Adaptive Petri Net Based on Irregular Cellular earning Automata and Its Application in Vertex Coloring Problem

S. Mehdi Vahidipour, Mohammad Reza Meybodi and Mehdi Esnaashari

Abstract— An adaptive Petri net, called APN-LA, that has been recently introduced, uses a set of learning automata for controlling possible conflicts among the transitions in a Petri net (PN). Each learning automaton (LA) in APN-LA acts independently from the others, but there could be situations, where the operation of a LA affects the operation of another LA by possibly enabling or disabling some of the transitions within the control of that LA. In such situations, it is more appropriate to let the learning automata within the APN-LA, cooperate with each other, instead of operating independently. In this paper, an adaptive Petri net system based on Irregular Cellular Learning Automata (ICLA), in which a number of learning automata cooperate with each other, is proposed. The proposed adaptive system, called APN-ICLA, consists of two layers: PN-layer and an ICLA-layer. The PN-layer is a Petri net, in which conflicting transitions are partitioned into several clusters. There should be a controller in each cluster to control the possible conflicts among the transitions in that cluster. The ICLA-layer in APN-ICLA provides the required controllers for the PN-layer. The ICLA-layer is indeed an ICLA, in which each cell corresponds to a cluster in the PN-layer. The LA resides in a particular cell in the ICLA-layer and acts as the controller of the corresponding cluster in the PN-layer. To evaluate the efficiency of the proposed system, several algorithms, based on the APN-ICLA for vertex coloring problem, are designed. Simulation results justify the effectiveness of the proposed APN-ICLA.

***Index Terms*—Adaptive Petri net, Learning Automata, Irregular Cellular Petri nets, Vertex-Coloring Problem.**

1. INTRODUCTION

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

In such information processing systems, controlling mechanisms may be needed with respect to synchronization and conflict resolution. For PNs, control can be thought as the mechanism, which specifies the order of firing the transitions, so that probable conflicts among transitions will be resolved [2]. Different controlling mechanisms have been proposed in the literature for resolving the conflicts among a set of enabled transitions in a PN, such as random mechanism [3], queue regimes [4], priority [5][6], external controller [7], and APN-LA [2]. Among these mechanisms, only APN-LA is an adaptive one, which adapts itself to the changes in markings caused by changes of conflicts among the transitions. Such an adaptation is useful when Petri nets are used for modeling dynamics of the real world problems [2].

In the APN-LA, the controllers that control different sets of conflicting transitions in the PN, act independently from each other. But there could be situations where resolving the conflict within a set of conflicting transitions, affects another set of conflicting transitions by possibly enabling or disabling some of the transitions within that set. In such situations, it seems more appropriate to let the controllers within the PN cooperate with each other, instead of operating independently. In the other words, if resolving

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

conflicts among two different sets of transitions affect each other, then their controllers must be aware of the decisions made by the others to make better decisions.

In this paper, irregular cellular learning automata (ICLA), which is a network of learning automata [8], has been utilized and a new adaptive Petri net, called APN-ICLA, is proposed. APN-ICLA consists of two layers: PN-layer and ICLA-layer, both of which are constructed according to the problem to be solved. The PN-layer is a Petri net, in which the conflicting transitions are partitioned into several clusters. A cluster in the PN-layer is mapped into a cell of ICLA-layer, thus, the LA residing in that cell acts as the controller of that cluster. The connections among the cells in ICLA-layer are determined by the locality defined in the application, for which the APN-ICLA is designed. Two clusters in the PN-layer are said to be neighbors, if their corresponding cells in the ICLA-layer are neighbors. The firing rules for the proposed adaptive Petri net will also be defined; therefore, the definition of the APN-ICLA will be completed. Finally, several algorithms based on the APN-ICLA will be designed to solve the vertex coloring problem in graphs.

The rest of the paper is organized as follows: Section 2 provides details on the background of this paper. In Section 3, the proposed APN-ICLA is described. Section 4 provides the application of the APN-ICLA to the graph coloring. Finally, in Section 5, the conclusion of this paper has been presented.

2. Background

In this Section, first, a brief review of learning automata and irregular cellular learning automata are presented. Then, Petri net is briefly described.

2.1. Learning Automata

A Learning Automaton (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 [9]. An action is chosen randomly as a sample realization of action probability distribution. The chosen action is then taken in the environment. Then, in turn, the environment responds to the taken action with a reinforcement signal. The action probability vector is then updated based on the reinforcement feedback from the environment. The environment is shown by $E = \{\underline{\alpha}, \underline{\beta}, \underline{c}\}$, where $\underline{\alpha} = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$ is the finite set of inputs, $\underline{\beta} = \{\beta_1, \beta_2, \dots, \beta_r\}$ is the set of outputs (i.e. the reinforcement signal), and $\underline{c} = \{c_1, c_2, \dots, c_m\}$ is the set of the penalty probabilities, where the element c_i is associated with the given action α_i. In a stationary random environment, the value of $c_i, (i = 1, \dots, m)$ is constant, however, it can vary during time in a non-stationary environment. Depending on the reinforcement signal at time instant k (i.e. $\beta(k)$), the environment can be divided into three classes: 1) P-model, in which $\beta(k)$ can take only two values 0 and 1; 2) Q-model, in which $\beta(k)$ can take a finite number of values in the interval [0, 1] ; 3) S-model, in which $\beta(k)$ can take a value in the interval [a, b].

LAs are classified into fixed-structure stochastic LAs and variable-structure stochastic LAs [9]. In the following, variable structure LA, which will be used in this paper, will be briefly described. A variable structure LA is represented by $(\underline{\alpha}, \underline{\beta}, \underline{q}, L)$, where $\underline{\alpha} = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$ is the set of actions, $\underline{\beta} = \{\beta_1, \dots, \beta_r\}$ is the set of inputs, $\underline{q} = \{q_1, q_2, \dots, q_m\}$ is the action probability set and L is the learning algorithm, and m 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 $\alpha(k)$ and $q(k)$ denote the action chosen at time instant k and the action probability vector, respectively. The recurrence equation, shown by (1) and (2), is a linear learning algorithm, which updates the action probability vector $\underline{q}$ using

$$q_j(k+1) = \begin{cases} q_j(k) + a(k)\left[1 - q_j(k)\right], & j = i \\ (1 - a(k))q_j(k) & , \forall j \neq i \end{cases} \tag{1}$$

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

$$q_j(n+1)=\begin{cases}(1-b(n))q_j(n) & ,j=i\\ \left(\frac{b(n)}{r-1}\right)+(1-b(n))q_j(n), & \forall j\neq i\end{cases} \quad (2)$$

when the taken action is penalized by the environment (i.e. $\beta(k)=1$). In equations (1) and (2), $a(k)\geq 0$ and $b(k)\geq 0$ denote the reward and penalty parameters that determine the amount of increases and decreases in the action probabilities, respectively. Based on these parameters, the recurrence equations (1) and (2) are divided into three classes: 1) L_{R-P} algorithm, in which $a(k)=b(k)$; 2) $L_{R-\varepsilon P}$ algorithm, in which $a(n)\gg b(n)$; 3) L_{R-I} algorithm in which $b(n)=0$. The notation SL_{R-P} is used in this paper, when LA uses the L_{R-P} learning algorithm and operates within an S-model environment.

At the time instant k, an LA operates as follows: 1) selects an action $\alpha(k)$ randomly based on $q(k)$; 2) performs $\alpha(k)$ on the environment and receives $\beta(k)$; and, 3) updates $q(k)$ using the learning algorithm L. A learning automaton is called LA with a variable set of actions, if the set of available actions for the LA can be varied over time [8].

The LA is, by design, a simple unit, by which simple decision makings can be performed. The full potential of the LA will be realized when a cooperative effort is made by a set of interconnected LAs to achieve the group synergy. In the other words, LA can be used as the building blocks of more complex learning models, such as hierarchical system of LA [10], distributed LA (DLA) [11], extended DLA [12], cellular LA (CLA) [13], and Irregular CLA [8].

2.2. Irregular Cellular Learning Automata

Before presenting Irregular Cellular Learning Automata (ICLA) in this section, a brief introduction to Cellular Automata (CA) and Cellular Learning Automata (CLA) will be provided.

A d-dimensional CA consists of an infinite d-dimensional lattice of identical cells. Each cell can assume a state from a finite set of states. The cells update their states synchronously on discrete steps according to a local rule. The new state of each cell depends on the previous state of itself as well as the previous states of its neighbors. The states of all cells in the lattice are described by a configuration. A configuration can be described as the state of the whole lattice. The local rule and the initial configuration of CA specify the evolution of CA, that is, how the configuration of CA evolves over time.

A CLA is a CA, in which an LA is assigned to each cell [14]. Each LA residing in a particular cell determines its action (state) on the basis of its action probability vector. Like CA, there is a local rule, under which the CLA operates. The local rule of CLA and the actions selected by the neighboring LAs of any particular LA, determine the reinforcement signal to that LA. The neighboring LAs (cells) of any particular LA (cell) constitute the local environment of that LA (cell). The local environment of an LA (cell) is non-stationary due to the fact that the action probability vectors of the neighboring LAs vary during the evolution of CLA. The operation of a CLA can be described in the following steps: At the first step, the internal state of every cell is determined according to the action probability vector of the LA residing in that cell. In the second step, the local rule of CLA determines the reinforcement signal to the LA residing in that cell. Finally, each LA updates its action probability vector based on the supplied reinforcement signal and the chosen action. This process continues until the desired result is obtained.

Formally, a d-dimensional CLA is a structure $\mathbb{A}=(Z^d,N,\Phi,A,F)$, where Z^d is a lattice of d-tuples of integer numbers, $N=\{\bar{x}_1,\bar{x}_2,\ldots,\bar{x}_m\}$ is a finite subset of Z^d called neighborhood vector ($\bar{x}_1\epsilon Z^d$), Φ is a finite set of states (the state of the cell c_i is denoted by ϕ_i), A denotes the set of LA each of which is assigned to one cell of the CLA, and $F_i:\underline{\phi_i}\rightarrow\underline{\beta}$ is the local rule of the CLA in each cell c_i, where $\underline{\beta}$ is the set of values that the reinforcement signal can take. It computes the reinforcement signal for each LA based on the actions selected by the neighboring LA.

ICLA (Figure 1) is a generalization of CLA, in which the restriction of regular structure is removed [8]. An ICLA is defined as an undirected graph, in which each vertex represents a cell and is equipped with an LA, and each edge induces an adjacency relation between two cells (two LAs). The LA residing in a particular cell determines its state (action) according to its action probability vector. Similar to CLA, there is a rule, under which the ICLA operates. The rule of ICLA and the actions selected by the neighboring LAs of any particular LA, determine the reinforcement signal to that LA. The neighboring LAs of any particular LA constitute the local environment of that LA. The local environment of an LA is non-stationary, because the action probability vectors of the neighboring LAs vary during the evolution of ICLA.

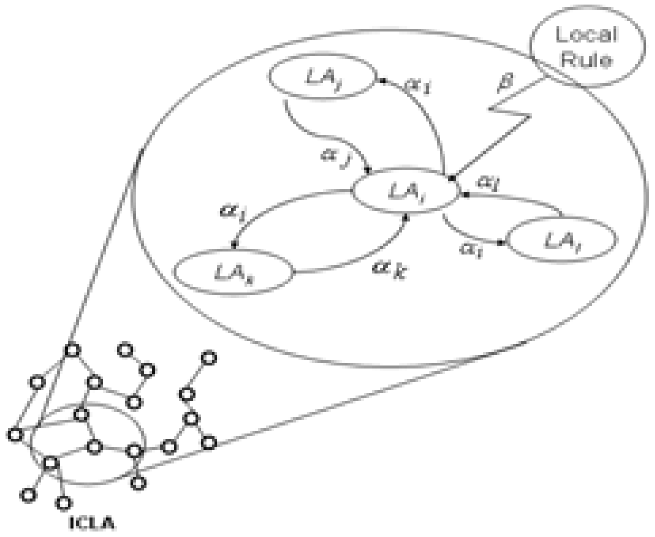

Figure 1: ICLA

The operation of ICLA is similar to the operation of CLA. At the first step, the internal state of each cell is determined based on the action probability vector of the LA residing in that cell. In the second step, the rule of ICLA determines the reinforcement signal to the LA residing in each cell. Finally, each LA updates its action probability vector according to the supplied reinforcement signal and the internal state of the cell. This process continues until the desired result is obtained. Formally, an ICLA is defined as follows [8].

Definition 1: ICLA is a structure $\mathbb{A} = (G < E, V >, \Phi, A, F)$, *where*

1) G is an undirected graph, with V as the set of vertices (cells) and E as the set of edges (adjacency relations).
2) Φ is a finite set of states. The state of the cell c_i is denoted by φ_i.
3) A is the set of LAs, each of which is assigned to one cell of ICLA.
4) $F_i: \underline{\phi}_i \to \underline{\beta}$ is the local rule of ICLA in the cell c_i, where $\underline{\phi}_i = \{\varphi_j | (i,j) \in E\} \cup \{\varphi_i\}$ is the set of states of all neighbors of c_i, and $\underline{\beta}$ is the set of valuesthat the reinforcement signal can assume. Local rule gives the reinforcement signal to each LA from the current actions selected by the neighboring LAs of that LA.

Comparing the above definition and the definition of CLA, the only difference is that the lattice $Z^d$and the neighborhood vector N in CLA are replaced by the undirected graph $G < E, V >$ in ICLA. It means that, instead of having a regular lattice structure, ICLA has an irregular graph-based structure. Note that in the definition of ICLA, no explicit definition is given for the neighborhood of each cell. It is implicitly defined in the definition of the graph G [8].

In what follows, ICLA has been considered with n cells. The LA_i, which has a finite action set $\underline{\alpha}_i$, is associated with cell c_i (for $i = 1, 2, \dots, n$) of ICLA. Let the cardinality of $\underline{\alpha_i}$ be m_i.

The operation of ICLA takes place as the following iterations. At iteration k, each LA selects an action. Let $\alpha_i \in \underline{\alpha_i}$ be the action selected by LA_i. Then all the LAs receive a reinforcement signal. Let $\beta_i \in \underline{\beta}$ be the reinforcement signal received by LA_i. This reinforcement signal is produced by the application of the local rule $F_i(\underline{\phi_i}) \to \underline{\beta}$. Higher values of β_i mean that the selected action of LA_i will receive higher penalties. Then, each LA_i updates its action probability vector according to the supplied reinforcement signal and its selected action α_i.

2.3. Petri nets

A Petri net (also known as a place/transition net or P/T net) is one of the several mathematical modeling languages for describing the distributed systems. The ordinary Petri net (PN) is a (P, T, W), where P is a non-empty finite set of places, T is a finite set of transitions, and $W: \big((P \times T) \cup (T \times P)\big) \to \mathbb{N}$ defines the interconnections of both sets of P and T. The following definitions are given for an ordinary PN:

- For each element x, either a place or a transition, its pre-set is defined as $\bullet x = \{y \in P \cup T | W(y, x) > 0\}$ and its post-set is defined as $\bullet x = \{y \in P \cup T | W(x, y) > 0\}$.
- A marking of a PN is a function $M: P \to \mathbb{N}$, where $M(p_i)$ denotes the number of tokens in p_i. A PN along with an initial marking M_0 creates a PN system denoted by (N, M_0).
- A marking M is $|P|$-vector and $M(p)$ is the non-negative number of tokens in place p.
- A set $\dot{P} \subseteq P$ is marked in a marking M, $iff\ \exists p \in \dot{P}, M(p) > 0$; otherwise $\dot{P}$ is unmarked or empty in M.
- A transition t is *enabled* in marking M, denoted by $M[t >, iff\ M(p) \geq W(p, t), \forall p \in P$.
- A transition t being enabled in marking M, may *fire* yielding a new marking given by $M'(p) = M(p) - W(p, t) + W(t, p), \forall p \in P$, denoted by $M[t > M'$.
- Two transitions t_1 and t_2 are in conflict, if both transitions are enabled in a marking M (i.e. $M[t >$ and $M[t' >$), but not both of them can be fired: $M[t > M' and\ M'[t' \ngtr$ or $M[t' > M'' and\ M''[t \ngtr$.
- Two transitions t_1 and t_2 are concurrent, if both transitions are enabled in a marking M, $M[t > and\ M[t' >$, and both transitions can be fired in any order without conflicts: $M[t > M' and\ M'[t' >$ and $M[t' > M'' and\ M''[t >$.
- The reachability set $[M_0 >$ is the smallest set that satisfies $M_0 \in [M_0 >$ and if $M' \in [M_0 >, M'[t > M''$ for some $t \in T$, then $M'' \in [M_0 >$.
- A *maximal Potential Conflict set* is a set of transitions $s_i = \{t_1, t_2, \cdots, t_{n_i}\}$ with the following conditions:
    - $\{\forall t_k \in s_i, \exists t_j \in s_i | t_k \neq t_j \wedge \bullet t_k \cap \bullet t_j \neq \emptyset\}$
    - $\{\forall t \in T \backslash s_i, \forall t_k \in s_i | \bullet t_k \cap \bullet \mathrm{t} = \emptyset\}$

A mechanism, which can be controlled, needs to be able to choose an enabled transition to fire in any specified marking. There are a number of controlling mechanisms, which have been addressed in the literature. Random selection [3] is the most commonly used controlling mechanism. Another set of mechanisms, which is applied for controlling the firing order in PNs, is the queue regime [4]. In this set of mechanisms, a queue is set for the enabled transitions. A number of approaches are addressed in the literature for selecting a transition from the transitions queue.

In the priority net, each transition is assigned a level of priority. An enabled transition, which the highest priority has been assigned to, will be fired at any marking [15]. The priority net can be either static [5] or dynamic [6]. In the static one, a transition's priority level does not change. However, in the dynamic one, a transition’s level of priority is fixed for a particular marking, but it differs for different markings.

Controlled Petri Nets (CtlPNs) are a type of PNs. However, in this class of PNs, there is a new kind of place, which is called the control place [7]. An external controller is allowed by this control place to influence the CtlPN evolution. In the majority of cases, the evolution of the CtlPN needs to be controlled by the external controller in order to keep the system in a particular set of allowed states, or in the other words, so that the CtlPN never reaches a set of forbidden markings. General methods, such as path-based approaches [16] and linear integer programming [17] can be used to design this type of control policy.

A controlling mechanism has been addressed in [2], which is called APN-LA. In this mechanism, the conflict resolvers are the learning automata. This mechanism can be applied to analyze and represent the learning algorithms, which are automata-based algorithms. For instance, in [2], APN-LA has been used by the authors for solving the problem of priority assignment in the queuing systems. In this case, the goal was providing a controlling mechanism to select jobs out of different classes, with no specified average for the service time, to be served by a single server, in order to minimize the overall waiting time of the system.

3. The Proposed Adaptive PN based on ICLA

In this section, an adaptive Petri net based on irregular cellular learning automata (APN-ICLA) will be proposed. An APN-ICLA consists of two layers: PN-layer and an ICLA-layer (Figure 2). First, the PN-layer will be formally defined and then, the ICLA-layer will be formally described. Then, a construction procedure of PN-layer from a PN is proposed. Finally, to describe the evolution of the APN-ICLA, its firing rules will be presented.

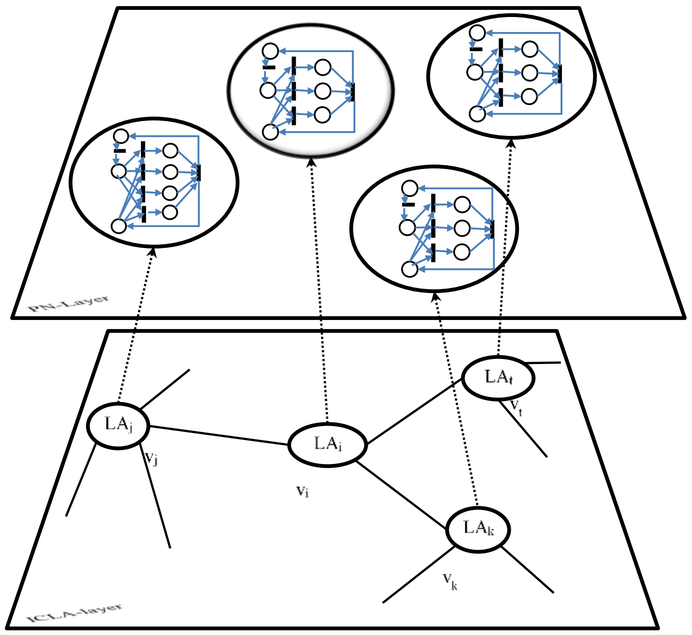

Figure 2: The two-layer model of an APN-ICLA

To construct the PN-layer, a Petri net is needed. This PN is designed according to the problem to be solved. Actually, this Petri net is used to design an algorithm to solve that problem. To create this PN-layer, first, all sets $s_i, i = 1, \ldots, n$ of maximal potential conflicts, which are called clusters in the PN, will be determined. Then, the remaining transitions in this PN, which are not in any of the maximal potential conflict sets $s_i, i = 1, \ldots, n$, form a cluster called s_0.

Every cluster $s_i, i = 1, \ldots, n$ in the PN-layer is mapped into a cell in the ICLA-layer. The connections among the cells in ICLA-layer are determined by the locality defined in the application, for which the APN-ICLA is designed. Two clusters in the PN-layer are said to be neighbors if their corresponding cells in the ICLA-layer are neighbors. The LA, which resides in a particular cell in the ICLA-layer, acts as the controller of the corresponding cluster in the PN-layer.

Similar to ICLA, there is a local rule, under which the APN-ICLA operates. The local rule of APN-ICLA and the actions selected by the neighboring LAs of any particular LA, determine the reinforcement signal to that LA. The neighboring LAs of any particular LA constitute the local environment of that LA. The local environment of an LA is non-stationary, because the action probability vectors of the neighboring LAs vary during the evolution of APN-ICLA.

3.1. Formal Definition

An APN-ICLA can be formally defined as follows:

Definition 2: An APN-ICLA with n cells is a tuple $\mathcal{N} = (PN - layer = (\hat{P}, \hat{T}, \widehat{W}, S), ICLA - layer = (G < E, V >, \Phi, A,\ F))$, *where*

1) PN-Layer:
   a. $\hat{P}$ is a finite set of places.
   b. $\hat{T} = T \cup \{t_1^u, \dots, t_n^u\}$ is a finite set of ordinary transitions and updating transitions. An updating transition t_i^u is an immediate transition, except when it fires, the internal structure of the assigned controller with cluster s_i is updated.
   c. $\widehat{W}: \left(\left(\hat{P} \times \hat{T} \right) \cup \left(\hat{T} \times \hat{P} \right) \right) \rightarrow \mathbb{N}$ defines the interconnection of $\hat{P}$ and $\hat{T}$.
   d. $S = \{s_0, s_1, \dots . s_n\}$ denotes a set of clusters, each consists of a set of transitions:
      i. $s_i, i = 1, \dots, n$ are sets of clusters, each is a maximal potential conflict set. Each cluster $s_i, i = 1, \dots, n$ is equipped with a controller which is responsible for resolving conflicts among the transitions in that cluster.
      ii. s_0 is the set of remaining transitions in $\hat{T}$.
2) *ICLA-layer:*
   a. Φ is a finite set of states. The state of the cluster $s_i, i = 1, \dots, n$ is denoted by φ_i.
   b. $A = \{LA_1, \dots . LA_n\}$ is a set of learning automata with varying number of actions.
   c. G is an undirected graph, with V as the set of vertices (clusters) and E as the set of edges (adjacency relations).
   d. $\hat{F} = \{F_1, \cdots, F_n\}$ is the set of local rules. $F_i: \underline{\Phi_i} \rightarrow \beta_i$ is the local rule of ICLA in the cluster $s_i, i = 1, \dots, n$, where $\underline{\Phi_i} = \left\{ \varphi_j | (i, j) \in E \right\} \cup \{\varphi_i\}$ is the set of states of all neighbors of s_i and β_i is the reinforcement signal. Upon the generation of β_i, LA_i updates its action probability vector using its learning algorithm.
3) $Mapping\ between\ the\ layers$:
   a. Each cluster $s_i, i = 1, \dots, n$, in the PN-layer is mapped into a cell c_i in the ICLA-layer.
   b. Two clusters s_i and s_j in the PN-layer are neighbors, if the cells c_i and c_j in the ICLA-layer are neighbors.
   c. LA_i, which resides in the cell c_i of the ICLA-layer, becomes the controller of the cluster s_i of the PN-layer. Number of actions of LA_i is equal to the number of transitions in the cluster s_i.

Definition 3: An APN-ICLA system is $(\mathcal{N}, M_0)$, *where* $\mathcal{N}$ *is an APN-ICLA and* M_0*is the initial marking.*

3.2. Construction of an PN-layer from a PN

A PN-layer is constructed from a PN according to the following steps:

1. Determine all sets $s_i, i = 1, \dots, n$ of maximal potential conflicts, which are called clusters in the PN. The remaining transitions in this PN, which are not in any of the maximal potential conflict sets $s_i, i > 0$, form a cluster called s_0.
2. For any cluster $s_i, i = 1, \dots, n$, the following steps are performed:
    a. Insert a newly updating transition t_i^u with one input and one output place.
    b. Connect the newly inserted output place to t_i^u with an inhibitor arc.
    c. Add the newly inserted output place to the pre-set of all transitions in cluster s_i.
    d. Find all shared input places of transitions in cluster s_i. Let the union of the pre-sets of these places be the pre-set of the newly inserted input place.
    e. Put a token in the newly inserted output place, if at least one token exists in one of the shared input places of transitions in cluster s_i.

To illustrate the construction of PN-layer according to the above steps, these steps are used to form a PN-layer from a Petri net given in Figure 3. The resulting PN-layer is given in Figure 4.

1. The maximal potential conflict is $\{t_1, t_2\}/\{t_4, t_5\}/\{t_6, t_7\}$. Following step 1 of the construction procedure, three clusters s_1, s_2, and s_3 are formed. These clusters are illustrated with dotted-rectangles in Figure 3. In the resulted APN-ICLA, three LAs, $LA_1/LA_2/LA_3$, with two actions, are responsible for controlling the conflicts among the transitions in $s_1/s_2/s_3$. One action of $LA_1/LA_2/LA_3$ corresponds to the selection of $t_1/t_4/t_6$ and the other action corresponds to the selection of $t_2/t_5/t_7$). The remaining transitions, t_3, t_8, and t_9 form the cluster s_0.
2. For cluster $s_1/s_2/s_3$:
    a. The updating transition $t_1^u/t_2^u/t_3^u$, the input place $p_9/p_{11}/p_{13}$ and the output place $p_{10}/p_{12}/p_{14}$ are inserted into PN (Figure 4).
    b. The newly output place $p_{10}/p_{12}/p_{14}$ is connected to the updating transition $t_1^u/t_2^u/t_3^u$ with an inhibitor arc.
    c. $p_{10}/p_{12}/p_{14}$ is added to the pre-set of$t_1/t_4/t_6$ and$t_2/t_5/t_7$.
    d. The place $p_1/p_3/p_6$ is the shared input place of the transitions $t_1/t_4/t_6$ and $t_2/t_5/t_7$. Therefore, the pre-set of this place, t_3 and t_9/t_2/t_4, is added to the pre-set of $p_9/p_{11}/p_{13}$.
    e. One/No/No token is appeared in $p_{10}/p_{12}/p_{14}$, because there is the one/no/no token in $p_1/p_3/p_6$.

3.3. Evolution of an APN-ICLA

The evolution of a Petri net can be described in terms of its initial marking and a number of firing rules [18]. At any given time during the evolution of a Petri net, the current marking of the PN evolves to a new marking by applying the firing rules.

Firing rules of an APN-ICLA differs from those of an ordinary PN, due to the fusion with the ICLA. In the remaining of this section, the firing rules for the APN-ICLA will be introduced. First, a number of definitions, which will be used later to define the firing rules of the APN-ICLA, will be provided.

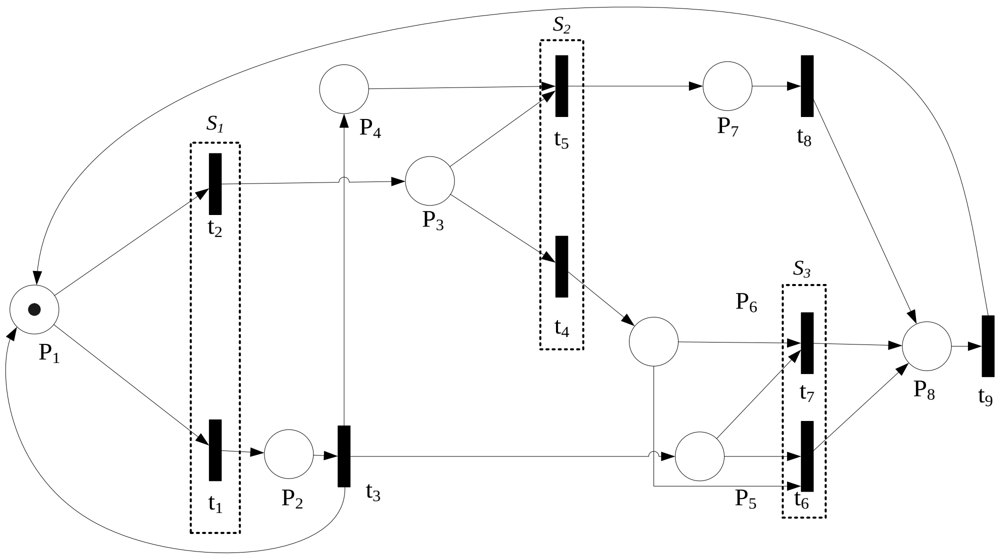

Figure 3: A PN model of a concurrent model.

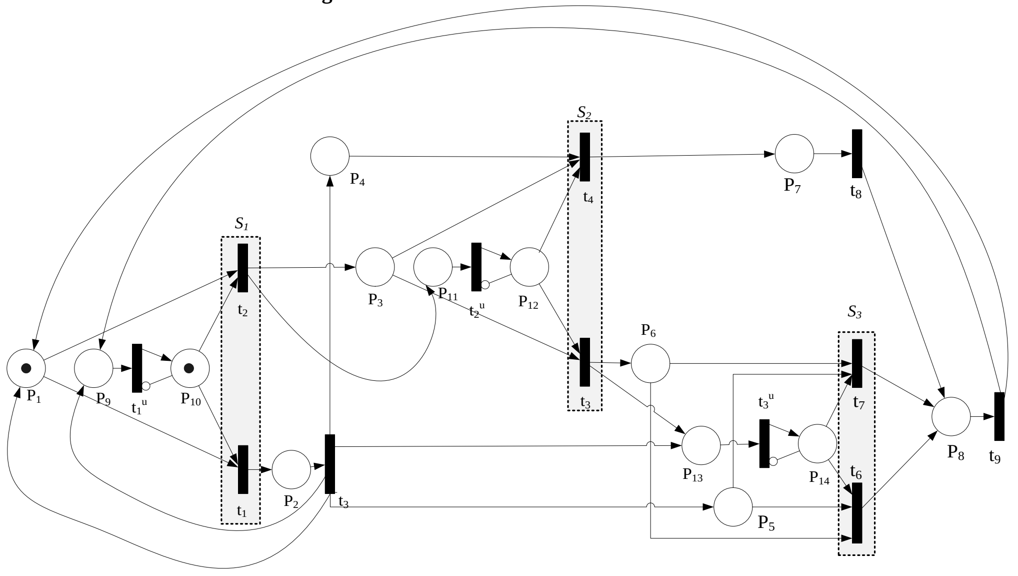

Figure 4: the yielded PN-layer from a PN shown in Figure 3

Definition 4: A cluster s_i in the PN-layer is enabled in marking M, when at least one transition in s_i is enabled.

Definition 5: LA_i in ICLA-layer is activated in marking M, if its corresponding cluster s_i in the PN-layer is enabled in marking M.

Definition 6: A fired cluster in the PN-layer is an enabled cluster, which is selected as the fired cluster, and a transition will be fired from this cluster in marking M. The activated LA, which corresponds to the fired cluster, selects this firing transition in marking M.

Definition 7: Effective Conflict: a subset $E_M^{s_i}$ of a cluster s_i is considered to be in effective conflict in marking M, if these conditions are held: $\forall t \in E_M^{s_i}$ is enabled in M and $\{\forall t' \in s | t' \notin E_M^{s_i}, \sim M[t'\rangle \}$.

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

1. A transition $t \in \hat{T} = T \cup T^U$ is considered to be enabled, if each input place p of t is marked with at least $\widehat{W}(p,t)$ tokens.
2. A cluster s_i of PN-layer is considered 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 $\frac{|t' \in s_0, M[t'\rangle|}{|t \in \hat{T}, M[t\rangle|}$, where $|\;\;|$ stands for norm operator; otherwise, an enabled cluster $s_i, i = 1, \dots, n$ is selected as fired with probability $\frac{\left|E_M^{s_i}\right|}{|t \in \hat{T}, M[t\rangle|}$ (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 be fired 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, the LA in the mapped cell 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 $\widehat{W}(p,t)$ tokens from each input place p of t, and adds $\widehat{W}(t,p)$ tokens to each output place p of t.
7. The local rule $F_i \in \hat{F}$ is used to generate the reinforcement signal β_i, when the updating transition $t_i^u \in T^u$fires.
8. Using β_i, LA_i updates its action probability vector using the learning algorithm.

4. AN APPLICATION OF APN-ICLA

In this section, the proposed APN-ICLA is used to solve the vertex coloring (VC) problem in a graph $\mathbb{G}$. Vertex coloring problem is a well-known coloring problem [19], in which a color is assigned to each vertex of the graph. A legal vertex coloring of graph $\mathbb{G} < E, V >$, where $V(\mathbb{G})$ is the set of $|V| = n$ vertices and $E(\mathbb{G})$ is the edge set, is to assign distinct colors to each vertex of the graph in such a way that no two endpoints of any edge are given the same colors. VC problem can be modeled by a quadruple $(V, E, \mathbb{C}, H)$, where V denotes the vertex-set of graph $\mathbb{G}$, E denotes the edge-set of graph $\mathbb{G}$, $\mathbb{C} = \{\mathbb{c}_1, \dots, \mathbb{c}_m\}$ denotes the set of colors assigned to the vertices, and $H: V \to \mathbb{C}$ is the coloring function that assigns a color to each vertex in such a way that $F(u) \neq F(v)$ for every $(u, v) \in E$. VC problem can be considered as a decision problem that aims at deciding for a given graph $\mathbb{G}$ whether or not the graph is P-colorable, and is called P-coloring problem. Graph $\mathbb{G} < E, V >$ is P-colorable, if it can be legally colored with at most P different colors. The chromatic number $\mathcal{X}(G)$ is the minimum number of colors required for coloring the graph, and a graph $\mathbb{G}$ is considered to be P-chromatic, if $\mathcal{X}(\mathbb{G}) = P$. It is known that an arbitrary graph can be colored with at most $\Delta + 1$ colors, where Δ denotes the maximum degree of the vertices in the graph (i.e. $\mathcal{X}(\mathbb{G}) \leq \Delta + 1$) [19].

The model used for experiments has been thoroughly described in Sections 4.1.1 and 4.1.2. Since the APN-ICLA consists of a PN-layer, first, the PN-layer for each vertex has been described in Section 4.1.1, and then, the PN-layer for the whole graph, which is called APN-ICLA-VC, has been defined in Section 4.1.2. Input data, which has been used for the experiments, is a subset of hard-to-color benchmarks, reported in DIMACS [21]. All simulations have been implemented by MATLAB 8.1 and have been run on a Pentium V Core 2 Dou 2.0 GHz. Simulation parameters are set as follows: a=b=.1, ε=.009, and τ=.95, which are described in more details later in Section 4.3.

4.1. Constructing an APN-ICLA for solving the VC Problem

To solve the VC problem with the APN-ICLA, first, a partition of PN-layer has been constructed for each vertex v_i of the graph $\mathbb{G}$ (i.e. the graph which is to be colored), and then, the complete PN-layer for the entire problem has been constructed by cascading n copies of the former layer into a PN-layer. This yielded PN-layer is used to construct an APN-ICLA, which is referred to as APN-ICLA-VC hereafter.

4.1.1. PN-layer for a Vertex

Each vertex v_i of the graph $\mathbb{G}$ is modeled by the simple PN depicted in Figure 5. In this PN, $P_{i,0}$ is the place of making decision for the color, which must be assigned to v_i and each place $P_{i,j} \in \{P_{i,1}, \dots, P_{i,m}\}$ represents a possible color for this vertex. Using the algorithm given in Section IV, this PN is converted into the PN-layer given in Figure 6. Note that the set $\{t_{i,1}, \dots, t_{i,m_i}\}$ is a maximal potential conflict, hence, it forms the cluster s_i. Transitions in this cluster are in effective conflict in the marking $M = P_{i,0} + P_{i,m_i+2}$. In this marking, the controller, which is the LA_i, is responsible for resolving effective conflicts and as a consequence, its responsible for determining the color of the vertex v_i. This is performed as follows: the action set of LA_i, which is referred to as $\underline{\alpha}_i$, contains m actions. When LA_i is activated in M, it selects one of its actions, say $\alpha_{i,r}$, according to its action probability vector. As a result, the transition $t_{i,\mathrm{r}}$, which corresponds to the selected action, is fired, hence, a token is appeared in the place $P_{i,r}$. This way, the learning automaton assigns a color to the vertex.

4.1.2. PN-layer for the Entire Problem

The PN-layer of the APN-ICLA-VC consists of n copies of the PN-layer shown in Figure 6, each of which represents a vertex of the graph.

The ICLA-layer of the APN-ICLA-VC consists of n cells; each cell c_i is a placeholder for the controller of the cluster s_i in the PN-layer. In this ICLA-layer:

1. Φ is a finite set of states. The state of the cell $c_i, i = 1, \dots, n$ is the color of vertex v_i.
2. $A = \{LA_1, \dots. LA_n\}$ is a set of learning automata with varying number of actions. The action set of $LA_i, i = 1, \dots, n$, $\underline{\alpha}_i$, contains the same number of actions denoted by m.
3. G, the graph of the ICLA, is equivalent to the undirected graph $\mathbb{G}$, which must be colored. In the other words, each cell c_i of the ICLA corresponds to the vertex v_i of the graph $\mathbb{G}$ and two cells c_i and c_j are neighbors in G, if vertices v_i and v_j are neighbors in $\mathbb{G}$.
4. $\hat{\mathrm{F}} = \{\mathrm{F}_1, \cdots, \mathrm{F}_\mathrm{n}\}$ is the set of local rules. F_i, the local rule related to LA_i, is executed upon the firing of t_i^u in the PN-layer and generates the reinforcement signal β_i for LA_i using the set of states of all neighboring cells of c_i. A simple local rule can be stated as follows:
    a. If the selected action of LA_i is different from the selected actions of all of its neighbors, then it is rewarded;
    b. Otherwise, it is penalized.

It should be noted that the local rule, stated above, is just a sample local rule. In real scenarios, any other local rules can be used instead.

Based on the proposed APN-ICLA-VC, different algorithms can be designed by:

1. Substituting different controlling mechanisms in the definition of local rules, or
2. Defining different action sets for LAs, or
3. Using different mechanisms for selecting an action, from the available action sets of LAs.

To be able to compare such algorithms with each other, an evaluative measure is required. In the next subsection, such a measure will be defined by using the results of the analysis of the proposed APN-ICLA-VC.

4.2. The analysis of the APN-ICLA-VC

Since the ICLA-layer of the APN-ICLA-VC does not change the reachability graph of the underlying PN-layer, any method capable of analyzing this layer is also able to analyze the APN-ICLA-VC. One such method is Discrete Time Markov Chain (DTMC) analysis. Thus, in what follows, first, the reachability graph of PN-layer is achieved in the vertex v_i and then, its equivalent DTMC will be determined. Finally, this DTMC will be used to study the behavior of the APN-ICLA-VC.

Let L_k denote the number of tokens in the place $p_{i,K}$ and $M_j = (L_0, L_1 \cdots L_{\mathrm{m_i}}, L_{\mathrm{m_i}+1}, L_{\mathrm{m_i}+2})$ denote the j^{th} marking of the PN-layer in the vertex v_i shown in Figure 5. The reachability graph of the vertex v_i, obtained from the initial marking $M_0 = (1,0 \ldots 0,0,1)$, is shown in Figure 7-a. The DTMC related to this reachability graph is shown in Figure 7-b.

To reduce the number of states in the DTMC, first, states $s_{i,0}$ and $s_{i,m+1}$ are eliminated from the DTMC (Figure 8-a). Since the analysis of the behavior of the DTMC in Figure 8-a is very complicated, the states of this DTMC are divided into two disjoint sets: 1) the set of allowable-coloring states ($Y_{i,1}$), and 2) the set of unallowable-coloring states ($Y_{i,2}$). $Y_{i,1}$ is the set of states, in which the color of the vertex v_i differs from the colors of all neighboring vertices, whereas $Y_{i,2}$ is the set of states, in which the color of the vertex v_i is equal to the color of at least one of the neighboring vertices. This way, the DTMC in Figure 8-a is reduced to the DTMC in Figure 8-b.

In the reduced DTMC of Figure 8-b, the transition-probability matrix $\mathbb{P}_i$ is as follows:

$$\mathbb{P}_i = \begin{bmatrix} Z_{i,2} & Z_{i,1} \\ Z_{i,2} & Z_{i,1} \end{bmatrix} \tag{3}$$

where

$$Z_{i,1} = \sum_{s_{i,j} \in Y_{i,2}} q_{i,j}\,, Z_{i,2} = \sum_{s_{i,j} \in Y_{i,1}} q_{i,j}.$$

The probability of being in the state $Y_{i,1}$ at instance time k is equal to the probability of selecting an allowable color in the vertex v_i at k. This probability can be stated using the following equation:

$$\pi_{Y_{i,1}}(k) = \sum_{l=1}^{m} \left[q_{\mathrm{i,l}}(k). \prod_{j \in \underline{\mathrm{N}}_i} \left(1 - \Gamma_{j,l}(k)\right) \right] \tag{4}$$

where $\underline{N}_i$ is the set of neighbors of the vertex v_i and $\Gamma_{i,l}(k)$ is defined according to equation (7), given below:

$$\Gamma_{i,l}(k) = \begin{cases} 1, \alpha_i(k) = \alpha_{i,l} \\ 0, \alpha_i(k) \neq \alpha_{i,l} \end{cases} \tag{5}$$

4.2.1. The Proposed Evaluative Measure

An algorithm for solving VC problem must try to 1) increase the value of $\pi_{Y_{i,1}}(k)$ in any vertex v_i of the graph, and 2) reduce the required number of colors. Thus, a measure for evaluating such algorithms considering these two features together should be defined as follows:

Definition 8: The evaluative measure $S(k)$ is defined according to the equations (6), in which $|Y_{i,1}(k)|$ indicates the number of allowable colors in v_i at instance k:

$$S(k) = \frac{1}{n} \sum_{i=1}^{n} s_i(k)$$
$$s_i(k) = \frac{\pi_{Y_{i,1}}(k)}{|Y_{i,1}(k)|} = \frac{\sum_{l=1}^{m} \left[q_{\mathrm{i,l}}(k). \prod_{j \in \underline{\mathrm{N}}_i} \left(1 - \Gamma_{j,l}(k)\right) \right]}{\sum_{l=1}^{m} \prod_{j \in \underline{\mathrm{N}}_i} \left(1 - \Gamma_{j,l}(k)\right)} \tag{6}$$

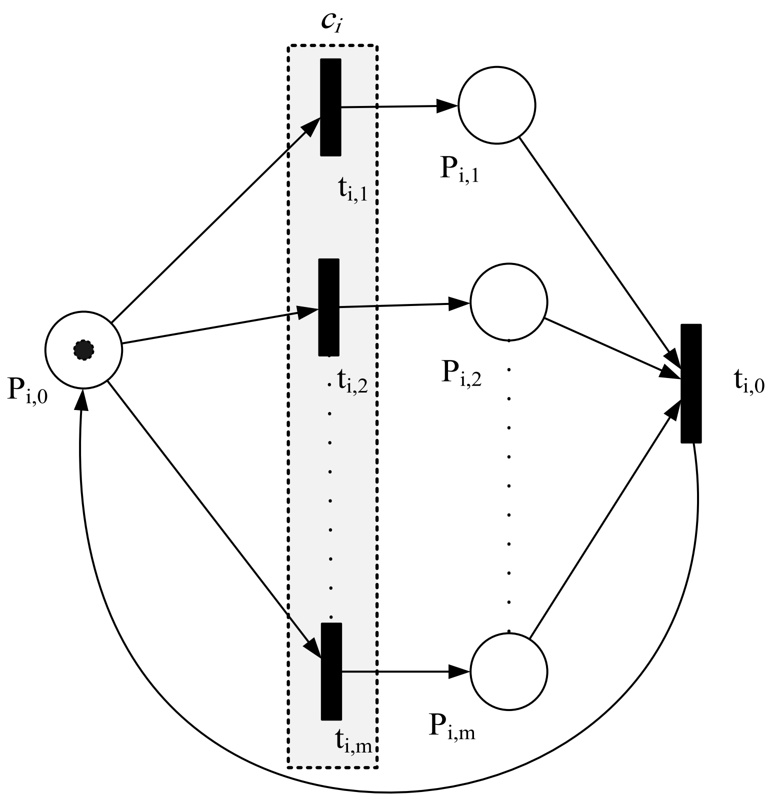

Figure 5: a simple PN for vertex v_i

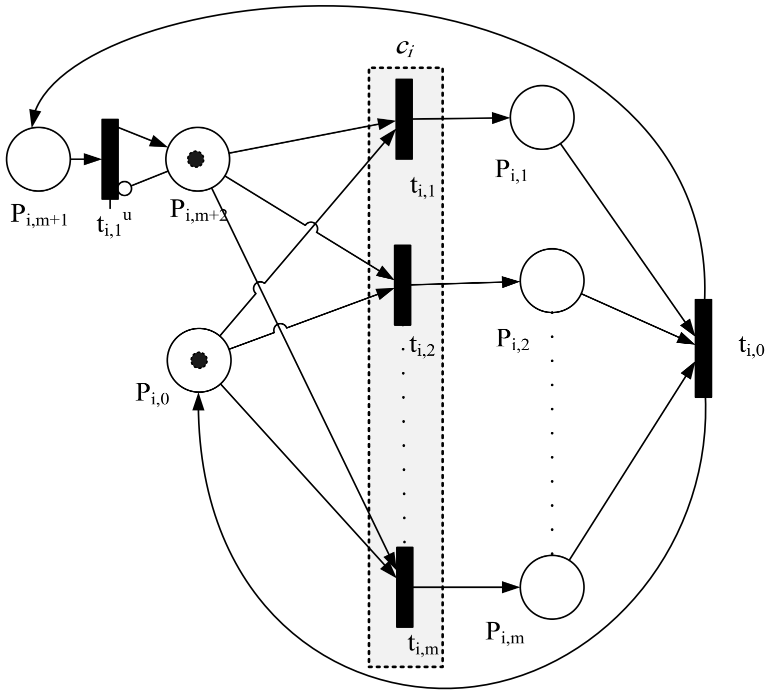

Figure 6: a partition of the PN-layer of APN-ICLA-VC for v_i

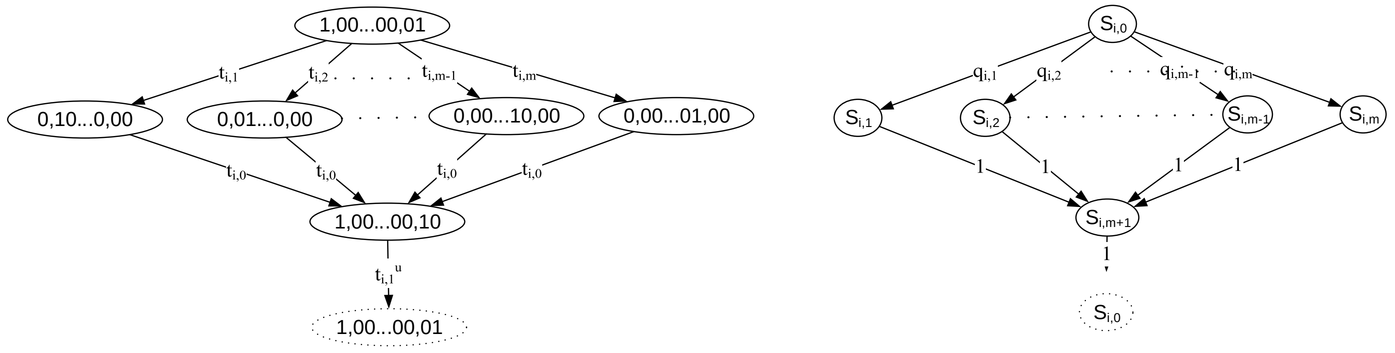

Figure 7: a) the reachability graph of APN-ICLA-VC for vertex v_i, b) the DTMC of APN-ICLA-VC for vertex v_i. For simplicity, $M_0(s_{i,0})$ has been duplicated (indicated by dash-line)

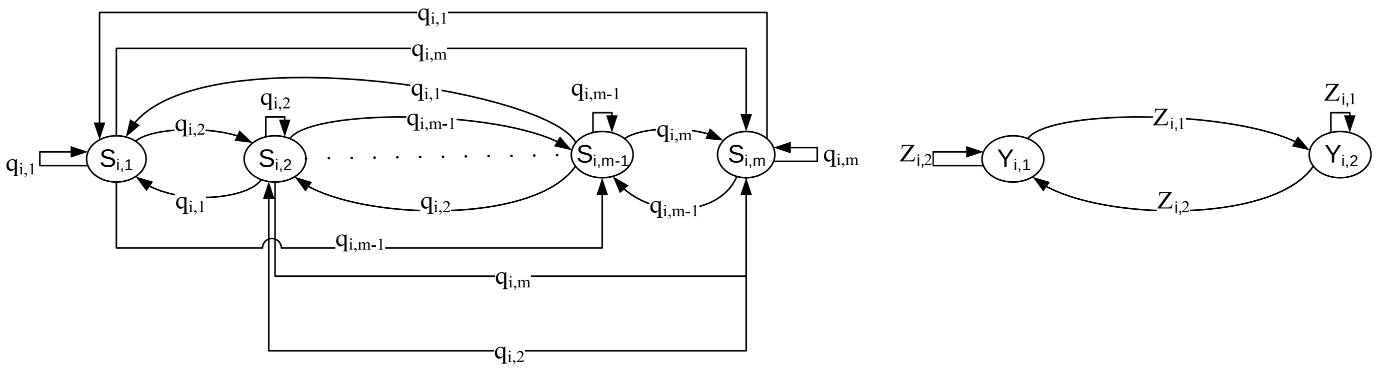

Figure 8: a) reduced DTMC for vertex v_i b) allowable and unallowable set of color assignment to vertex v_i.

4.3. Solving VC problem based on the APN-ICLA-VC

In this section, it will be shown that the APN-ICLA-VC can represent the different algorithms for solving the VC problem. Theses algorithms are compared with each other in terms of the following criteria:

1. CN: Number of colors required for coloring the graph,
2. $\mathcal{U}$: Total number of updates on the learning automata in the ICLA-layer, up to the time instant $\mathbb{k}$, where $\mathbb{k}$ is defined to be the smallest time instant, at which the colors of all vertices in the graph are fixed,
3. $\mathbb{S}(\mathbb{k})$: The average value of the evaluative measure $S(\mathbb{k})$ is defined according to the following equation:

$$\mathbb{S}(\mathbb{K}) = \frac{1}{\mathbb{K}}\sum_{k=1}^{\mathbb{K}} S(k) \qquad (7)$$

To study the efficiency of algorithms, a number of simulation studies have been conducted on a subset of hard-to-color benchmarks reported in DIMACS [21]. The rest of this section is divided into two sections. First, in the experiment One, the APN-ICLA-VC is used to implement and compare four different algorithms introduced in [19][20] in each of which an ICLA is used to solve the VC problem. Then, the experiment Two proposes a novel local rule and studies its behavior using the APN-ICLA-VC.

4.3.1. Experiment One

In this section, four different algorithms are described, namely APN1 to APN4, for solving VC problem. These algorithms are introduced in [19][20].

APN1: In the first algorithm, which in this paper is called APN1, it is assumed that the number of available colors for coloring each vertex v_i or equivalently, the number of actions in the action set of LA_i, is equal to the number of vertices in the graph $\mathbb{G}$ which is $m = \left|\underline{\alpha_i}\right| = |V|$. In the APN1, when LA_i is activated, an action from its action set is selected. All learning automata in ICLA-layer update their action probability vectors using the L_{R-I} learning algorithm. The local rules of ICLA-layer in APN1 algorithm are simple local rules, defined as follows:

- If the selected action of LA_i is different from the selected actions of all of its neighbors, then it is rewarded;
- Otherwise, it is penalized.

APN2: In the first algorithm, the number of available actions for each LA is high. This significantly prolongs the time required by each LA to find a suitable action which in turn, prolongs the total running time of the algorithm. On the other hand, as mentioned before, it is known that an arbitrary graph can be colored with at most $\Delta + 1$ colors, where Δ denotes the maximum degree of the vertices in the graph. To obtain a faster algorithm, APN2 is proposed, in which the number of available actions for each LA is equal to $\Delta + 1$. Due to the reduction in number of actions of each LA, it is expected that the APN2 colors the graph in less time and with a smaller number of colors compared to the APN1.

APN3: In APN2, all LAs have the same number of actions. But it is obvious that a vertex v_i and its neighbors can be colored using at most $\Delta_i + 1$ different colors, where Δ_i is the degree of vertex v_i. Therefore, the number of actions of each vertex v_i can be reduced to $\Delta_i + 1$. This algorithm is referred to as APN3.

APN4: In APN3, the number of available actions can be reduced in some LAs. This reduction is can be considered as an enhancement, because a suitable mechanism for solving VC problem must try to decrease the number of required colors for coloring the graph. In this enhanced algorithm, called APN4, learning automata with varying number of available action sets are used [22]. If the action probability of an action $\alpha_{i,l}$ falls below a certain threshold σ, then $\alpha_{i,l}$ is removed from the available set of actions of the LA_i.

The termination condition of all aforementioned algorithms in each vertex v_i is that the action probability of one of the actions of the learning automaton LA_i, say action $\alpha_{i,l}$, reaches a predetermined threshold τ. From this time on, the color of the vertex v_i will be fixed to the corresponding color, $\mathbb{c}_l$.

One problem with algorithms APN1 to APN4 is that they cannot guarantee to find a legal coloring of the graph. To eliminate the problem of improper coloring, the following modification is applied to all of the mentioned algorithms: All LAs are considered to be learning automata with varying number of available action sets. The available actions for an LA at any time instant k are those which are not selected by any adjacent LAs at that time instant. This property allows each LA to pick only legal colors.

In this simulation study, the time instant k is increased whenever the LA corresponding to the vertex with the maximum degree is updated. Simulation parameters are set as follows: $a = b = .1$, $\sigma = .009$, and $\tau = .95$. The results of this simulation study, which are reported in

TABLE 1, indicate that by moving from APN1 to APN4, CN and $\mathcal{U}$ criteria decrease and $\mathbb{S}(\mathbb{k})$ criterion increases. In the other words, APN4 is the best and APN1 is the worst algorithm for coloring graphs. The reason behind this is that by decreasing the number of available actions for each learning automaton, its convergence rate and its accuracy increase.

TABLE 1: results of experiment for proposed algorithms applied in APN-ICLA-VC

	APN1			APN2			APN3			APN4		
Graph	C	$\mathcal{U}$	$S(\mathbb{k})$	C	$\mathcal{U}$	$S(\mathbb{k})$	C	$\mathcal{U}$	$S(\mathbb{k})$	C	$\mathcal{U}$	$S(\mathbb{k})$

	N			N			N			N		
DSJC125_1	5	1044679	.439	5	116002	.561	5	35679	.801	5	26876	.842
DSJC250_5	19	1067235	.494	18	124642	.558	17	46410	.760	17	39055	.827
le450_15a	15	4279467	.561	15	480839	.687	15	162445	.775	15	107575	.892
le450_15b	15	4629513	.556	15	497638	.635	15	183004	.734	15	138001	.798
le450_15c	16	4672330	.562	15	589748	.663	15	311569	.762	15	265139	.815
le450_15d	17	5284660	.585	15	502709	.663	15	299948	.771	15	203358	.809
le450_25c	26	5516730	.469	26	6211564	.607	25	313236	.790	25	255099	.825
le450_25d	27	5737808	.448	26	6368408	.592	25	334672	.699	25	275394	.786

4.3.2. Experiment Two

This simulation study is conducted to evaluate the effect of the local rule used in the ICLA-layer of the APN-ICLA-VC, on the algorithm used for solving VC problem. For this simulation study, the simple local rules used by APN4 are modified as follows:

- The selected action of LA_i is rewarded if it is different from the selected actions of all of its neighbors and s_i(k) ≥ $\mathbb{S}_i$(k), where $\mathbb{S}_i$(k) is defined as $\mathbb{S}_i(k) = \frac{1}{k}\sum_{j=1}^{k} s_i$(j).
- Otherwise, it is penalized.

In the other words, the selected action of LA_i is rewarded if 1) it is different from the selected actions of its neighbors and 2) this selection results in better s_i than the average of s_i resulted from the previous selections of this LA.

The APN4 algorithm with this newly defined local rule is referred to as **APN5** hereafter. The results of comparing APN5 to APN4 in terms of CN, $\mathcal{U}$, and $\mathbb{S}(\mathbb{k})$ criteria are summarized in TABLE 2. These results indicate that APN5 outperforms APN4 in terms of all the mentioned criteria. This efficiency is the result of considering the proposed evaluating measure s_i in the local rule of the ICLA-layer.

TABLE 2: results of experiment for APN5 applied in APN-ICLA-VC

	APN4			APN5		
Graph	CN	$\mathcal{U}$	$S(\mathbb{k})$	CN	$\mathcal{U}$	$S(\mathbb{k})$
DSJC125_1	5	26876	.842	5	12593	.897
DSJC250_5	17	39055	.827	17	27263	.851
le450_15a	15	107575	.892	15	78335	.910
le450_15b	15	138001	.798	15	59340	.864
le450_15c	15	265139	.815	15	102384	.849
le450_15d	15	203358	.809	15	11965	.831
le450_25c	25	255099	.825	25	103542	.830
le450_25d	25	275394	.786	25	157722	.798

5. CONCLUSION

In this paper, adaptive Petri net based on irregular cellular learning automata (APN-ICLA) was proposed. This is a two-layered model, which consists of a PN-layer and an ICLA-layer and they both are constructed according to the problem to be solved. The PN-layer is a Petri net, in which the conflicting transitions are partitioned into several clusters. The second layer of the APN-ICLA is an irregular cellular learning automata. A cluster in the PN-layer is mapped into a cell in the ICLA-layer such that the learning automaton, which resides in a particular cell in the ICLA-layer, resolves the conflicts among the transitions in the corresponding cluster in the PN-layer. To show the applicability of the proposed APN-ICLA, it is utilized for solving the Vertex Coloring (VC) problem. First, an adaptive Petri net called APN-ICLA-VC was constructed from APN-ICLA. Then, based on APN-ICLA-VC, several algorithms were proposed to solve the VC problem. Using the theoretical analysis of the APN-ICLA-VC, an evaluative measure was provided for comparing these algorithms. This measure considers the following two points: 1) The probability of selecting an allowable color and 2) The required number of colors. According to this measure, a number of computer simulations were conducted and the algorithms were compared. The results of these simulations indicated that by considering the proposed evaluative measure in designing an algorithm, graphs can be colored faster and with fewer numbers of colors.

References

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

[2] S. M. Vahidipour, M. R. Meybodi, and M. Esnaashari, "Learning Automata Based Adaptive Petri net and Its Application to Priority Assignment in Queuing Systems with Unknown Parameters", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 45, no. 10. pp. 1373-1384, 2015.

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

[4] H. D. Burkhard, "Ordered firing in Petri nets", *EIK (Journal of information processing and cybernetics*, vol. 17, No. 2/3, pp. 71-86, 1981.

[5] F. Bause, “On the analysis of Petri net with static priorities”, *Acta Informatica*, Vol. 33, pp. 669-685, 1996.

[6] F. Bause, “Analysis of Petri nets with a dynamic priority method”, *Application and Theory of Petri Nets*, pp.215-234, 1997.

[7] L. E. Holloway and B. H. Krogh, "Controlled Petri nets: A tutorial survey", *the11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*, Springer Berlin Heidelberg, pp. 158-168, 1994.

[8] M. Esnaashari and M. R. Meybodi, “Irregular Cellular Learning Automata", *IEEE Transactions on Cybernetics*, vol. 45, no. 8, pp. 1622-1632, 2015.

[9] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*, Prentice Hall, 1989.

[10] M. A. L. Thathachar and P. S. Satstry, "A Hierarchical System of Learning Automata That Can Learn The Globally Optimal Path," *Information Sciences*, Vol. 42, No. 2, pp. 743-166, 1997.

[11] H. Beigy and M. R. Meybodi, "Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problem", *International Journal of Uncertainty, Fuzziness and Knowledge- based Systems*, Vol. 14, No. 5, pp. 591-617, 2006.

[12] M. R. Mollakhalili Meybodi and M. R. Meybodi, "Extended Distributed Learning Automata: An Automata-based Framework for Solving Stochastic Graph Optimization Problems", *Applied Intelligence*, vol. 41, vo. 3, pp. 923-940, 2014.

[13] M. R. Meybodi, H. Beigy, and M. Taherkhani, "Cellular Learning Automata and its Applications", *Sharif Journal of Science and Technology*, Vol. 19, No. 25, pp. 54–77, 2003.

[14] H. Beigy and M. R. Meybodi, "A Mathematical Framework for Cellular Learning Automata", *Journal of Advances in Complex Sys*tems, Vol. 7, No. 3, pp. 295–320, 2004.

[15] M. Hack, *Petri net languages*, C.S.G. Memo 124, Project MAC, M.I.T., 1975.

[16] B. H. Krogh, J. Magott, and L. E. Holloway, "On the complexity of forbidden state problems for controlled marked graphs", *the 30th IEEE Conference on Decision and Control*,1991.

[17] Y. Li and W. M. Wonham, "Control of vector discrete-event systems. II. Controller synthesis", *IEEE Transactions on Automatic Control*, vol. 39, no. 3, pp.512-531, 1994.

[18] T. Murata, "Petri nets: properties, analysis and applications", *Proc. IEEE*, vo1.77, pp.541-580, 1989.

[19] J. A. Torkestani and M. R. Meybodi, "Graph Coloring Problem Based on Learning Automata", *Proceedings of International Conference on Information Management and Engineering (ICIME 2009)*, Kuala Lumpur, Malaysia, pp. 718-722, 2009.

[20] J. A. Torkestani, “A new Approch to the vertex coloring Problem”, *Cybernetics and Systems*, vol. 44, no. 5, pp. 444-466, 2013.

[21] Ftp://dimacs.rutgers.edu/pub/challenge/graph/.

[22] J. Zhang, C. Wang, and M. C. Zhou, "Last-Position Elimination-Based Learning Automata", *IEEE Transactions on Cybernetics* , vol.44, no.12, pp.2484,2492, Dec. 2014.