LA-PP system, if algorithm $L_{R-I}$ is used to update the action probability vector $q(n)$ *of* LA$_1$, then the relation $P^{(1)} > P^{(2)}$ holds when $n$ goes to infinity.

*Proof:* Appendix B. ■

## D. Simulation Results

In this section, we conduct a set of computer simulations to study the behavior of the ASPN-LA-PP in comparison to that of the SPN-PP in terms of the average waiting time of the queuing system. To have stable queues, in all simulations, we consider $\rho_i = \lambda_i/\mu_i$, $i = 1, 2$ being less than 1. Without loss of generality, we assume $\mu_1 > \mu_2$. Let $\lambda_1, \lambda_2, \mu_1$, and $\mu_2$ take one of the following values: 0.2, 0.4, 0.6, 0.8, or 1. All reported results are averaged over 100 independent runs. Each run of simulation consists of servicing 15 000 jobs.

*1) Experiment 1:* In this experiment, we study the average waiting times of different queuing systems resulted from the ASPN-LA-PP and the SPN-PP. The results of this experiment are reported in Table I. Each row in this table represents a queuing system with specific values of $\lambda_1$, $\lambda_2$, $\mu_1$, and $\mu_2$. The average waiting times of each queuing system resulted from the SPN-PP and the ASPN-LA-PP are given in the last two columns of the corresponding row of that queuing system. Regarding this table, when the LA involves in the PA, that is the ASPN-LA-PP, the average waiting time of the system is shorter than in the SPN-PP where no learner participates in the PA process.

*2) Experiment 2:* This experiment is conducted to study the PA behavior of the ASPN-LA-PP. To this end, we define $n^*$ as the number of jobs, after which the PP of $Q_1$ is higher than that of $Q_2$, i.e., $P^{(1)} > P^{(2)}$. In this experiment, $\mu_1 = 1.0$ and $\mu_2 = 0.8$ and the values of $\rho_1$ and $\rho_2$ change in the range [0.2, 0.8]. Fig. 7 gives the results of this experiment. In this figure, region A, region below the dotted-line, contains the results of the experiment when $\rho_1 > \rho_2$ and region B (above the dotted-line) contains the results when $\rho_1 < \rho_2$. The following points are concluded from this figure.

1) $n^*$ assumes lower values in region A in comparison to region B. This is due to the fact that when $\rho_1 > \rho_2$, the server gives services to jobs in $Q_1$ more frequently than the time when $\rho_1 < \rho_2$ and hence, it can learn the parameters of this queue earlier. In other words, the ASPN-LA-PP samples from $Q_1$ more frequently and thus, the PP of this queue passes the lower bound $\mu_1/\mu_1 + \mu_2$ earlier.

TABLE I
AVERAGE WAITING TIMES OF DIFFERENT QUEUING SYSTEMS RESULTED FROM THE ASPN-LA-PP AND THE SPN-PP

| Set of parameters | | | | Mean time of service (Sec.) | |
|---|---|---|---|---|---|
| $\lambda_1$ | $\mu_1$ | $\lambda_1$ | $\mu_1$ | SPN-PP | ASPN-LA-PP |
| *0.8* | *1* | *0.6* | *0.8* | *24.59* | ***10.19*** |
| *0.8* | *1* | *0.4* | *0.8* | *14.64* | ***7.53*** |
| *0.8* | *1* | *0.2* | *0.8* | *3.56* | ***2.74*** |
| *0.6* | *1* | *0.6* | *0.8* | *18.6* | ***11.53*** |
| *0.6* | *1* | *0.4* | *0.8* | *6.04* | ***5.11*** |
| *0.6* | *1* | *0.2* | *0.8* | *0.12* | ***0.11*** |
| *0.4* | *1* | *0.6* | *0.8* | *8.3* | ***8.28*** |
| *0.4* | *1* | *0.4* | *0.8* | *0.19* | ***0.18*** |
| *0.4* | *1* | *0.2* | *0.8* | *0.05* | ***0.04*** |
| *0.2* | *1* | *0.6* | *0.8* | *0.065* | ***0.064*** |
| *0.2* | *1* | *0.4* | *0.8* | *0.034* | ***0.033*** |
| *0.2* | *1* | *0.2* | *0.8* | *26.05* | ***11.3*** |
| *0.6* | *0.8* | *0.4* | *0.6* | *6.72* | ***4.68*** |
| *0.6* | *0.8* | *0.2* | *0.6* | *12.8* | ***10.51*** |
| *0.4* | *0.8* | *0.4* | *0.6* | *0.15* | ***0.13*** |
| *0.4* | *0.8* | *0.2* | *0.6* | *0.25* | ***0.24*** |
| *0.2* | *0.8* | *0.4* | *0.6* | *19.64* | ***9.63*** |
| *0.4* | *0.6* | *0.2* | *0.4* | *25.38* | *7* |
| *0.8* | *1* | *0.4* | *0.6* | *9.04* | ***3.93*** |
| *0.8* | *1* | *0.2* | *0.6* | *25.03* | ***4.8*** |
| *0.8* | *1* | *0.2* | *0.4* | *18.7* | ***8.68*** |
| *0.6* | *1* | *0.4* | *0.6* | *0.3* | ***0.26*** |
| *0.6* | *1* | *0.2* | *0.6* | *10.45* | ***3.68*** |
| *0.6* | *1* | *0.2* | *0.4* | *5.3* | ***4.16*** |
| *0.4* | *1* | *0.4* | *0.6* | *0.08* | ***0.07*** |

2) In both regions, for a fixed $\rho_1$, by decreasing the value of $\rho_2$, the value of $n^*$ increases. Note that when $\rho_2$ decreases, the number of instances where server is idle and there exist jobs in both queues, decreases. Thus, $\text{LA}_1$ has less chance of selecting actions and receiving responses from the environment.
3) Therefore, the learning time increases, with subsequent rise in $n^*$.
4) In both regions, for a fixed $\rho_2$, by decreasing the value of $\rho_1$ the value of $n^*$ increases. This is again due to the fact that when $\rho_1$ decreases, the number of times decreases when server is idle and job queues are nonempty.
5) The least value of $n^*$ is obtained when both $\rho_1$ and $\rho_2$ assumes their highest values and the highest value $n^*$ is obtained when both $\rho_1$ and $\rho_2$ assumes their least values.

We repeated this experiment for all queuing systems reported in Table I and the results are in Fig. 8. In the figure, $x$-axis represents the values of $< \mu_1, \mu_2 >$ and $y$-axis represents the values of $< \rho_1, \rho_2 >$. Region A plots the results when $\rho_1 > \rho_2$ and region B plots the results when $\rho_2 > \rho_1$. In region A, $y$-axis is sorted first by $\rho_1$ and then by $\rho_2$, whereas in region B, $y$-axis is sorted first by $\rho_2$ and then by $\rho_1$. What we can see in this figure for all considered queuing systems is consistence with what we have concluded from Fig. 7.

## E. PA Problem in Queuing System With m Queues

In this section, we design an ASPN-LA for solving the PA problem for a queuing system with $m$ queues and a server.

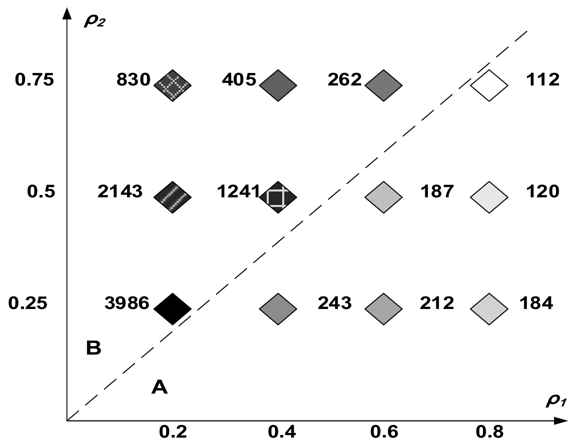


Fig. 7. Value of $\boldsymbol{n}^*$ for different queuing systems.

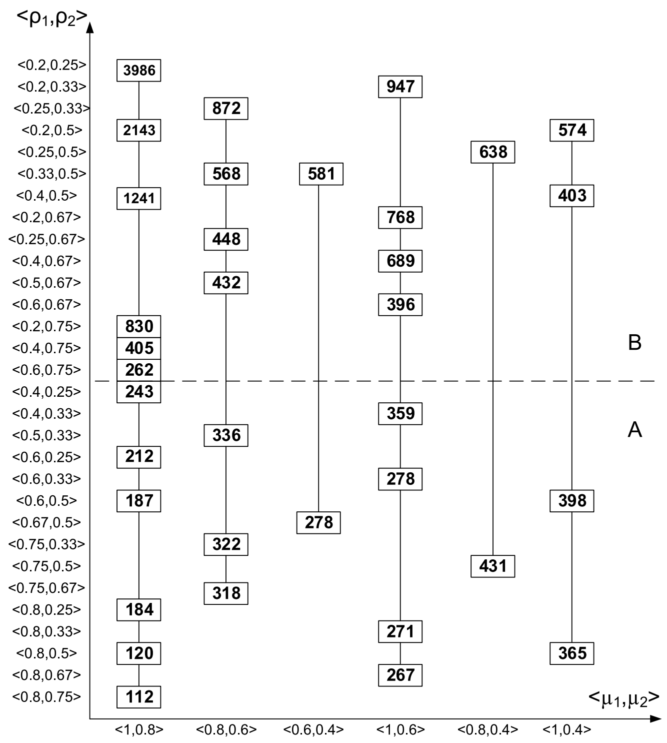


Fig. 8. Value of $\boldsymbol{n}^*$ for all queuing systems reported in Table I.

Based on the ASPN-LA-PP explained in Section VI-B, a new ASPN-LA called ASPN-LA-[$m$]PP will be constructed as shown in Fig. 9. To develop the ASPN-LA-[$m$]PP system, the reinforcement signal generator function set $\hat{F} = \{f_1\}$ is needed. $f_1$ is executed upon the firing of $t_1^u$ and generates the reinforcement signal $\beta_1$ for $\text{LA}_1$. For parameters $\delta(n)$ and $\Gamma_i(n)$, introduced in Section VI-B, $f_1$ can be described using

$$\begin{cases} \beta_1 = 1; & \delta(n) \geq \frac{1}{\mathrm{m}} \times \sum_{\mathrm{i}} \Gamma_{\mathrm{i}}(n) \\ \beta_1 = 0; & \delta(n) < \frac{1}{\mathrm{m}} \times \sum_{\mathrm{i}} \Gamma_{\mathrm{i}}(n) \end{cases} \tag{6}$$

where $\beta_1 = 1$ is the penalty signal and $\beta_1 = 0$ is the reward signal. Using $\beta_1$, generated according to (6), $\text{LA}_1$ updates its action probability vector, $\hat{q}(n)$, according to the $L_{R-I}$ algorithm

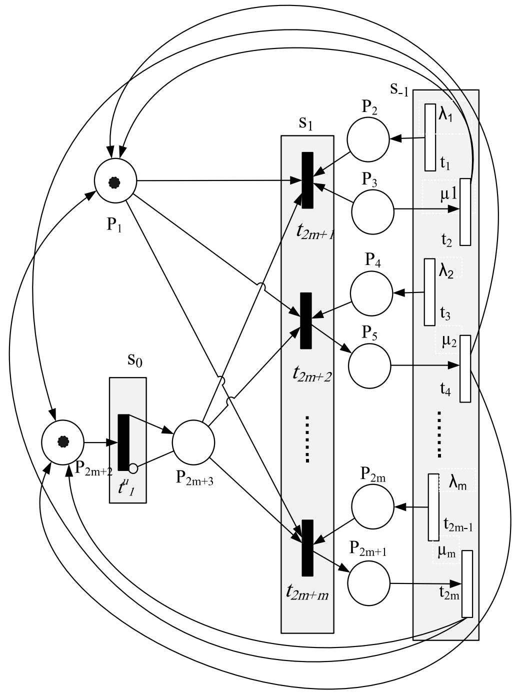


Fig. 9. ASPN-LA-[$m$]PP models the PA problem in a queuing system with $m$ queues and a server.

described in (1). Then the probability vector of the actions of the chosen subset is rescaled according to (4).

## VII. Conclusion

In this paper, we proposed a new APN which is a fusion of PN and learning automata called APN-LA. In the proposed APN-LA, conflicts among transitions are resolved adaptively with aid of learning automata. An extension of this APN obtained from fusion of the SPN and LA called ASPN-LA was also proposed. To show the applicability of these APNs, we presented their applications to solve PA problem in a queuing system with one sever and multiple queues under unknown characteristics. In PA problem, we seek a probabilistic PA strategy which can select jobs from the queues in order to minimize average waiting time of the queuing system. The steady-state behavior of the queuing system managed by the proposed APNs was analyzed theoretically and it was shown that it can assign the highest priority to the queue with the lowest average service time. Computer simulations reported in this paper did in fact verify the theoretical results.

## Appendix A

The finite and irreducible CTMC corresponding to the ASPN-LA-PP (Fig. 5) is shown in Fig. 6(b) where each state $j$ corresponds to the tangible marking $M_j$ of the ERG [Fig. 6(a)]. Considering this CTMC, the steady-state probability vector $\pi$ can be derived using equation $\pi\mathbb{Q} = 0$ where $\mathbb{Q}$ is the infinitesimal generator matrix [53]. We define $P^{(1)} = \pi_1 + \pi_4$ and $P^{(2)} = \pi_3 + \pi_5$, where $\pi_i$ denotes the steady-state probability of the CTMC being in state $i$ and $P^{(i)}$ denotes the steady-state probability of marking set $S^{(i)}, i = 1, 2$. To follow steady-state analysis to obtain $P^{(1)}$ and $P^{(2)}$, in (7), we drive equation $\pi\mathbb{Q} = 0$ for states 1, 4, 3, and 5:

$$\begin{aligned}\mu_1\pi_1 &= \lambda_1\pi_4\\ \mu_2\pi_3 &= \lambda_2\pi_5\\ \mu_1\pi_4 + \lambda_1\pi_4 &= q_1^*\mu_1\pi_1 + q_1^*\mu_2\pi_3 + q_1^*\mu_1\pi_{18} + q_1^*\mu_2\pi_{19}\\ \mu_2\pi_5 + \lambda_2\pi_5 &= q_2^*\mu_1\pi_1 + q_2^*\mu_2\pi_3 + q_2^*\mu_1\pi_{18} + q_2^*\mu_2\pi_{19}.\end{aligned} \tag{7}$$

By simplification of (7)

$$\begin{aligned}\mu_1(\pi_1 + \pi_4) &= q_1^*(\mu_1\pi_1 + \mu_2\pi_3 + \mu_1\pi_{18} + \mu_2\pi_{19})\\ \mu_2(\pi_3 + \pi_5) &= q_2^*(\mu_1\pi_1 + \mu_2\pi_3 + \mu_1\pi_{18} + \mu_2\pi_{19})\end{aligned}$$

and

$$\frac{(\pi_1 + \pi_4 = P^{(1)})}{(\pi_3 + \pi_5 = P^{(2)})} = \frac{q_1^*}{q_2^*} \times \frac{\mu_2}{\mu_1}.$$

To select more jobs from the first queue, the value of $P^{(1)}$ must be greater than $P^{(2)}$. Substituting $q_2^* = 1 - q_1^*$, the equation $q_1^* > \mu_1/\mu_1 + \mu_2$ holds. Therefore, Theorem 1 is proved.

## Appendix B

When a higher priority is assigned to the class of jobs with lower average service time, lower total waiting time is obtained [49], [50]. From learning procedure $L_{R-I}$ shown in (1), if action 1 is attempted at instant $n$, the probability $q_1(n)$ is increased at instant $n+1$ by an amount proportional to $1 - q_1(k)$ for a favorable response and fixed for an unfavorable response. By this method, it follows that $\{q(n)\}_{n>0}$ can be described by a Markov-process with state space of the unit interval [0, 1], when automaton operates in an environment with penalty probabilities $\{c_1, c_2\}$. The schema $L_{R-I}$ consists of two absorbing states $e_1 = [1, 0]$ and $e_2 = [0, 1]$. Since the probability $q_i(n)$ can decrease only when $\alpha_i$ is chosen and results a favorable response, the probability $q(k) = e_i$ holds if $q(n) = e_i$ for $i = 1, 2$ and all $k \geq n$. Hence, $V \triangleq \{e_1, e_2\}$ represents the set of all absorbing states and the Markov process $\{q(n)\}_{n>0}$ generated by the schema $L_{R-I}$ converges to the set $V$ with probability one.

To study the asymptotic behavior of the process $\{q(n)\}_{n>0}$, a common method is to compute the conditional expectation of $q_1(n+1)$ given $q_1(n)$. For the schema $L_{R-I}$, this computation shows that the expected value of $q_1(n)$ increases or decreases monotonically with $n$ depending on whether $c_2$ is greater or less than $c_1$. Study on the asymptotic behavior shows that $q_1(n)$ converges to *0* with a higher probability when $c_1 > c_2$ and to 1 with a higher probability when $c_1 < c_2$ if the initial probability is $q(0) = [1/2, 1/2]$ [13].

To prove Theorem 2, it is enough to show that penalty signal generated by the ASPN-LA-PP for the first class of jobs is lower than the one for the second class. In the queuing system, each class of job has a service time distribution with an exponential density function as

$$f_i(t) = \mu_i e^{-\mu_i t}. \tag{8}$$

The reinforcement signal generator function produces the penalty signal $\beta_i(n) = 1, i = 1, 2$ for class $i$ in instant $n$ with probability of $c_i(n), i = 1, 2$. Let $\Gamma(n) = 1/2 \times [\Gamma_1(n) + \Gamma_2(n)]$ be the average waiting time up to instant $n$. The value of $c_i(n)$ can be defined by (10) [49]

$$c_i(n) = \text{prob}[\delta_i(n) > \Gamma(n)] = e^{-\mu_i \Gamma(n)}, i = 1, 2 \qquad (9)$$

where $\delta_i(n)$ is the execution time of job $n$ selected from the $i$th class for service. Assume that $\mu_1 > \mu_2$, it is clear that $c_1(n) < c_2(n)$ holds with probability one for all $n$ [49]. Therefore, in the ASPN-LA-PP system, $q_1(n)$ converges to 1 with probability one when $n$ goes to infinity. In other words, according to Theorem 1, $q_1(n) > \mu_1/\mu_1 + \mu_2$ holds when $n$ goes to infinity when $L_{R-I}$ is used to update the action probability vector of $\text{LA}_1$. Therefore, Theorem 2 is proved.

## References

[1] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Berlin, Germany: Springer, 2013.

[2] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[3] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1981.

[4] F. Bause, "On the analysis of Petri net with static priorities," *Acta Informat.*, vol. 33, no. 5, pp. 669–685, 1996.

[5] F. Bause, "Analysis of Petri nets with a dynamic priority method," in *Application and Theory of Petri Nets*. Berlin, Germany: Springer, 1997, pp. 215–234.

[6] A. Marsan *et al.*, *Modeling With Generalized Stochastic Petri Net* (Parallel Computing). Chichester, U.K.: Wiley, 1995.

[7] K. Jensen, "Colored Petri nets and the invariant-method," *Theor. Comput. Sci.*, vol. 14, no. 3, pp. 317–336, 1981.

[8] H. D. Burkhard, "Ordered firing in Petri nets," *EIK J. Inf. Process. Cybern.*, vol. 17, nos. 2–3, pp. 71–86, 1981.

[9] H. D. Burkhard, "Control of Petri nets by finite automata," *Fundamental Informatica (Series IV),* vol. 2. Berlin, Germany: Sektion Univ., 1983, pp. 185–215.

[10] M. C. Zhou and K. Venkatesh, *Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach*. Singapore: World Scientific, 1998.

[11] M. A. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. New York, NY, USA: Springer, 2004.

[12] A. Rezvanian *et al.*, "Sampling from complex networks using distributed learning automata," *Phys. A, Stat. Mech. Appl.*, vol. 396, pp. 224–234, Feb. 2014.

[13] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. New York, NY, USA: Prentice-Hall, 1989.

[14] M. R. M. Meybodi and M. R. Meybodi, "Extended distributed learning automata: An automata-based framework for solving stochastic graph optimization problems," *Appl. Intell.*, vol. 41, no. 3, pp. 923–940, 2014.

[15] J. A. Torkestani and M. R. Meybodi, "Finding minimum weight connected dominating set in stochastic graph based on learning automata," *Inf. Sci.*, vol. 200, pp. 57–77, Oct. 2012.

[16] H. Beigy and M. R. Meybodi, "Learning automata based dynamic guard channel algorithms," *J. Comput. Electr. Eng.,* vol. 37, no. 4, pp. 601–613, 2011.

[17] M. Esnaashari and M. R. Meybodi, "A learning automata based scheduling solution to the dynamic point coverage problem in wireless sensor networks," *Comput. Netw.*, vol. 54, no. 14, pp. 2410–2438, 2010.

[18] H. Beigy and M. R. Meybodi, "A learning automata based algorithm for determination of the number of hidden units for three layers neural networks," *Int. J. Syst. Sci.*, vol. 40, no. 1, pp. 101–118, Jan. 2009.

[19] S. M. Vahidipour and M. R. Meybodi, "Learning in random neural networks using learning automata," in *Proc. Inf. Knowl. Technol. Conf.*, Shiraz, Iran, 2013.

[20] M. Esnaashari and M. R. Meybodi, "Deployment of a mobile wireless sensor network with k-coverage constraint: A cellular learning automata approach," *Wireless Netw.*, vol. 19, no. 5, pp. 945–968, 2013.

[21] F. Amiri, N. Yazdani, H. Faili, and A. Rezvanian, "A novel community detection algorithm for privacy preservation in social networks," in *Intelligent Informatics*. Berlin, Germany: Springer, 2013, pp. 443–450.

[22] H. Morshedlou and M. R. Meybodi, "Decreasing impact of SLA violations: A proactive resource allocation approach for cloud computing environments," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 156–167, Apr./Jun. 2014.

[23] J. Zhang, C. Wang, and M. C. Zhou, "Last-position elimination-based learning automata," *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2484–2492, Dec. 2014.

[24] J. A. Torkestani and M. R. Meybodi, "A learning automata-based cognitive radio for clustered wireless ad-hoc networks," *J. Netw. Syst. Manage.,* vol. 19, no. 2, pp. 278–297, 2010.

[25] M. A. Thathacher and B. R. Harita, "Learning automata with changing number of actions," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 17, no. 6, pp. 1095–1100, Nov./Dec. 1987.

[26] G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. S. Trivedi, "Automated generation and analysis of Markov reward models using stochastic reward nets," in *Linear Algebra, Markov Chains, and Queuing Models.* New York, NY, USA: Springer, 1993, pp. 145–191.

[27] M. Hack, "Petri net languages," Project MAC, Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. C.S.G. Memo 124, 1975.

[28] M. Kounty, "Modeling systems with dynamic priorities," in *Advances in Petri Nets.* Berlin, Germany: Springer, 1992, pp. 251–266.

[29] H. D. Burkhard, "An investigation of controls for concurrent systems based on abstract control languages," *Theor. Comput. Sci.*, vol. 38, nos. 2–3, pp. 193–222, 1985.

[30] L. E. Holloway and B. H. Krogh, "Controlled Petri nets: A tutorial survey," in *Proc. 11th Int. Conf. Anal. Optim. Syst. Discrete Event Syst.,* Sophia-Antipolis, France, 1994, pp. 158–168.

[31] Y. Li and W. M. Wonham, "Control of vector discrete-event systems. II. Controller synthesis," *IEEE Trans. Autom. Control*, vol. 39, no. 3, pp. 512–531, Mar. 1994.

[32] B. H. Krogh, J. Magott, and L. E. Holloway, "On the complexity of forbidden state problems for controlled marked graphs," in *Proc. 30th IEEE Conf. Decis. Control*, vol. 1. Brighton, U.K., 1991, pp. 85–91.

[33] J. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*, vol. 8. New York, NY, USA: Springer, 1998.

[34] G. Stremersch, *Supervision of Petri Nets*, vol. 13. New York, NY, USA: Springer, 2001.

[35] Y. Chen, Z. Li, and A. Al-Ahmari, "Nonpure Petri net supervisors for optimal deadlock control of flexible manufacturing systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 2, pp. 252–265, Mar. 2013.

[36] L. E. Holloway, B. Krogh, and A. Giua, "A survey of Petri net methods for controlled discrete event systems," *Discrete Event Dyn. Syst.*, vol. 7, no. 2, pp. 151–190, 1997.

[37] U. Asar, M. Zhou, and R. J. Caudill, "Making Petri nets adaptive: A critical review," in *Proc. IEEE Netw. Sens. Control*, Tucson, AZ, USA, 2005, pp. 644–649.

[38] L. Rutkowski and K. Cpalka, "Flexible neuro-fuzzy systems," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 554–574, May 2003.

[39] R. Khosla and T. Dillon, "Intelligent hybrid multi-agent architecture for engineering complex systems," in *Proc. Int. Conf. Neural Netw.*, vol. 4. Houston, TX, USA, 1997, pp. 2449–2454.

[40] L. C. Jain and N. Martin, *Fusion of Neural Networks, Fuzzy Sets and Genetic Algorithms: Industrial Applications*. Boca Raton, FL, USA: CRC Press, 1999.

[41] M. G. Kadjinicolaou, M. B. E. Abdelrazik, and G. Musgrave, "Structured analysis for neural networks using Petri nets," in *Proc. 33rd Midwest Symp. Circuits Syst.*, Calgary, AB, USA, 1990, pp. 770–773.

[42] Z. Ding, Y. Zhou, and M. C. Zhou, "Modeling self-adaptive software systems with learning Petri nets," in *Proc. Int. Conf. Softw. Eng.*, Hyderabad, India, 2014, pp. 464–467.

[43] M. Chen, J. S. Ke, and J. F. Chang, "Knowledge representation using fuzzy Petri nets," *IEEE Trans. Knowl. Data Eng.*, vol. 2, no. 3, pp. 311–319, Sep. 1990.

[44] X. F. Zha, S. Y. E. Lim, and S. C. Fok, "Integration of knowledge-based systems and neural networks: Neuro-expert Petri net models and applications," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2. Leuven, Belgium, 1998, pp. 1423–1428.

[45] M. Gao, M. Zhou, X. Huang, and Z. Wu, "Fuzzy reasoning Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 3, pp. 314–324, May 2003.

[46] G. Looney, "Fuzzy Petri nets for rule-based decision making," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 18, no. 1, pp. 178–183, Jan./Feb. 1988.

[47] J. Cardoso, R. Valette, and D. Dubois, "Possibilistic Petri nets," *IEEE Trans. Syst., Man, Cybern. B, Cybern.,* vol. 29, no. 5, pp. 573–582, Oct. 1999.

[48] S. Barzegar *et al.*, "Formalized learning automata with adaptive fuzzy colored Petri net; an application specific to managing traffic signals," *Sci. Iran.*, vol. 18, no. 3, pp. 554–565, 2011.

[49] M. R. Meybodi and S. Lashmivarahan, "A learning approach to priority assignment in a two class M/M/1 queuing system with unknown parameters," in *Proc. Yale Workshop Adapt. Syst. Theory*, New Haven, CT, USA, 1983, pp. 106–109.

[50] A. Cobham, "Priority assignment in waiting line problems," *Oper. Res.*, vol. 2, no. 1, pp. 70–76, 1954.

[51] L. Kleinrock, *Queuing Systems*. New York, NY, USA: Wiley, 1975.

[52] Y. Jiang, C. K. Tham, and C. C. Ko, "A probabilistic priority scheduling discipline for multi-service networks," *Comput. Commun.*, vol. 25, no. 13, pp. 1243–1254, 2002.

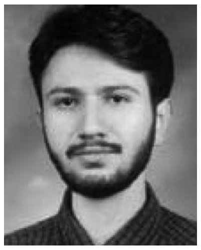
[53] G. Bolch *et al.*, *Queuing System and Markov Chain*, 2nd ed. Hoboken, NJ, USA: Wiley, 2006.

**Mohammad Reza Meybodi** received the B.S. and M.S. degrees in economics from Shahid Beheshti University, Tehran, Iran, in 1973 and 1977, respectively, and the M.S. and Ph.D. degrees in computer science from Oklahoma University, Norman, OK, USA, in 1980 and 1983, respectively.

He was an Assistant Professor at Western Michigan University, Kalamazoo, MI, USA, from 1983 to 1985, and an Associate Professor at Ohio University, Athens, OH, USA, from 1985 to 1991. He is currently a Full Professor with the Department of Computer Engineering, Amirkabir University of Technology, Tehran. His current research interests include channel management in cellular networks, learning systems, parallel algorithms, soft computing, and software development.

**S. Mehdi Vahidipour** received the B.Sc. degree in software enineering from Azad University, Kashan, Iran, in 2000 and the M.Sc. degree in artificial intelligence from Shiraz University, Shiraz, Iran, in 2003. He is currently pursuing the Ph.D. degree from the Amirkabir University of Technology, Tehran, Iran, all in computer engineering.

He is a Lecturer with the University of Kashan, Kashan, Iran. His current research interests include distributed artificial intelligence, learning automata, and adaptive Petri nets.



**Mehdi Esnaashari** received the B.S., M.S., and Ph.D. degrees in computer engineering from the Amirkabir University of Technology, Tehran, Iran, in 2002, 2005, and 2011, respectively.

He is currently an Assistant Professor with Iran Telecommunications Research Center, Tehran. His current research interests include computer networks, learning systems, soft computing, and information retrieval.