Cellular Adaptive Petri Net Based on Learning Automata and Its Application to Vertex Coloring Problem

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

Abstract— In a Petri net, a decision point is raised when two or more transitions are simultaneously enabled in a given marking. The decision to be made at such a point is the selection of an enabled transition for firing. Decision making in Petri nets is accomplished by so called controlling mechanisms. Whenever a Petri net is used to represent an algorithm, applying different controlling mechanisms results in different instances of that algorithm. Recently, an adaptive controlling mechanism has been introduced for Petri nets in which several learning automata are used as decision makers during the evolution of the Petri nets. The Petri net with this adaptive controlling mechanism is referred to as APN-LA. Using APN-LA, one may be able to design adaptive algorithms to solve problems. There are problems for which designing a single APN-LA is a tedious work and results in a large and complex model. One class of such problems is the cellular problem, in which an identical algorithm must be executed in each cell and the solution to the problem is achieved from the cooperation of such identical algorithms. To avoid having large and complex APN-LAs for cellular problems, in this paper, cellular adaptive Petri net, called CAPN-LA, is proposed, which consists of a cellular structure and a number of identical APN-LAs. In CAPN-LA, each APN-LA represents the algorithm which must be executed in each cell and the required cooperation between the neighboring cells will be handled by means of cooperation between the APN-LAs in those cells. The notation of expediency for this model is defined and using the steady-state analysis of the behavior of the CAPN-LA model, conditions under which, this model is expedient are stated. A measure of expediency is defined for comparing different CAPN-LAs according to their expected reward. A CAPN-LA which receives higher expected reward is regarded as more expedient. The proposed CAPN-LA is then used to design different algorithms for the classic problem of vertex coloring. The measure of expediency is calculated for these algorithms and results of using them for coloring vertices of different graphs are also included.

Index Terms—Adaptive Petri net, Learning Automata, Cellular Algorithm, graph-based problems, Vertex Coloring (VC) Problem.

I. INTRODUCTION

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

Petri nets can also be used to represent algorithms of different classes [2][3]. One class of algorithms is the cellular algorithms [4] which commonly follows the distributed computing model [5]. This model is appropriate for representing cellular algorithms because it is made up of cells, like points in a lattice or like squares in a checker board, with local interactions. Each cell executes the same algorithm, referred to as the local algorithm, and the whole problem is solved by a means of cooperation between neighboring cells. Petri nets can be used to represent such cellular algorithms as well. For this to be made possible, two steps can be taken:

1) A Petri net must be designed for representing the local algorithm to be executed in each cell. This Petri net is then replicated into all cells.

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, Tehran, Iran.
M. Esnaashari is with the Information Technology Department, ITRC, Tehran, Iran.

2) A mechanism must be devised to handle the required cooperation between any two neighboring cells.

In the first step, when a Petri nets is designed for the local algorithm, a local solution is formed. This solution entails a number of decision points, at which there needs to be mechanisms for making decisions. Each decision point in a Petri net is aroused when two or more transitions are enabled in a given marking. At such point, the decision to be made is the selection of an enabled transition for firing among the set of enabled transitions. In Petri net literature, such decision making mechanisms are referred to as controlling mechanisms [6]. Different controlling mechanisms have been proposed in literature so far such as random mechanism [7], queue regimes [8], priority [9][10], external controller [11], and APN-LA [3]. Having a local solution, any of these controlling mechanisms may result in a specific local algorithm.

For the cooperation mechanism between any two neighboring cells, which is required in the second step above, a common way which usually is adopted in the Petri net literature, is representing the cooperation within the structure of the two Petri nets, designed for the cells; that is by connecting the two Petri nets using ordinary elements of Petri nets such as common places and inhibitor arcs [1]. This results in a large Petri net for the whole algorithm, which is extremely hard to analyze and simulate [12][13].

To overcome the issue of large Petri nets, in this paper, we suggest that the cooperation between the two neighboring cells to be handled within the controlling mechanisms of their Petri nets, instead of their structures. This can be possible only if the controlling mechanisms are capable of cooperating with each other. To the best of our knowledge, among the controlling mechanisms proposed for Petri nets so far, the APN-LA mechanism is the only one which includes such property. This is due to the fact that in this mechanism, controlling is handled using a Learning Automaton (LA), and LAs have been shown to be able to cooperate with each other [14]. Therefore, in this paper, a cellular adaptive Petri net, called CAPN-LA, in which a number of APN-LAs are organized into a cellular structure, has been proposed. This cellular structure represents a neighborhood relationship between the APN-LAs. Each APN-LA represents a local algorithm to be executed within a cell and the cooperation between neighboring cells is handled by the cooperation between the LAs that reside in the Petri nets of the cells.

For the proposed CAPN-LA, the concept of expediency is introduced. Expediency is a notion of learning. Any model that is said to learn must then do at least better than its equivalent pure-chance model. Informally, a CAPN-LA is considered to be expedient if, in the long run, the local algorithm provided by each of its APN-LAs outperforms a local algorithm which is provided by an equivalent APN-Pure-Chance-Automaton (APN-PCA). The equivalent APN-PCA of an APN-LA can be formed by replacing the LA in that APN-LA with a pure-chance automaton; that is, in APN-PCA, the controlling mechanism randomly selects an enabled transition for firing among the set of enabled transitions. The steady-state behavior of CAPN-LA is studied and then, conditions under which a CAPN-LA becomes expedient are presented.

Finally, CAPN-LA has been utilized to present different algorithms to solve vertex coloring (VC) problem [15]. To this end, first, an APN-LA has been designed for representing a local algorithm to be executed in any vertex of the graph. This APN-LA is then replicated into all vertices. Next, a CAPN-LA is constructed, in which the controlling mechanism handles the required cooperation between APN-LAs. The controlling mechanism of each APN-LA has a number of controlling parameters; thus, using different sets of parameters in APN-LAs of the CAPN-LA results in different algorithms for solving VC problem. In this paper, we use 6 different sets of controlling parameters for APN-LAs to form 6 different algorithms for solving VC problem. To be able to compare the results of these algorithms to each other, a measure is proposed based on the concept of expediency in CAPN-LA. A number of simulation studies are conducted in which the results of the presented algorithms are compared to each other in terms of the proposed measure of expediency.

The rest of the paper is organized as follows: Section II represents the basic definitions. Section III reviews the subject of related work. In Section IV, the proposed CAPN-LA is described. In Section V, the behavior of CAPN-LA is analyzed. In Section 0, application of the CAPN-LA to the vertex coloring problem is provided. Section VII includes the conclusion.

II. Learning Automata and Adaptive Petri Net

In this Section, first, a brief review of learning automata and Petri nets is presented. Then, adaptive Petri net based on learning automata is shortly described.

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 [14]. An action is chosen randomly 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 signal from the environment. Environment is shown by a triple $E = \{\underline{\alpha}, \underline{\beta}, \underline{c}\}$, 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, varying with 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 [14]. In what follows, 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

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

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

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

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} 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 the time [16].

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 other words, LA can be used as the building blocks of more complex learning models. These complex models include hierarchical system of LA [17], distributed LA (DLA) [18], extended DLA [19], network of LA [20], multi-level game of LA [21][22], cellular LA (CLA)[23], CLA-based evolutionary computing [24], differential evolution-based CLA [25], CLA-based particle swarm optimization [26], and Irregular CLA [27].

2) *Petri nets*

A Petri net is one of several mathematical modeling languages for describing the distributed systems. An ordinary Petri net (PN) is a triple (P, T, W), where P is a non-empty finite set of places, T is a finite set of transitions, and $W: \big((P \times T) \cup (T \times P)\big) \rightarrow \mathbb{N}$ 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 $x\bullet = \{y \in P \cup T | W(x,y) > 0\}$.
- A marking of a PN is a function $M: P \rightarrow \mathbb{N}$ 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 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, given a marking, 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 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 satisfying $M_0 \in [M_0 >$ and if $M' \in [M_0 >, M'[t > M''$ for some $t \in T$ then $M'' \in [M_0 >$.

A set of firing rules in a Petri net determines the evolution of the Petri net from the current marking of the system to a new marking [6][7]. In other words, this set 1) determines the set of enabled transitions from available transitions; 2) selects a transition from the set of enabled transitions for firing; and, 3) generates a new marking by firing the selected transition. The second step, within the above steps, is the one where a decision making needs to be performed for selecting an enabled transition for firing. This is where controlling mechanism is used.

A controlling mechanism must be able to select an enabled transition for firing in any of the following two situations: 1) All enabled transitions are concurrent and they are causally independent, which means one transition may fire before, after or in parallel with the others [7] and 2) some of the enabled transitions are in conflict, which means if any of them is fired, none of the others can be fired [7]. In this paper, the mechanisms of selecting an enabled transition among a set of concurrent transitions are referred to as *concurrent selectors* and the other mechanisms are referred to as *conflict resolvers*, hereafter.

3) *Adaptive Petri net based on Learning Automata*

An adaptive Petri net based on learning automata (APN-LA) is a Petri net obtained from the fusion between Petri nets and LAs [3]. In this Petri net, an LA is used as the conflict resolver of the controlling mechanism and a random mechanism is used as the concurrent selector. The LA is assigned to a cluster of transitions to resolve conflicts among them. To form such clusters, we consider the transitions with the same input places as the members of a cluster. In such a way, the conflicting transitions form one or more clusters. The remaining transitions, which are not in conflict with any other transitions, form another cluster, in which the concurrent selector, or in other words, the random mechanism, is used to select an enabled transition for firing.

Like other Petri nets, APN-LAs can be designed to represent an algorithm for solving a problem. To this end, a construction procedure is described by the following steps [3]:

1. An ordinary Petri net for modeling the existing problem is constructed.
2. The transitions in this Petri net are divided into $n+1$ clusters $s_0, s_1, \cdots, s_n$. A cluster $s_i, (i = 1, \dots, n)$ is formed by clustering all transitions with the following property: for each transition t_k in the cluster, there exists at least one transition $t_j \neq t_k$ in the cluster, such that t_k and t_j have at least one input place in common and also two transitions in two different clusters do not have any input place in common. Also note that in any PN, two transitions are in conflict if they have at least one input place in common. The remaining transitions in the Petri net, which are not in conflict with any other transitions, form the cluster s_0.
3. For each cluster $s_i, (i = 1, \dots, n)$, a transition with one input and one output place will be inserted into the Petri net. These extra elements are used for providing a way of fusion between Petri nets and learning automata. Inserted transitions are immediate transitions, except that, when they fire, the internal structure of the LAs is updated. We refer to such transitions as updating transitions. Updating transitions are added to the cluster s_0. More details on this construction procedure is described in [3].

Firing rules of an APN-LA system differs from those of an ordinary PN due to the fusion with LAs [3]. The set of firing rules of an APN-LA, in a given marking, can be briefly described as follows[1]:

1- Determine the set of enabled transitions.
2- Specify a firing cluster among the set of enabled clusters; a cluster of transitions is enabled in a given marking when at least one of its transitions is enabled. This selection is performed randomly.
3- If the firing cluster is the cluster s_0, then the concurrent selector is used to select a transition in s_0 for firing.
4- Otherwise, the conflict resolver of the firing cluster is used to select a transition in that cluster for firing.
5- The selected transition is fired and a new marking is generated.
6- If the fired transition is an updating transition, then the internal state of the related conflict resolver is updated.

III. Related Work

This section consists of three parts. The first part briefly reviews cellular applications of Petri nets. The controlling mechanisms, used in Petri nets so far, are shortly described in the second part. Finally, the third part provides an overview on the learning automata-based algorithms, designed for solving vertex coloring problem.

1) *PN and cellular applications*

Petri nets are suitable for modeling cellular applications [28]. In [29], PNs are used to model and represent the dynamic behavior of a cell's operations in a Cellular Manufacturing System (CMS). In each cell of CMS,

[1] More details on the firing rules of the APN-LA are given in [3].

a cell control system is required to determine the cell's operations, as well as handling machine breakdowns in a real time manner. Such a control system can be reliably modeled by designing a Petri net for the CMS. Other applications of Petri nets for CMS are reported in literature [30][31][32].

PNs can be used to model different strategies in cellular networks such as channel assignment. For instance, in [33], PNs are used to model a centralized Dynamic Channel-Assignment (DCA), as well as a distributed DCA (DDCA) strategy. A kind of handshaking mechanism is embedded in the PN so that no two cells within the channel reuse distance can use the same channel. PNs are also used in resource management of cellular networks [33], bandwidth allocation in IEEE 802.16 networks [35], handoff process in cellular mobile telephone systems [36], and fairness analysis in cellular networks [37].

Petri nets are widely used in other fields of computer networks. In [38], an extended PN is introduced as a formal verification technique of IEEE 802.11 MAC protocol. This extended PN is applied as a case study to the centralized control mechanism of IEEE 802.11, which is used to support delay sensitive streams and fading data traffics. In [39], a colored PN is used to model and analyze the station to station (STS) protocol in computer networks. In [40], a PN is used to model the Session Initiation Protocol (SIP) and to accomplish its verification. Another interesting work is presented in [41] where the 802.11i 4-way handshaking mechanism is analyzed utilizing high-level PNs. The results of the analysis confirmed that this mechanism is vulnerable to the Denial-of-Service attack.

Several usages of Petri nets have also been reported in the field of wireless sensor networks (WSNs). They have been used to model the sleep/wake mechanism in WSNs [42][43], study the performance of data management schemes [44], evaluate the energy consumption of a wireless sensor node [45], analyze the behavior of a routing protocol [46], evaluate a power consumption strategy [47], and analyze a localization scheme [48] for WSNs.

2) The controlling mechanisms in Petri nets

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 [6] 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 [8]. 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 [9][10]. The priority net can be either static [9] or dynamic [10]. 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 places, which is called control places [11]. 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 other words, so that the CtlPN never reaches a set of forbidden markings. General methods [11] such as path-based approaches and linear integer programming can be used to design this type of control policy.

A controlling mechanism has been addressed in [3], 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. For instance, in [3], 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 to provide 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) *Vertex coloring algorithms based on learning automata*

The problem of Vertex Coloring (VC) is a variation of the Graph Coloring (GC) problem [15][49]. The problem of Graph Coloring is a classical problem of combinatorial optimization in the graph theory. The graph coloring is broadly used in real life applications such as computer air traffic flow management [51], timetabling [52][53], and light wave lengths assignment in optical networks [54].

In Vertex Coloring problem, each vertex of the graph is assigned a color in a way that there are no two adjacent vertices with the same color [49]. Since the vertex coloring problem is NP-hard for the general graphs [55], the exact algorithms can merely be used for the small graphs, while very large graphs usually come up in a wide range of applications [49]. On the other hand, in the real life applications, a near optimal coloring of the graph will usually suffice. Therefore, a host of algorithms with polynomial time approximation has been introduced to find a near optimal solution for the coloring problem [15]. The approximation methods, which have been addressed in the literature can be categorized as genetic algorithms [56], fuzzy-based optimizations [57], evolutionary algorithms [58], neural network approaches [59], etc.

A number of learning automata-based algorithms for solving the VC problem have been addressed in the literature. In a major number of these algorithms, as the process of learning reaches to the end, each vertex learns how to select a color in order to color the graph with the smallest number of colors, with no neighbors with the same color [49]. For example, an approximation algorithm has been introduced in [60], in which an LA is assigned to each vertex. The number of actions of this LA is equal to the number of non-neighbor vertices of the vertex. Each iteration of the algorithm consists of a few stages. In the beginning, the learning automata select a subset of the non-neighbor graph vertices and assign them the same color. Then, the rest of the uncolored vertices will be selected and colored. An iteration is considered completed if the entire vertices are colored. Here, if the size of the color-set (the number of the selected colors) is smaller than the number of the selected colors, which were chosen earlier, the coloring is rewarded; otherwise it is penalized. When the algorithm is reaching to the end, each LA learns how to select one of the non-neighbor vertices, so that the number of selected colors is minimized. The other LA applications in the graph coloring have been discussed in the literature [61][62].

There are a number of algorithms for the problem of vertex coloring in which complex models are used based on LA. For example, in order to solve the problem of vertex coloring, an irregular cellular learning automata ICLA-base algorithm has been addressed in [63]. In this algorithm, the graphs are modeled by an ICLA to be colored. Each vertex in the graph corresponds to an ICLA cell. Then each cell will be equipped with an LA. The activation of the LA is automatic. When activated, a color will be selected for the assigned cell. For a proper selection of colors by an LA, the LAs can cooperate with the neighboring LAs, which are located in the neighboring cells. If the number of the colors in a coloring is larger than the number of the best coloring found so far, or if the coloring is not allowed, the selected coloring will be penalized. If none of these two conditions occur, the selected coloring will be rewarded. When the algorithm is reaching to its end, the LAs learn how to choose the colors in a way that the graph is entirely colored in a legal way with the smallest number of colors. The other ICLA applications in the problem of Graph Coloring are addressed in the literature [64][65].

IV. The Proposed Cellular Adaptive PN based on LA

In this section, a Cellular Adaptive Petri net based on Learning Automata, called CAPN-LA, will be proposed. A CAPN-LA consists of a number of identical APN-LAs organized into a cellular structure. This cellular structure consists of a number of cells with local neighborhood relationship, and an APN-LA is assigned to each cell (Figure 1). The neighborhood relationship between cells in the cellular structure represents the neighborhood relationship between APN-LAs. Each APN-LA represents a local algorithm to be executed within a cell of the cellular structure, and the cooperation among these local algorithms specifies the behavior of the CAPN-LA.

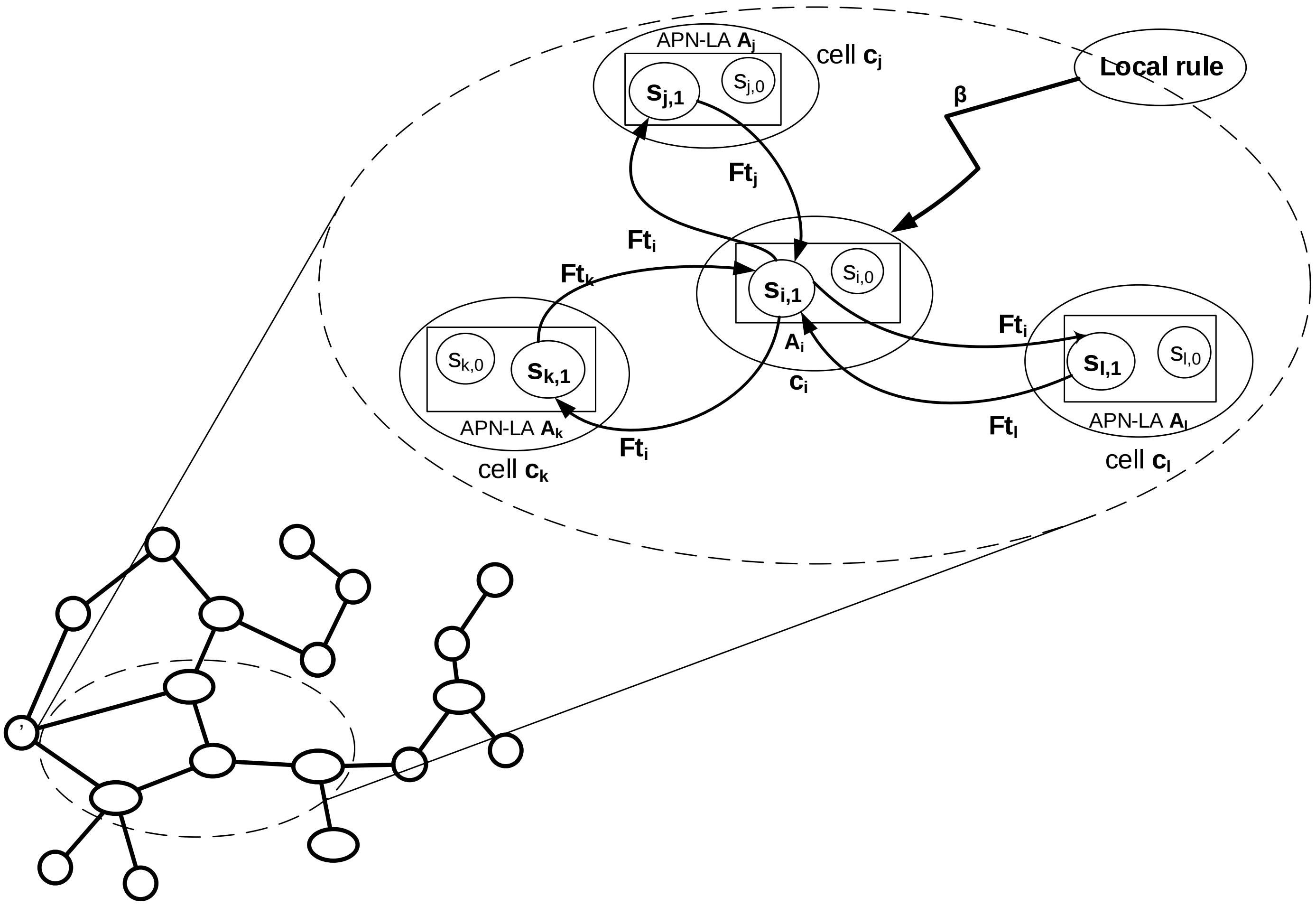

Figure 1: The Cellular Adaptive Petri net based on Learning Automata (CAPN-LA)

The APN-LA assigned to the cell c_i, is denoted by A_i. Like other Petri nets, each A_i has a controlling mechanism with two parts: 1) a concurrent selector and 2) a conflict resolver. The concurrent selector in APN-LA is a random mechanism and its conflict resolver is a learning automaton (LA) with varying number of actions. This LA, denoted by LA_i, is assigned to a cluster of transitions, i.e. $s_{i,1}$, which may conflict with each other in some markings. For such a marking, LA_i selects an enabled transition, denoted by Ft_i, among the enabled transitions in $s_{i,1}$ for firing.

Two clusters $s_{i,1}$ and $s_{j,1}$ in the CAPN-LA are neighbors, if their corresponding APN-LAs, A_i and A_j are neighbors. The neighboring clusters of any particular cluster, indicated by $s_{i,1}$, constitute the local environment of the LA in that cluster, i.e. LA_i, which provides the reinforcement signal to LA_i. This signal is generated by a local rule which captures the last firing transitions of the cluster $s_{i,1}$ and its neighboring clusters as the input and generates the reinforcement signal for LA_i as the output. The local rule provides a means of cooperation between neighboring clusters of conflicting transitions, which in turn, results in cooperation between neighboring APN-LAs. This indeed provides a means of cooperation between local algorithms represented by neighboring APN-LAs.

Formally, a CAPN-LA structure can be defined according to Definition 1, given below.

Definition 1: A cellular Adaptive Petri net based on Learning Automata (CAPN-LA) is a structure $\mathcal{N} = (G < \mathbb{E}, V >, \{A_1, \ldots, A_n\}, \hat{F})$, *where*

a. G is an undirected graph, with V as the set of vertices (cells) and $\mathbb{E}$ as the set of edges (adjacency relations).

b. $\{\mathrm{A}_1, \dots, \mathrm{A}_\mathrm{n}\}$ is a finite set of APN-LAs where $n = |V|$. Each A_i is assigned to a cell c_i, and is defined by a 5-tuple $A_i = (\widehat{P_\iota}, \widehat{T_\iota}, \widehat{W_\iota}, S_i, LA_i)$ in which $\widehat{P_\iota}$ is the set of places, $\widehat{T_\iota} = T \cup \{t_{i,1}^u\}$ is a finite set of transitions and an updating transitions $t_{i,1}^u$, $\widehat{W_\iota}: \left(\left(\widehat{P_\iota} \times \widehat{T_\iota}\right) \cup \left(\widehat{T_\iota} \times \widehat{P_\iota}\right)\right) \to \mathbb{N}$ defines the interconnection of $\widehat{P_\iota}$ and $\widehat{T_\iota}$, LA_i is a learning automaton with varying number of actions, and $S_i = \{s_{i,0}, s_{i,1}\}$ is a set of clusters of transitions where

   i. $s_{i,0}$ contains the concurrent transitions from $\widehat{T_\iota}$. No pair of transitions in this cluster have any input places in common, i.e. $\{\forall t_k \in s_{i,0}, \forall t_j \in s_{i,0} | \bullet \mathrm{t_k} \cap \bullet \mathrm{t_j} = \emptyset\}$. A random mechanism is used as the concurrent selector in this cluster.

   ii. $s_{i,1}$ contains the remaining transitions in $\widehat{T_\iota}$ which are not in $s_{i,0}$. The set of transitions in the cluster $s_{i,1}$ may conflict with each other in some markings, and thus a conflict resolver is required to select an enabled transition for firing in this cluster.

c. $\hat{F} = \{F_1, \dots, F_n\}$ is a set of local rules which implies the cooperation among clusters of conflicting transitions. $F_i: \underline{s}_{i,1} \to \beta_i$ is the local rule of the cell c_i, where $\underline{s}_{i,1} = \left\{s_{j,1} \middle| (i,j) \in E\right\} \cup \{s_{i,1}\}$ is the set of conflicting transitions of all neighbors of A_i, and β_i is the reinforcement signal related to LA_i. Upon the generation of β_i, LA_i updates its action probability vector using its learning algorithm.

Definition 2: a CAPN-LA system is a 2-tuple $(\mathcal{N}, M_0)$, where

- $\mathcal{N}$ is a CAPN-LA structure,
- $M_0 = [\mu_{1,0}, \dots, \mu_{n,0}]$ is the initial marking vector of $\mathcal{N}$, such that $\mu_{i,0}$ is the initial marking of A_i. M is used to represent the set of markings of $\mathcal{N}$ and μ_i to represent the set of markings of A_i; $\mu_{i,j}$ represents the j^{th} marking in A_i. To represent an arbitrary marking in the APN-LA, the notation μ will be used.

A CAPN-LA system evolves from the current marking vector to a new marking vector according to the firing rules of it constituting APN-LAs. The CAPN-LA is asynchronous and hence, each of its APN-LAs evolves independently from the others. The firing rules of the APN-LA A_i, assigned to the cell c_i, in any marking M can be described as follows:

1. A transition $t \in \widehat{T_\iota}$ is said to be enabled if each input place p of t is marked with at least $\widehat{W_\iota}(p,t)$ tokens.
2. Only one enabled transition $t \in \widehat{T_\iota}$ can fire in each marking M. t is selected form the fired cluster.
3. The cluster $s_{i,0}$(or $s_{i,1}$) is selected as the fired cluster according to probability $P_{i,0}$(or $P_{i,1}$). Two probability values $P_{i,0}$ and $P_{i,1}$ are calculated based on the number of enabled transitions in $s_{i,0}$ and $s_{i,1}$; that is $P_{i,0} = \frac{|t' \in s_{\mathrm{i},0}, M[t'>|}{|t \in \widehat{T_\iota}, M[t>|}$ and $P_{i,1} = \frac{|t' \in s_{\mathrm{i},1}, M[t'>|}{|t \in \widehat{T_\iota}, M[t>|}$, where $|\quad|$ stands for norm operator and $P_{i,0} + P_{i,1} = 1$.
4. If $s_{i,0}$ is the fired cluster, then an enabled transition t is randomly selected from $s_{i,0}$ for firing (concurrent selector mechanism); Otherwise, LA_i is activated. The available action set of the activated LA_i consists of the actions corresponding to the enabled transitions. 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.
5. Firing of a transition $t \in \widehat{T_\iota}$ removes $\widehat{W_\iota}(p,t)$ tokens from each input place p of t, and adds $\widehat{W_\iota}(t,p)$ tokens to each output place p of t.

6. The local rule $F_i \in \hat{F}$ is used to generate the reinforcement signal β_i for LA_i when the updating transition $t_{i,1}^u$ fires. For generating the reinforcement signal, the last selected actions (the last fired transitions from $s_{i,1}$) of neighboring APN-LAs are used.
7. Using β_i, LA_i updates its action probability vector using its learning algorithm.

V. BEHAVIOR OF CAPN-LA

In this section, the steady-state behavior of the proposed CAPN-LA model will be analyzed. Before that, some definitions will be provided, which will be used later for the analysis of the CAPN-LA.

Definition 3: At time instant k, the conflicting transitions configuration in the CAPN-LA is denoted by $\underline{q}(k) = \left(\underline{q}_1, \underline{q}_2, \ldots, \underline{q}_n\right)^T$ where $\underline{q}_i = \left(q_i^1, \ldots, q_i^{m_i}\right)^T$ is the selection probability of transitions in the cluster $s_{i,1}$ and T denotes the transpose operator.

Definition 4: The set of probabilistic configurations $\mathcal{K}$ in CAPN-LA, is

$$\mathcal{K} = \left\{\underline{q} \middle| \underline{q} = \left(\underline{q}_1, \ldots, \underline{q}_n\right)^T, \underline{q}_i = \left(q_i^1, \ldots, q_i^{m_i}\right)^T, \forall y, \forall i, 0 \le q_i^y \le 1, \textstyle\sum_y q_i^y = 1\right\} \quad (3)$$

Definition 5: The conflicting transitions evolution from a given initial configuration $\underline{q}(0) \in \mathcal{K}$ is a sequence of configurations $\{\underline{q}(k)\}_{k\ge 0}$, such that $\underline{q}(k+1) = \mathcal{G}(\underline{q}(k))$, where $\mathcal{G}$ is a mapping $\mathcal{G} : \mathcal{K} \to \mathcal{K}$. Whenever an updating transition is fired, this mapping will be performed.

Definition 6: A pure-chance automaton is an automaton that chooses each of its actions with equal probability. Therefore, for a pure-chance automaton with m_i-actions $q_i^j, j = 1, \ldots, m_i$ is equal to $\frac{1}{m_i}$.

Definition 7: An APN-PCA, is an APN-LA in which instead of a learning automaton, the conflicts among transitions are resolved by a pure-chance automaton.

Definition 8: A CAPN-PCA is a CAPN-LA, in which APN-PCAs are utilized instead of APN-LAs.

Definition 9: The neighboring set of any cluster $s_{i,1}$, denoted by N_i, is defined as the set of all clusters with conflicting transitions in the neighborhood cells of the cellc_i, that is,

$$N_i = \{s_{j,1} | (i,j) \in \mathbb{E}\} \quad (4)$$

Let $\mathcal{N}_i$ be the cardinality of N_i. In addition, assume the following notations:

- *$N_i^j, (1 \le j \le \mathcal{N}_i)$ is the j^{th} neighboring cluster of the cluster $s_{i,1}$,*
- *Ft_i denotes the last fired transition of the cluster $s_{i,1}$,*
- *$ED(\mu, s_{i,1})$ indicates the number of enabled transitions within $s_{i,1}$ in marking μ,*
- *$\pi_\mu(k)$ represents the probability of the APN-LA A_i being in a marking μ at time instant k,*
- *$\mu_{i,l} \in \mu_i$ is the l^{th} marking from the marking set μ_i of the APN-LA A_i,*
- *t_i^j indicates the j^{th} transition in the cluster $s_{i,1}$,*
- *$q_i^j(k)$ is the probability of choosing the transition t_i^j from the cluster $s_{i,1}$ for firing at time instant k,*
- *En(t, μ) denotes a Boolean function, in which it is indicated whether or not the transition t is enabled*

in marking μ, and it is defined in equation (5), given as below,

$$En(t, \mu) = \begin{cases} 1, & if\ \mu[t > \\ 0, & otherwise \end{cases}. \tag{5}$$

- *$\hat{q}_i^j(k,\mu)$ is the scaled probability of choosing the enabled transition t_i^j for firing in marking μ. $\hat{q}_i^j(k,\mu)$ is calculated, when transition t_i^j is enabled in marking μ, according to the equation (6). $\hat{q}_i^j(k,\mu) = 0$ when transition t_i^j is not enabled in marking μ.*

$$\hat{q}_i^j(k,\mu) = \frac{q_i^j(k)}{\sum_{t_i^l | En(t_i^l,\mu)} q_i^l(k)} \tag{6}$$

Definition 10: The average reward of firing the transition t_i^j at time instant k is defined as

$$d_i^j\left(\underline{q}(k)\right) = \sum_{t_1 \in N_i^1, \dots, t_{\mathcal{N}_i} \in N_i^{\mathcal{N}_i}} F_i(t_1, \dots, t_{\mathcal{N}_i}; t_i^j) \prod_{t_l \in t_1, \dots, t_{\mathcal{N}_i}, t_l \in s_{L,1}} q_L^{t_l} \tag{7}$$

and the average reward for the APN-LA A_i at time instant k is defined as

$$D_i\left(\underline{q}(k)\right) = \sum_{\mu \in \mu_i} \left(\sum_{t_i^j} d_i^j\left(\underline{q}(k)\right) \times \hat{q}_i^j(k,\mu) \right) \times \pi_\mu(k) \tag{8}$$

In equation (7), $F_i(t_1, \dots, t_{\mathcal{N}_i}; t_i^j)$, the local rule of the cell c_i, generates the reinforcement signal $\beta_i(k)$ for a set of transitions $\{t_1, \dots, t_{\mathcal{N}_i}; t_i^j\}$; This set consists of transitions $t_l, 1 \le l \le \mathcal{N}_i, l \ne i$ from the neighboring clusters of $s_{i,1}$ as well as transition t_i^j from the cluster $s_{i,1}$. The average reward of transition t_i^j is calculated by equation (7), a weighted sum of F_i , where the weights are the products of the probabilities of firing transitions $\{t_1, \dots, t_{\mathcal{N}_i}\}$. The average reward for all transitions in the cluster $s_{i,1}$ is calculated for a marking set μ_i in equation (8). Note that by definition, $\hat{q}_i^j(k,\mu) \ne 0$ only when the transition t_i^j is enabled. In other words, enabled transitions are only considered for calculating the average reward in the cluster $s_{i,1}$.

It has been assumed that $d_i^j\left(\underline{q}\right) \ne 0$ for all i, j, and $\underline{q}$. In other words, in any configuration, any transition has a non-zero chance of receiving reward.

Definition 11: The total average reward for the CAPN-LA at configuration $\underline{q} \in \mathcal{K}$ is the sum of the average rewards for all APN-LAs in the CAPN-LA, that is

$$D\left(\underline{q}\right) = \sum_i D_i\left(\underline{q}\right) \tag{9}$$

and notation $D\left(\underline{q}^{PC}\right)$ is referred to as the total average reward for a CAPN-PCA.

Expediency is a notion of learning. Any model that is said to learn must then do at least better than its equivalent pure-chance model. In the rest of this section, we define the expediency concept for the CAPN-LA.

Definition 12: A CAPN-LA is assumed to be expedient with respect to the cluster $s_{i,1}$ if $\lim_{k\to\infty} \underline{q}(k) = \underline{q}^*$exists and (10) holds for every i where π_μ^* is the steady-state probability of marking μ.

$$\lim_{k\to\infty} E\left[D_i\left(\underline{q}(k)\right)\right] > \sum_{\mu\in\mu_i}\left[\frac{1}{ED(\mu, s_{i,1})} \times \sum_{t_i^j} d_i^j\left(\underline{q}^*\right)\right] \times \pi_\mu^* \qquad (10)$$

In other words, a CAPN-LA is expedient with respect to the APN-LA A_i if, in the long run, A_i performs better (receives more reward) than an APN-PCA.

Definition 13: A CAPN-LA is assumed to be expedient if it is expedient with respect to any of its constituting APN-LAs.

Theorem 1: A CAPN-LA, in which the learning automata with the SL_{R-P} learning algorithm are the conflict resolvers, regardless of the local rule being used, is expedient.

Proof: It has to be shown that CAPN-LA with SL_{R-P} learning automata is expedient with respect to all of its APN-LAs, that is, $\lim_{k\to\infty} \underline{q}(k) = \underline{q}^*$ exists and (11) holds for every i.

$$\lim_{k\to\infty} E\left[D_i\left(\underline{q}(k)\right)\right] > \sum_{\mu\in\mu_i}\frac{1}{ED(\mu, s_{i,1})} \times \left[\sum_{t_i^j} d_i^j\left(\underline{q}^*\right)\right] \times \pi_\mu^*, for\ every\ i \qquad (11)$$

Lemma 1: Consider a probabilistic configuration $\underline{q} \in \mathcal{K}$ which evolves with SL_{R-P} learning algorithm. Regardless of the initial configuration $\underline{q}(0)$, $\lim_{k\to\infty} \underline{q}(k)$ exists and $\lim_{k\to\infty} E\left[\underline{q}(k)\right] = \underline{q}^*$.

Proof: They both have been proven in [27], in which $\underline{q}(k)$ represents a probabilistic matrix constructed by the action probability vector of a team of LAs at time instant k. Theses LAs form an irregular cellular learning automata and all of them evolve with SL_{R-P} learning algorithm.

Lemma 2: Consider a probabilistic configuration $\underline{q} \in \mathcal{K}$ which evolves with SL_{R-P} learning algorithm. Let ${q_i^j}^*$be the selection probability of the transition t_i^j in the cluster $s_{i,1}$ in the long run and $d_i^j\left(\underline{q}^*\right)$ be the generated reward value by equation (7). Regardless of the local rule being used, the value of ${q_i^j}^*$ is proportional to the value of $d_i^j\left(\underline{q}^*\right)$

Proof: It has been shown in [27] that the value of ${q_i^j}^*$ is inversely proportional to the received penalty from the environment, denoted by $rp_i^j(\underline{q}^*)$, that is ${q_i^j}^* \propto \frac{1}{rp_i^j(\underline{q}^*)}$. Since the reward signal, $d_i^j\left(\underline{q}^*\right)$ is equal to $1 - rp_i^j(\underline{q}^*)$, we get

$${q_i^j}^* \propto \frac{1}{1 - d_i^j\left(\underline{q}^*\right)} \qquad (12)$$

In equation (12), if the value of $d_i^j\left(\underline{q}^*\right)$ is increased, then the denominator decreases and consequently, the $q_i^{j^*}$ value increases and vice versa. This is an indication of the fact that the $q_i^{j^*}$ value is proportional to the $d_i^j\left(\underline{q}^*\right)$ value and hence the lemma.

By taking the Lemma 1 into consideration, it is only required to be shown that the inequality (11) holds. By using the average reward definition in equation (8), the left hand side of (11) can be rewritten as follows:

$$\lim_{k\to\infty} E\left[D_i\left(\underline{q}(k)\right)\right] = \sum_{\mu\in\mu_i} \lim_{k\to\infty} E\left[\left(\sum_{t_i^j}\left[d_i^j\left(\underline{q}(k)\right)\times \hat{q}_i^j(k,\mu)\right]\right)\times \pi_\mu(k)\right] \tag{13}$$

Since $d_i^j\left(\underline{q}(k)\right)$, $\hat{q}_i^j(k)$, and $\pi_\mu(k)$ are independent, the above equation can be simplified to

$$\lim_{k\to\infty} E\left[D_i\left(\underline{q}(k)\right)\right] = \sum_{\mu\in\mu_i}\sum_{t_i^j}\left(\lim_{k\to\infty} E\left[d_i^j\left(\underline{q}(k)\right)\right]\times \lim_{k\to\infty} E\left[\hat{q}_i^j(k,\mu)\right]\right)\times \lim_{k\to\infty} E\left[\pi_\mu(k)\right] \tag{14}$$

Considering Lemma 1, $\lim_{k\to\infty} E\left[\underline{q}(k)\right] = \underline{q}^*$, and hence $\lim_{k\to\infty} E\left[\hat{q}_i^j(k)\right] = \hat{q}_i^{j^*}$ for all i and j. In addition, $\lim_{k\to\infty} E\left[\pi_\mu(k)\right] = \pi_\mu^*$. Using equation (7), $\lim_{k\to\infty} E\left[d_i^j\left(\underline{q}(k)\right)\right]$ can be computed as follows:

$$\begin{aligned}\lim_{k\to\infty} E\left[d_i^j\left(\underline{q}(k)\right)\right] &= \lim_{k\to\infty} E\left[\sum_{t_1\in N_i(1),\ldots,t_{\mathcal{N}_i}\in N_i(\mathcal{N}_i)} F_i(t_1,\ldots,t_{\mathcal{N}_i};t_i^j)\prod_{t_l\in t_1,\ldots,t_{\mathcal{N}_i},t_l\in s_{L,1}} q_L^{t_l}\right]\\ &= \sum_{t_1\in N_i(1),\ldots,t_{\mathcal{N}_i}\in N_i(\mathcal{N}_i)} F_i(t_1,\ldots,t_{\mathcal{N}_i};t_i^j)\prod_{t_l\in t_1,\ldots,t_{\mathcal{N}_i},t_l\in s_{L,1}} \lim_{k\to\infty} E\left[q_L^{t_l}\right]\\ &= d_i^j\left(\underline{q}^*\right)\end{aligned} \tag{15}$$

Thus,

$$\lim_{k\to\infty} E\left[D_i\left(\underline{q}(k)\right)\right] = \sum_{\mu\in\mu_i}\left(\sum_{t_i^j} d_i^j\left(\underline{q}^*\right)\times \hat{q}_i^{j^*}\right)\times \pi_\mu^* \tag{16}$$

Now, for every i, it needs to be shown that:

$$\sum_{\mu\in\mu_i}\left(\sum_{t_i^j} d_i^j\left(\underline{q}^*\right)\times \hat{q}_i^{j^*}\right)\times \pi_\mu^* > \sum_{\mu\in\mu_i}\left(\frac{1}{ED(\mu,s_{i,1})}\right)\times\sum_{t_i^j} d_i^j\left(\underline{q}^*\right)\times \pi_\mu^* \tag{17}$$

For every $\mu\in\mu_i$, the above equation can be simplified to

$$\sum_{t_i^j} d_i^j\left(\underline{q}^*\right)\times \hat{q}_i^{j^*} > \frac{1}{ED(\mu,s_{i,1})}\times\sum_{t_i^j} d_i^j\left(\underline{q}^*\right) \tag{18}$$

In the above inequality, each side is a convex combination of $d_i^j\left(\underline{q}^*\right)$. In the right hand side convex combination of (18), all of the $d_i^j\left(\underline{q}^*\right)$ have the same weights which equals to $\frac{1}{\mathrm{ED}(\mu,\mathrm{s}_{\mathrm{i},1})}$ whereas in the left hand side convex combination, the weight of each $d_i^j\left(\underline{q}^*\right)$ equals to $\hat{q}_i^{j^*}$.It was shown that in Lemma 2, the $q_i^{j^*}$value is proportional to the $d_i^j\left(\underline{q}^*\right)$ value and therefore, $\hat{q}_i^{j^*}$is also proportional to the $d_i^j\left(\underline{q}^*\right)$. This indicates that the greater the value of $d_i^j\left(\underline{q}^*\right)$, the greater its weight will be. As a result, the right hand side convex combination of the inequality (18) is smaller than the one on the left hand side, and thus the theorem is proved. ■

1) Norms of Behavior

To judge the behavior of the proposed CAPN-LA, it is necessary to set up quantitative norms of behavior. Here, we define a new metric, namely measure of expediency, in terms of the firing probabilities of the transitions in CAPN-LA.

Definition 14: Measure of expediency is defined according to equation (19). $M_E < 0$ indicates that the CAPN-LA performs worse than its equivalent CAPN-PCA, $M_E = 0$ indicates that the CAPN-LA is the pure-chance, and $M_E > 0$ indicates that the CAPN-LA performs better than its equivalent CAPN-PCA. A higher value for M_E means that the CAPN-LA is more expedient.

$$M_E = \left(\frac{\lim_{k\to\infty} E\left[D\left(\underline{q}(k)\right)\right]}{D\left(\underline{q}^{PC}\right)}\right) - 1 \qquad (19)$$

VI. An application of CAPN-LA

In this section, the proposed CAPN-LA is used to solve the vertex coloring (VC) problem in a graph $\mathbb{G}$. First, an APN-LA will be constructed for the problem of coloring a graph in a vertex using m colors, called APN-LA-VC, and then its behavior will be analyzed. Next, different algorithms will be represented for solving VC problem using the replication of the APN-LA-VC in all vertices of graph $\mathbb{G}$ (cells of CAPN-LA) which is to be colored. The yielded structure is denoted by CAPN-LA-VC. Finally, multiple simulation results will be given.

The model used for experiments is thoroughly described in Sections VI.2)A and VI.2)B. Since the CAPN-LA consists of a number of APN-LAs, we have first described these APN-LAs, referred to as APN-LA-VCs, in Section VI-2)A, and then defined the CAPN-LA-VC as a graph of APN-LA-VCs in Section VI-2)B. Input data, used for the experiments, is a subset of hard-to-color benchmarks reported in DIMACS [66]. All simulations have been implemented by MATLAB 8.1 and have been run on a Pentium V Core 2 Duo 2.0 GHz. Simulation parameters are set as follows: $a = b = .1$, i.e. the reward and penalty parameters, thresholds ε and τ are equal to $.009$ and $.95$ respectively, which are described in more details later in this section.

1) The VC Problem in a graph

Vertex coloring problem is a well-known coloring problem [15][49] in which a color is assigned to each vertex of the graph. A legal vertex coloring of graph $\mathbb{G} < V, \mathbb{E} >$, where $V(\mathbb{G})$ is a set of $|V| = n$ vertices and $\mathbb{E}(\mathbb{G})$ is an 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 color. A solution to VC problem can be modeled by a quadruple

$(V, \mathbb{E}, \mathbb{C}, H)$, where V denotes the vertex-set of graph $\mathbb{G}$, $\mathbb{E}$ 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 $H: V \rightarrow \mathbb{C}$ is the coloring function that assigns a color to each vertex such that $F(u) \neq F(v)$ for every $(u, v) \in \mathbb{E}$. Graph $\mathbb{G} < V, \mathbb{E} >$ is P-colorable, if it can be legally colored with at most P different colors. VC problem can be considered as a decision making problem that aims at deciding for a given graph whether or not the graph is P-colorable, and is called P-coloring problem. The chromatic number $\mathcal{X}(G)$ is the minimum number of colors required for coloring the graph, and a graph $\mathbb{G}$ is said to be P-chromatic, if $\mathcal{X}(\mathbb{G}) = P$. It has been shown 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. $\mathcal{X}(\mathbb{G}) \leq \Delta + 1$.

2) *Solving the VC Problem with a* CAPN-LA

To solve the VC problem with the CAPN-LA, first, an APN-LA will be constructed, referred to as APN-LA-VC, to represent a local algorithm to VC problem for each cell (or vertex) of graph $\mathbb{G}$, which is to be colored. Finally, a CAPN-LA will be constructed to represent the whole algorithm to solve VC problem by replicating the APN-LA-VC into all cells (or vertices) of graph $\mathbb{G}$. This yielded CAPN-LA is referred to as CAPN-LA-VC hereafter. Note that, since the cell c_i is the vertex v_i, hereafter (in some cases), vertex v_i may be referred to as cell c_i and vice versa.

A.The APN-LA-VC

Each vertex v_i of the graph $\mathbb{G}$ is modeled by the simple PN depicted in Figure 2. 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 construction procedure of the APN-LA [3], this PN is converted into an APN-LA, called APN-LA-VC, given in Figure 3. Note that the set $\{t_{i,1}, \dots, t_{i,m}\}$ is a cluster of conflicting transitions and forms the cluster $s_{i,1}$. In this marking, the conflict resolver, i.e. LA_i, is responsible for resolving effective conflicts, and as a consequence, is responsible for determining the color of the vertex v_i. This is done as follows: the action set of LA_i referred to as $\underline{\alpha}_i$, contains m actions. When LA_i is activated, it selects one of its actions, say $\alpha_{i,r}$, according to its action probability vector. As a result, the transition $t_{i,\mathrm{r}}$, corresponding to the selected action, is fired and a token is appeared in the place $P_{i,r}$. This way, the APN-LA-VC assigns a color to the vertex.

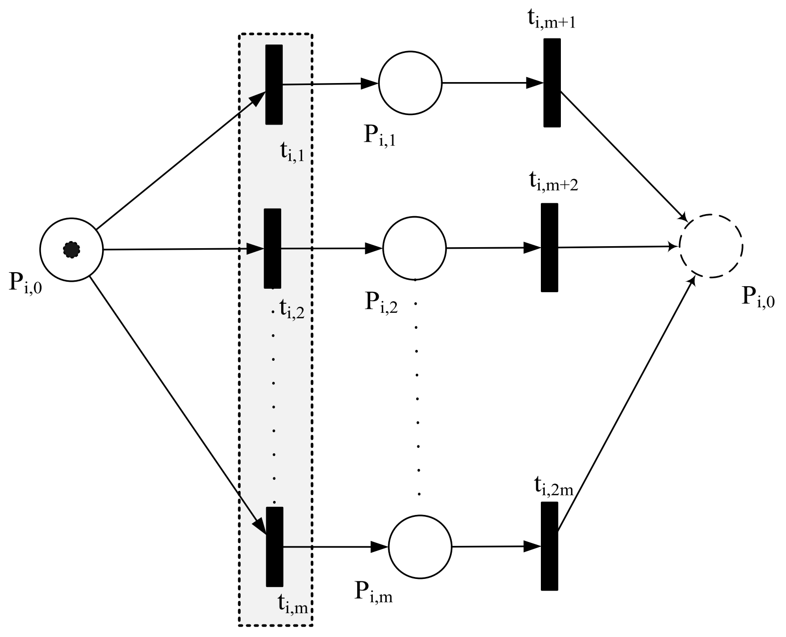

Figure 2: a simple PN for vertex v_i. For simplicity, $P_{i,0}$ has been duplicated (indicated by dash-line)

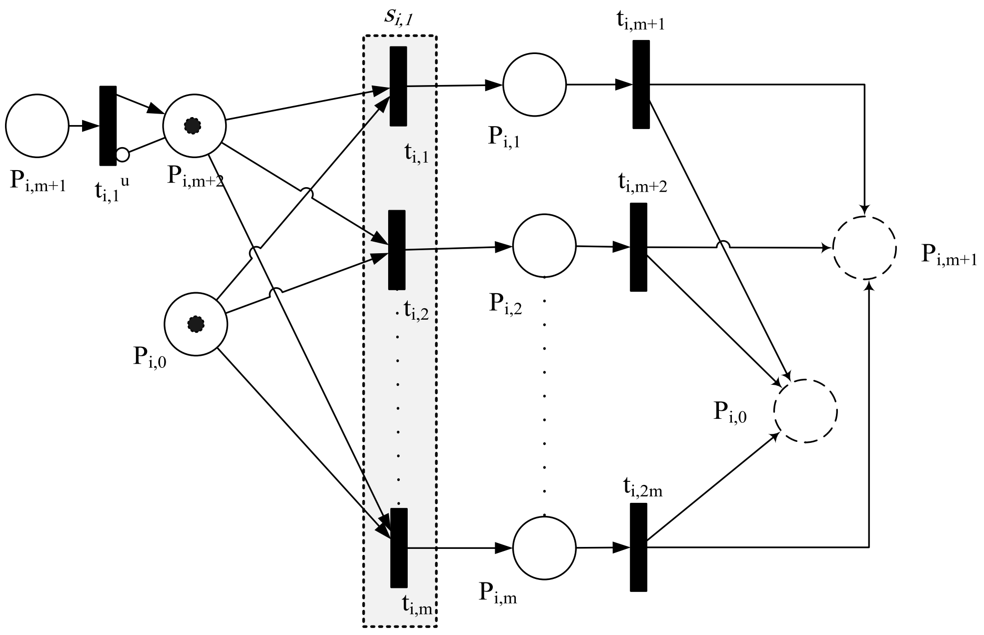

Figure 3: an APN-LA for vertex v_i. For simplicity, $P_{i,0}$ and $P_{i,m+1}$ have been duplicated (indicated by dash-line)

B.The CAPN-LA-VC

A CAPN-LA-VC is a graph of a number of APN-LA-VC such that a APN-LA-VC is assigned in each vertex of the graph. The CAPN-LA-VC system is defined as below:

- $\mathbb{G} =< \mathrm{V}, \mathbb{E} >$ is an undirected graph which is to be colored.
- $\{\mathrm{A}_1, \dots, \mathrm{A}_\mathrm{n}\}$ is a finite set of APN-LA-VCs as shown in Figure 3.
- $M_0 = [\mu_{1,0}, \dots, \mu_{n,0}]$ is the initial marking vector of $\mathcal{N}$, where $\mu_{i,0} = p_{i,0} + p_{i,m+2}$ is the initial marking of A_i.
- $\hat{\mathrm{F}} = \{\mathrm{F}_1, \cdots, \mathrm{F}_\mathrm{n}\}$ is the set of local rules. F_i, the local rule related to LA_i, is executed upon the firing of $t_{i,1}^u$ in the A_i and generates the reinforcement signal β_i for LA_i using the set of last firing transitions of all neighboring clusters of $s_{i,1}$. A simple local rule can be stated as follows:
    a. If the selected transition of $\mathrm{s}_{i,1}$ is different from the last selected transitions 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. In real scenarios, any other local rules can be used instead.

Based on the proposed CAPN-LA-VC, different algorithms can be designed by:

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

A solution of VC problem is obtained from the configuration of the conflict resolvers in the CAPN-LA-VC (refer to Definition 3). To this end, *max* operator is utilized over the action probability vector $\underline{q_i}$ to determine the color of vertex v_i; the color related to the action with the highest value in $\underline{q_i}$ is assigned to v_i.

To be able to compare such algorithms with each other, an evaluative measure is required. For this purpose, we utilize the measure of expediency, defined in Section V.1). In the next subsection, this measure will be calculated for CAPN-LA-VC using the results of the analysis of the proposed APN-LA-VC.

3) *Measure of Expediency for the CAPN-LA-VC*

To calculate the measure of expediency for the CAPN-LA-VC, we must determine the average reward for the vertex v_i at time instant k according to the following equation, which is a repetition of equation (8).

$$D_i\left(\underline{q}(k)\right) = \sum_{\mu\in\mu_i}\left(\sum_{t_i^j|En\left(t_i^j,\mu\right)} d_i^j\left(\underline{q}(k)\right)\times \hat{q}_i^j(k,\mu)\right)\times \pi_\mu(k) \qquad (20)$$

Note that according to equation (7), $d_i^j\left(\underline{q}(k)\right)$ depends on the local rule of the CAPN-LA-VC. In other words, different algorithms result in different average rewards in the vertex v_i (D_i) due to the differences in their local rules.

To achieve $\pi_\mu(k)$ in equation (20), reachability analysis method for APN-LA-VC can be used. To this end, first, the reachability graph of APN-LA-VC related to the vertex v_i is obtained and then, its equivalent DTMC, Discrete Time Markov Chain, will be achieved. Finally, this DTMC will be used to achieve $\pi_\mu(k)$.

Let first define a number of notations which is utilized in the rest of this section:

- L_k denotes the number of tokens in the place $P_{i,K}$,
- Marking $\mu_{i,j} = (L_0, L_1 \cdots L_{m_i}, L_{m_i+1}, L_{m_i+2})$ denotes the j^{th} marking of Petri net shown in Figure 3. This Petri net is related to APN-LA-VC A_i which is assigned to vertex v_i
- The initial marking is $\mu_{i,0} = (1,0,\dots,0,0,1)$ in which two values L_0 and L_{m_i+2} are equal to 1 and other values are equal to 0.

Figure 4 shows the reachability graph of A_i (Figure 3), obtained from the initial marking $\mu_{i,0}$. Figure 5 also shows the DTMC related to this reachability graph. The marking $\mu_{i,j}$ in the reachability graph is represented by state $\sigma_{i,j}$ in the DTMC. State $\sigma_{i,j}, (j = 1,\dots,m)$ states that the color j is assigned to vertex v_i.

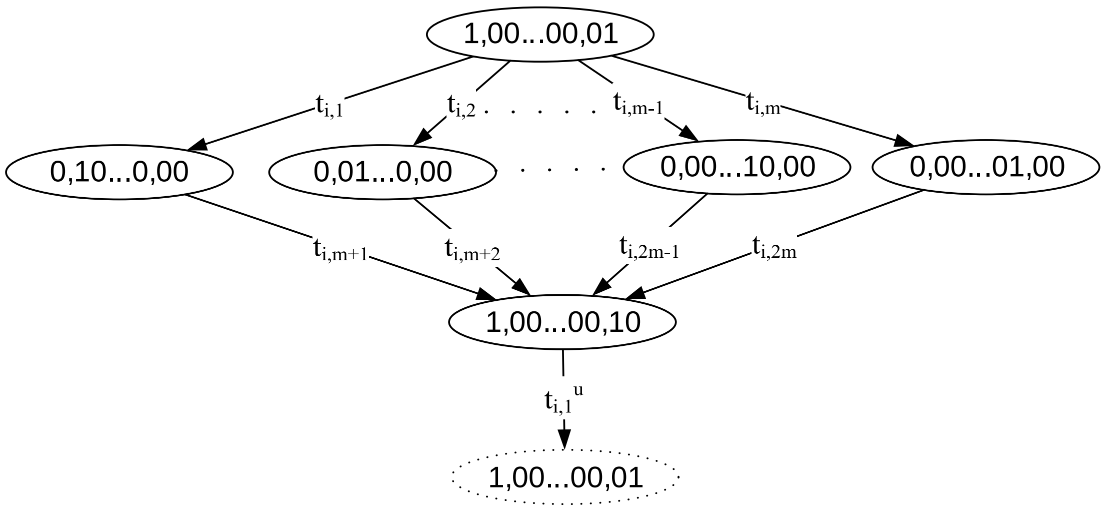

Figure 4: the reachability graph of APN-LA-VC A_i. For simplicity, $\mu_{i,0}$ has been duplicated (indicated by dash-line)

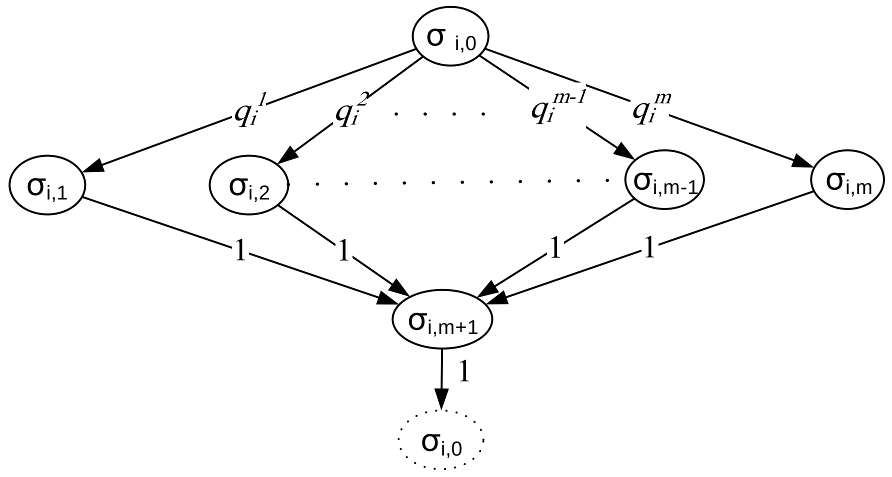

Figure 5: the DTMC of APN-LA-VC A_i. q_i^j is the probability of assigning color j to vertex v_i. For simplicity, $\sigma_{i,0}$ has been duplicated (indicated by dash-line)

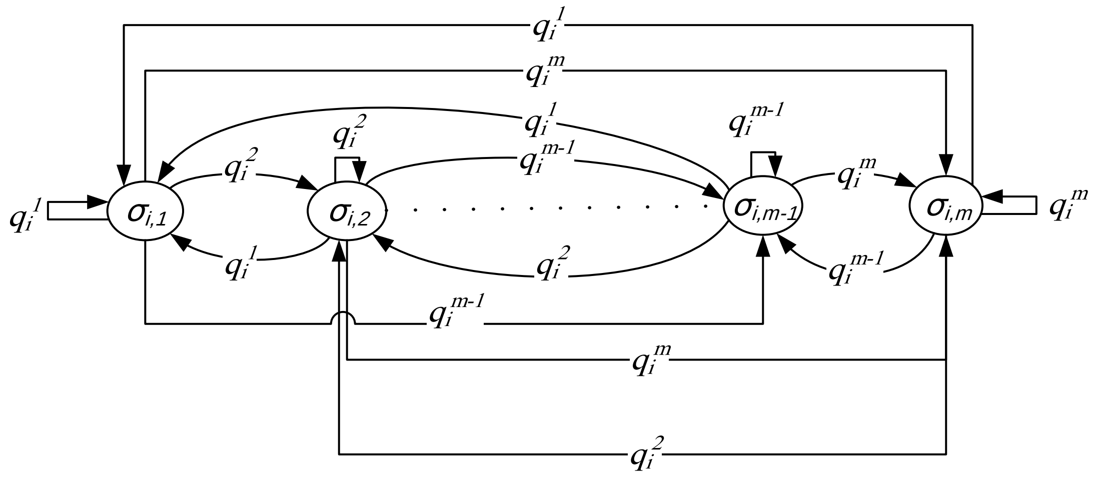

Figure 6: reduced DTMC for A_i

To reduce the number of states in the DTMC, useless states $\sigma_{i,0}$ and $\sigma_{i,m+1}$ are eliminated from the DTMC (Figure 6). From DTMC shown in Figure 6, $\pi_{\sigma_{i,l}}(k) = q_i^l(k)$. Therefore, the equation (20) is rewritten as equation (21) given below, in which $q_i^j(k)$ is used instead of $\hat{q}_i^j(k)$ because all transitions in the cluster $s_{i,1}$ are enabled in any state σ and therefore $q_i^j(k) = \hat{q}_i^j(k,\sigma)$.

$$D_i\left(\underline{q}(k)\right) = \sum_{\sigma\in\{\sigma_{i,1},\dots,\sigma_{i,m}\}} \left(\sum_{t_i^j} d_i^j\left(\underline{q}(k)\right) \times q_i^j(k) \right) \times q_i^l(k) \qquad (21)$$

Since $\sum_{t_i^j} d_i^j\left(\underline{q}(k)\right) \times q_i^j(k)$ is independent of the set of states $\sigma \in \{\sigma_{i,1}, \dots, \sigma_{i,m}\}$, we can rewrite equation (21) as below

$$D_i\left(\underline{q}(k)\right) = \left(\sum_{t_i^j} d_i^j\left(\underline{q}(k)\right) \times q_i^j(k) \right) \sum_{\sigma\in\{\sigma_{i,1},\dots,\sigma_{i,m}\}} q_i^l(k) \qquad (22)$$

Since $\sum_{\sigma \in \{\sigma_{i,1},\ldots,\sigma_{i,m}\}} q_i^l(k) = 1$, the average reward for the vertex v_i at time instant k is simplified to

$$D_i\left(\underline{q}(k)\right) = \left(\sum_{t_i^j} d_i^j\left(\underline{q}(k)\right) \times q_i^j(k)\right) \qquad (23)$$

According to the above equation, the measure of expediency for the CAPN-LA-VC at time instance k is defined as equation (24) given below

$$M_E(k) = \left(\frac{E\left[\sum_i D_i\left(\underline{q}(k)\right)\right]}{\sum_i \frac{1}{m_i}\sum_{t_i^j} d_i^j\left(\underline{q}(k)\right)}\right) - 1 \qquad (24)$$

The value of $M_E(k)$ is used for comparing different VC-algorithms represented by the CAPN-LA-VC, such that a VC-algorithm with higher value of $M_E(k)$ is more expedient than a VC-algorithm with lower value of $M_E(k)$ at time instance k.

4) A New Algorithm for Solving VC Problem

As mentioned before, different algorithms in CAPN-LA-VC can be designed by substituting different rules in the definition of local rules. In this section, we propose a new local rule for CAPN-LA-VC. To achieve that, we first analyze the DTMC of CAPN-LA-VC using the reduction of its states and then, using the analysis's results, we propose a new local rule. The efficiency of this new local rule will be shown in simulation results, described in the next section.

The states of DTMC shown in Figure 6 can be 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 the same as the color of at least one of the neighboring vertices. As a result, the graph in Figure 6 is reduced to a new DTMC with two states which is shown in Figure 7. In the reduced DTMC, the transition-probability matrix $\mathbb{P}_i$ is as follows:

$$\mathbb{P}_i = \begin{bmatrix} Z_{i,2} & Z_{i,1} \\ Z_{i,2} & Z_{i,1} \end{bmatrix} \qquad (25)$$

where

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

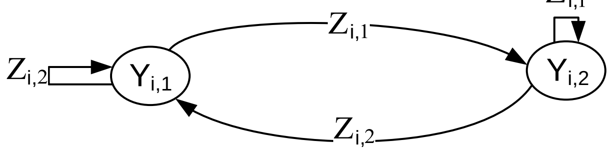

Figure 7: reduced DTMC for A_i

The probability of being in the state $Y_{i,1}$ at time instant k, i.e. $\pi_{Y_{i,1}}(k)$, 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:

$$\pi_{Y_{i,1}}(k) = \sum_{l=1}^{m} \left[q_i^l(k). \prod_{j \in \underline{N}_i} \left(1 - \Gamma_{j,l}(k)\right) \right] \qquad (26)$$

where $\underline{N_i}$ is the set of neighbors of the vertex v_i and $\Gamma_{i,l}(k)$ is defined according to equation (27). Informally, q_i^l, the probability of assigning color l to vertex v_i, takes part in $\pi_{Y_{i,1}}(k)$ if no neighbor of the vertex v_i is colored with color l.

$$\Gamma_{i,l}(k) = \begin{cases} 1, \alpha_i(k) = \alpha_{i,l} \\ 0, \alpha_i(k) \neq \alpha_{i,l} \end{cases} \qquad (27)$$

Now, we define a measure according to the following equations in which $\left|Y_{i,1}(k)\right|$ indicates the number of allowable colors in v_i at time instant k:

$$\omega_i(k) = \frac{\pi_{Y_{i,1}}(k)}{\left|Y_{i,1}(k)\right|} = \frac{\sum_{l=1}^{m}\left[q_i^l(k). \prod_{j \in \underline{N}_i}\left(1 - \Gamma_{j,l}(k)\right)\right]}{\sum_{l=1}^{m} \prod_{j \in \underline{N}_i}\left(1 - \Gamma_{j,l}(k)\right)} \qquad (28)$$

Using the proposed measure $\omega_i(k)$, the proposed new local rule is defined as follows:

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

In 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 ω_i than the average of ω_i resulted from previous selections of this LA.

5) *Simulation Results*

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

- CN: Number of colors required for coloring the graph,
- $\mathcal{U}$: Total number of updates on the learning automata, up to the time instant $\Bbbk$, where $\Bbbk$ is defined to be the smallest time instant at which the colors of all vertices in the graph are fixed,
- $M_E(\Bbbk)$: The value of the measure of expediency defined according to equation (24).

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 [66]. 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 five different VC-algorithms. The first four algorithms are the state of the art algorithms introduced in [49][50]; in each of which an ICLA is used to solve the VC problem. The fifth algorithm is proposed in this paper by introducing a novel local rule.
- Then, the experiment Two compares two scenarios of cooperation between APN-LAs to solve the VC problem.

A. Experiment One: Simulation on the CAPN-LA-VC

In this section, four different algorithms introduced in [49][50], namely VC1 to VC4, and a new algorithm proposed in this paper, namely VC5, are described.

VC1: In the first algorithm, which we call VC1, 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 $\mathbb{G}$; that is $m = \left|\underline{\alpha_i}\right| = |V|$. In the VC1, when LA_i is activated, an action from its action set is selected. All learning automata in the CAPN-LA-VC update their action probability vectors using an L_{R-I} learning algorithm. The local rules of the CAPN-LA-VC in VC1 algorithm are actually just simple local rules, defined as follows:

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

VC2: 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 shown 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, in VC2, 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 VC2 colors the graph in less time and with a smaller number of colors compared to the VC1.

VC3: In VC2, 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 VC3.

VC4: Since a suitable mechanism for solving VC problem needs to decrease the number of required colors for coloring the graph, an enhanced algorithm reduces the number of available actions in each LA, where possible. In this enhanced algorithm, called VC4, learning automata with varying number of available action set are used, which is introduced in [67]; if the action probability of an action $\alpha_{i,l}$ falls below a certain threshold ε, then $\alpha_{i,l}$ is removed from the available set of actions of the LA_i.

VC5: All above algorithms, i.e. VC1 to VC4, utilize the simple local rule defined for VC1. In this paper, we propose a new local rule in Section 4). Here, the VC4 algorithm with this proposed local rule is referred to as VC5.

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. Then, from this time on, the color of the vertex v_i will be fixed to the corresponding color, i.e. $\mathbb{c}_l$.

One problem with algorithms VC1 to VC4 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 set. The available actions for an LA at any time instant k are those which are not selected by any neighboring 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: $a = b = .1$, $\varepsilon = .009$, and $\tau = .95$. The results of this simulation study, which are reported in TABLE 1, indicate that:

- By moving from VC1 to VC4, CN and $\mathcal{U}$ criteria decrease and $M_E(\mathbb{k})$ criterion increases. In other words, VC4 is the best and VC1 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 as well as its accuracy increase.
- VC5 is always better or at least equal to VC4 in in terms of all the mentioned criteria. This efficiency is the results of considering the proposed measure ω_i in the local rule of the CAPN-LA-VC system.

TABLE 1: results of experiment for proposed algorithms represented by the CAPN-LA-VC

Algorithm / Graph	VC1			VC2			VC3			VC4			VC5		
	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	CN	$\mathcal{U}$	$M_E(\mathbb{K})$
DSJC125_1	5	1044679	.439	5	116002	.561	5	35679	.801	5	26876	.842	5	12593	.897
DSJC125_5	19	1067235	.494	18	124642	.558	17	46410	.760	17	39055	.827	17	27263	.851
DSJC125_9	48	1308101	.486	45	128119	.579	44	55224	.758	44	46407	.802	44	32004	.827
DSJC250_1	10	641605	.563	8	84145	.689	8	28757	.752	8	24925	.857	8	14661	.874
DSJC250_5	31	4418776	.532	29	421963	.672	28	130756	.739	28	100635	.838	28	45645	.839
DSJC250_9	76	6741839	.510	75	626042	.671	73	233568	.699	72	150091	.833	72	59200	.847
DSJC500_1	14	1110579	.502	13	144954	.602	12	86799	.721	12	67603	.870	12	50270	.889
DSJC500_5	52	2712399	.492	51	345971	.588	49	222161	.706	49	156358	.813	49	83597	.814
DSJC500_9	139	6735701	.482	131	648780	.578	128	324640	.685	127	285403	.773	127	158908	.782
le450_15a	15	4279467	.561	15	480839	.687	15	162445	.775	15	107575	.892	15	78335	.910
le450_15b	15	4629513	.556	15	497638	.635	15	183004	.734	15	138001	.798	15	59340	.864
le450_15c	16	4672330	.562	15	589748	.663	15	311569	.762	15	265139	.815	15	102384	.849
le450_15d	17	5284660	.585	15	502709	.663	15	299948	.771	15	203358	.809	15	11965	.831
le450_25c	26	5516730	.469	26	621156	.607	25	313236	.790	25	255099	.825	25	103542	.830
le450_25d	27	5737808	.448	26	636840	.592	25	334672	.699	25	275394	.786	25	157722	.798

B. Experiment two: Cooperation scenarios

This simulation study is conducted to compare the following two scenarios: 1) Handling the required cooperation between the neighboring cells within the controlling mechanisms, 2) Handling the required cooperation between the neighboring cells within the structure of Petri net. To this end, a Petri net consisting of multiple APN-LAs will be introduced, in which the required cooperation between any two neighboring vertices is handled within the structure of their APN-LAs. To handle the cooperation within the structure, a common way is to use inhibitor arcs. An inhibitor arc disables using of a color for a vertex when one of its neighbors uses this color for coloring. Therefore, to solve the VC problem, the APN-LA of the neighboring vertices are connected together by inhibitor arcs in order to disable using the illegal colors in the coloring of vertices. Figure 8 illustrates an APN-LA with its inhibitor arcs corresponding to the vertex v_i. The presented APN-LA in this figure is replicated in all vertices of the graph $\mathbb{G}$ and this way, a large Petri net will be achieved. This Petri net is referred to as APN-LA-IA, hereafter.

The number of required inhibitor arcs in the APN-LA-IA for coloring the graph $\mathbb{G}$ with n vertices is equal to $\sum_{i=1}^{n} m_i \times \Delta_{\mathrm{i}}$ where m_i is the number of available colors for coloring in vertex v_i and Δ_{i} is the degree of vertex v_i. Although using these inhibitor arcs decreases the number of markings in the state-space of APN-LA-IA, it increases the complexity of the structure of APN-LA-IA. To generate a legal coloring by APN-LA-IA, the lowest priority is considered for transitions $t_{i,m+1}, \ldots, t_{i,2m}, (i = 1, \ldots, n)$. This way, the assigned color to each vertex (i.e. appeared token in place $P_{i,1}\left|P_{i,2}\right| \ldots |P_{i,m}$) stays the same until the next coloring assignment to the vertex. The algorithm represented by APN-LA-IA is referred to as **VC6**. In VC6, the number of actions of each vertex v_i is equal to $\mathit{\Delta}_i + 1$ where $\mathit{\Delta}_i$ is the degree of vertex c_i. When the action probability of one of these actions, corresponding to $LA_i, (i = 1, \ldots, n)$, reaches a predetermined threshold τ, the color of the vertex v_i will be fixed to the corresponding color.

TABLE 2 compares VC5 (applied on the CAPN-LA-VC) and VC6 (applied on the APN-LA-IA) in terms of CN,$\mathcal{U}$, and $M_E(\mathbb{K})$ criteria. In addition, this table also reports the required number of inhibitor arcs for representing two mentioned scenarios. Consider a graph $\mathbb{G}$ with n vertices and $|\mathbb{E}|$ edges, in which the degree of vertex $v_i, (i = 1, .., n)$ is $\mathit{\Delta}_i$ and the number of possible colors for vertex v_i is $m_i = \mathit{\Delta}_i + 1$. Regarding the size of PNs (in terms of the number of places, transitions, arcs, and inhibitor arcs) used in these two scenarios, one can mention the following points:

- In the CAPN-LA-VC, for each vertex v_i, there exists a Petri net (Figure 3) in which there are $m_i + 3$ places, $2m_i$ transitions, $6m_i + 2$ arcs, one updating transition, and one inhibitor arc. Therefore, we have n Petri nets in which, there are totally $|\mathbb{E}| + 4n$ places, $2|\mathbb{E}| + 2n$ transitions, $6|\mathbb{E}| + 8n$ arcs, n updating transitions, and n inhibitor arcs.
- In the APN-LA-VC, there exists only one Petri net for the whole graph (Figure 8) in which there are $|\mathbb{E}| + 4n$ places, $2|\mathbb{E}| + 2n$ transitions, $6|\mathbb{E}| + 8n$ arcs, n updating transitions, and $n + \sum_{i=1}^{n} m_i \Delta_i$ inhibitor arcs.

From the
TABLE 2, the following remarks may be concluded:
VC5 outperforms VC6 in terms of CN, $\mathcal{U}$, and $M_E(\mathbb{K})$ criteria.

- When the cooperation between the neighboring cells is handled within the controlling mechanisms, the complexity of the firing rules in Petri net is decreased.

VII. CONCLUSION

In this paper, cellular adaptive Petri net based on learning automata (CAPN-LA) was proposed. The CAPN-LA consists of a graph and a number of APN-LAs, which each APN-LA being assigned to a vertex of the graph. Two APN-LAs assigned to two adjacent vertices are neighbors. In the APN-LA, which is constructed from a Petri net, transitions are divided into two clusters of concurrent and conflicting transitions. In the cluster of conflicting transitions, a learning automaton is utilized for resolving the conflicts. Two conflict resolvers assigned to two neighboring APN-LAs cooperate with each other to solve the existing problem. To solve the vertex coloring problem with CAPN-LA, an APN-LA was designed for representing a local algorithm to be executed in any vertex of the graph. This APN-LA, called APN-LA-VC, was then replicated into all vertices. Next, a CAPN-LA was constructed, called CAPN-LA-VC, in which the controlling mechanism handles the required cooperation between two neighboring APN-LA-VCs. Based on CAPN-LA-VC, several state of the art algorithms were represented. In addition, a new algorithm was also proposed which tries to achieve the following two goals simultaneously: 1) increasing the probability of selecting an allowable color at each vertex of the graph and 2) decreasing the required number of colors. Next, using the theoretical analysis of the APN-LA-VC, an expediency measure is provided for comparing these algorithms. According to this measure, a number of computer simulations were conducted and the algorithms were compared. The results of simulations indicated that the proposed algorithm, is more expedient than the state of the art algorithms and is able to color graphs with lower number of iterations and fewer numbers of colors.

TABLE 2: results of experiment for VC6 represented by the APN-LA-IA. #IA is the required number of inhibitor arcs.

Algorithm / Graph	VC5				VC6			
	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	#IA	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	#IA
DSJC125_1	5	12593	.897	125	5	35722	.782	20279
DSJC125_5	17	27263	.851	125	18	46415	.745	495871
DSJC125_9	44	32004	.827	125	44	55231	.756	1565939
DSJC250_1	8	14661	.874	250	8	28765	.750	178900
DSJC250_5	28	45645	.839	250	28	130771	.734	3974526
DSJC250_9	72	59200	.847	250	73	233584	.698	1253204
DSJC500_1	12	50270	.889	500	12	86814	.719	1289234
DSJC500_5	49	83597	.814	500	49	222180	.704	31560758
DSJC500_9	127	158908	.782	500	128	324658	.685	101382226
le450_15a	15	78335	.910	450	15	162443	.777	737068
le450_15b	15	59340	.864	450	15	182986	.736	730290
le450_15c	15	102384	.849	450	15	311587	.761	2709830
le450_15d	15	119652	.831	450	16	299983	.767	2730334
le450_25c	25	103542	.830	450	26	313249	.773	3091758
le450_25d	25	157722	.798	450	26	334671	.699	3098528

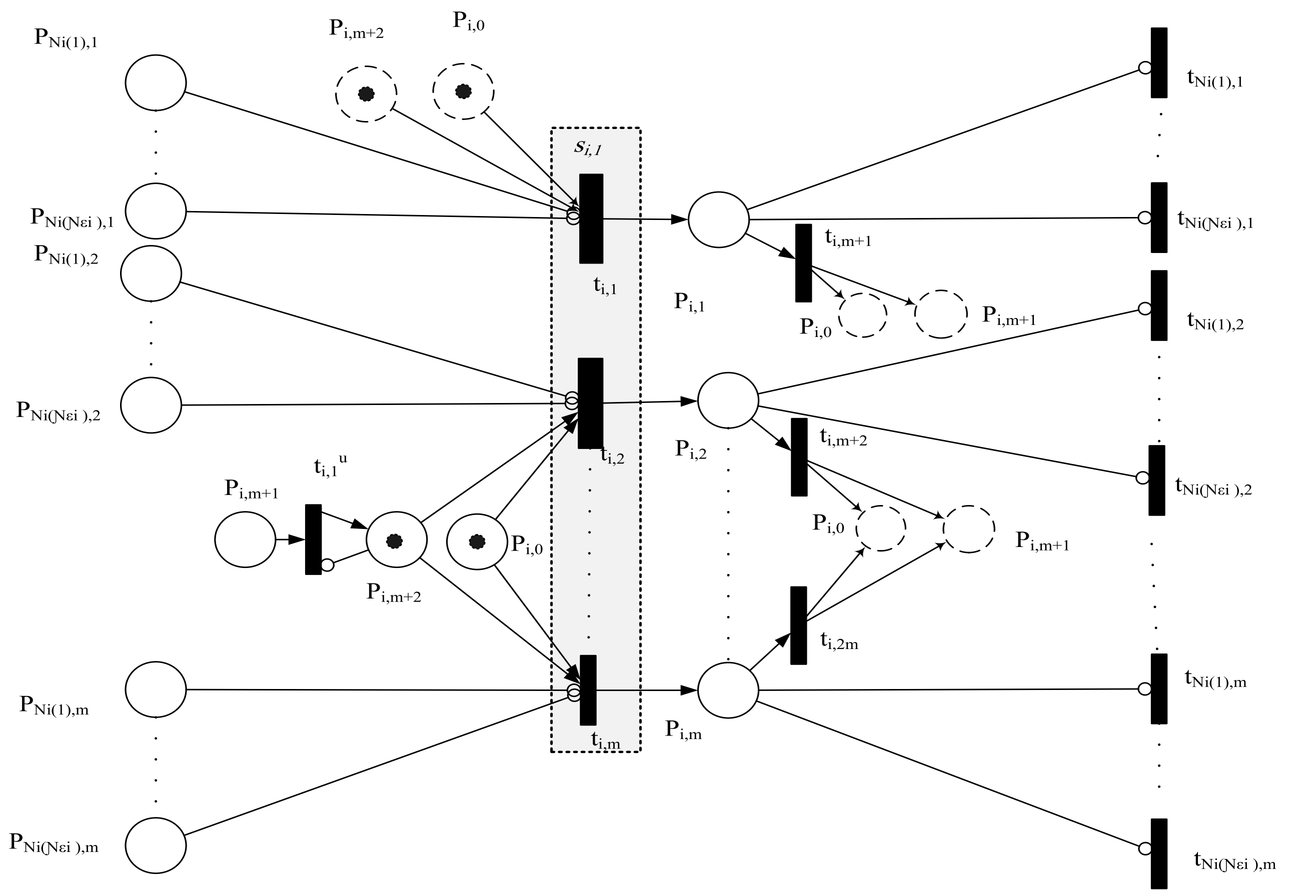

Figure 8: An APN-LA corresponding to vertex v_i in which inhibitor arcs are utilized to disable illegal coloring of the neighboring vertices. The vertex v_i has $|NE_i|$ neighbors indicated by $N_i(1), \dots, N_i(\mathcal{NE}_i)$. For simplicity several places have been duplicated (indicated by dash-line).

References

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

[2] W. Reisig, "The Synthesis Problem", *Transactions on Petri Nets and Other Models of Concurrency VII*, K. Jensen, W. M. P. van der Aalst, G. Balbo, M. Koutny, and K. Wolf, Eds. Springer Berlin Heidelberg, pp. 300–313, 2013.

[3] 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.

[4] J. D. Terán-Villanueva, et. al., "Cellular processing algorithms", *Soft Computing Applications in Optimization*, Control, and Recognition, Springer Berlin Heidelberg, pp. 53-74, 2013.

[5] N. Linial, "Locality in distributed graph algorithms", *SIAM Journal on Computing*, vol. 21, no. 1, pp. 193-201, 1992.

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

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

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

[9] F. Bause, "On the analysis of Petri net with static priorities", *Acta Informatica*, vol. 33, pp. 669-685, 1996.

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

[11] 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.

[12] R. M. Fujimoto, "Parallel simulation: parallel and distributed simulation systems", *Proceedings of the 33nd conference on winter simulation*, IEEE Computer Society, pp. 147-157, 2001.

[13] G. Chiola and A. Ferscha, "Distributed simulation of Petri nets", *IEEE Concurrency*, no. 3, pp. 33-50, 1993.

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

[15] T. R. Jensen, and B. Toft, *Graph Coloring Problems*, John Wiley & Sons, USA, 1994..

[16] M. A. Thathacher and B. R. Harita, "Learning automata with changing number of actions", *IEEE Transactions on Systems*, Man and Cybernetics, vol. 17, no. 6, pp. 1095-1100, 1987.

[17] 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.

[18] 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.

[19] 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.

[20] R. J. Williams, "Toward a Theory of Reinforcement Learning Connectionist Systems", *Technical Report*, Nu-ccs-88-3, Northeastern University, Boston, MA, 1988.

[21] E. A. Billard, "Stability of Adaptive Search in Multi-Level Games under Delayed Information", *IEEE Transactions on Systems*, Man, and Cybernetics-Part A: Systems and Humans, vol. 26, pp. 231-240, 1996.

[22] E. A. Billard, "Chaotic Behavior of Learning Automata in Multi-Level Games under Delayed Information", *Proc. of IEEE Intl. Conf. on Systems, Man, and Cybernetics*, pp. 1412-1417, 1997.

[23] 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.

[24] R. Rastegar, A. R. Arasteh, A. Harriri, and M. R. Meybodi, "A Fuzzy Clustering Algorithm Using Cellular Learning Automata based Evolutionary Algorithm", *Proc. of the Fourth Intl. Conf. on Hybrid Intelligent Systems (HIS04)*, Japan, Kitakyushu, pp. 310-314, 2004.

[25] R. Vafashoar, M. R. Meybodi, and A. H. Momeni, "CLA-DE: A Hybrid Model based on Cellular Learning Automata for Numerical Optimization", *Journal of Applied Intelligence*, Springer Verlag, vol. 36, no. 3, pp. 735-748, 2012.

[26] B. Hashemi and M. R. Meybodi, "Cellular Pso: A Pso for Dynamic Environments", *Advances in Computation and Intelligence*, Lecture Notes in Computer Science, vol. 5821/2009, pp. 422-433, 2009.

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

[28] S. Y. Lin and T. Y. Chan, “A Petri-Net-Based Automated Distributed Dynamic Channel Assignment for Cellular Network”, *Vehicular Technology, IEEE Transactions on*, vol. 58, no. 8, pp.4540-4553, 2009.

[29] S. H. Teng and J. T. Black, “Cellular manufacturing systems modeling: the Petri net approach”, *Journal of Manufacturing Systems*, vol. 9, no. 1, pp. 45-54, 1990.

[30] G. Bruno and P. Biglia, "Performance Evaluation and Validation of Tool Handling in Flexible Manufacturing Systems Using Petri Nets", *International Workshop on Timed Petri Nets*, Torino, Italy, pp. 64-7 l, 1985.

[31] R. Ravichandran and A.K. Chakravarty, "Decision Support in Flexible Manufacturing Systems Using Timed Petri Nets", *Journal of Manufacturing Systems*, vol. 5, no. 2, pp. 89-101, 1986.

[32] A. A. Merabet. "Synchronization of Operations in a Flexible Manufacturing Cell: The Petri Net Approach", *Journal of Manufacturing Systems*, vol. 5, no. 3, pp. 161-169, 1986.

[33] F. James-Romero, D. Munoz-Rodriguez, C. Molina, and H. Tawfik, “Modeling resource management in cellular systems using Petri nets”, *IEEE Trans. Veh. Technol.*, vol. 46, no. 2, pp. 298–312, May 1997.

[34] Y. Ma, J. J. Han, and K. S. Trivedi, "Call admission control for reducing dropped calls in CDMA cellular systems", *Computer Communications*, vol. 25, no. 7, pp. 689-699, 2002.

[35] S. Geetha and R. Jayaparvathy, “Modeling and Analysis of bandwidth Allocation in IEEE 802.16 MAC: A Stochastic Reward net Approach,” Int. J. Communications, Network ans System Sciences , vol. 3, no. 7, pp. 631–637, July 2010.

[36] S. Dharmaraja, K. S. Trivedi, and D. Logothetis, “Performance modeling of wireless networks with generally distributed handoff interarrival times”, Computer Communications, vol. 26, no. 15, pp.1747-1755, 2003.

[37] R. Schoenen, A. Bin Sediq, H. Yanikomeroglu, G. Senarath, and Z. Chao, "Fairness analysis in cellular networks using stochastic Petri nets", *Personal Indoor and Mobile Radio Communications (PIMRC)*, 22nd IEEE International Symposium on, pp. 1983-1988. 2011.

[38] R. J. Haines, G. R. Clemo, and A. T. D. Munro, “Petri-nets for formal verification of MAC protocols”, *IET Software*, vol. 1, no. 2, pp. 39-47, 2007.

[39] A. Salah and K. Mustafa, *Protocol Verification and Analysis using Colored Petri Nets*, Technical report, DePaul University, Chicago USA, 2004.

[40] Y. Pengand, Y. Zhanting, and W. Jizeng, “Petri net model of session initiation protocol and its verification”, *International Conference on Communications (ICC)*, vol. 1, pp. 1861–1864, 2007.

[41] J. Liu, X. Ye, J. Zhang, and J. Li, “Security verification of 802.11i 4-way handshake protocol”, *International Conference on Communications (ICC’08)*, vol. 1, pp. 1642–1647, 2008.

[42] H. Rodríguez, et al., *Using Petri Net for Modeling and Analysis of an Encryption Scheme for Wireless Sensor Networks*, INTECH Open Access Publisher, 2010.

[43] B. Liu, F. Ren, C. Lin, and X. Jiang, “Performance analysis of sleep scheduling schemes in sensor networks using stochastic Petri net”, *International Conference on Communications (ICC’08)*, vol. 1, pp. 4278 – 4283, 2008.

[44] Z. Rongfei, et. al., "Performance Analysis of Data Management in Sensor Data Storage via Stochastic Petri Nets", *Global Telecommunications Conference (GLOBECOM 2010)*, IEEE, vol. 1, no. 5, pp.6-10, 2010.

[45] A. Shareef and Zhu. Yifeng, "Energy modeling of wireless sensor nodes based on Petri nets", *the 39th International Conference on Parallel Processing (ICPP)*, IEEE, pp. 101-110, 2010.

[46] D. M. Ibrahim, et. al., "Modelling of CVBF algorithm using Colored Petri Nets", *the 9th International Conference on Computer Engineering and Systems (ICCES)*, vol. 26, no. 31, pp. 22-23, 2014.

[47] A. Dâmaso, N. Rosa, and P. Maciel, "Using Colored Petri Nets for Evaluating the Power Consumption of Wireless Sensor Networks", *International Journal of Distributed Sensor Networks*, 2014.

[48] S. Chen, et. al., “Modelling and performance analysis de wireless sensor network systems using Petri nets”, *International Technical Conference on Circuits Systems, Computers and Communications*, pp. 1689-1692, 2008.

[49] J. Akbari Torkestani, *Channel Assignment and Multicast Routing in Mobile Ad Hoc Networks Based on Learning Automata*, PhD dissertation, Dep. of Eng., Univ. Islamic Azad, Iran, 2009.

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

[51] N. Barnier, and P. Brisset, “Graph Coloring for Air Traffic Flow Management”, *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques*, Le Croisic, France, pp. 133-147, 2002.

[52] M. Carter, G. Laporte, and S. Lee, “Examination Timetabling: Algorithmic Strategies and Applications”, *Journal of the Operational Research Society*, vol. 47, pp. 373-83, 1996.
[53] R. Lewis, and B. Paechter, “Finding Feasible Timetables using Group Based Operators”, *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 3, pp. 397-413, 2007.
[54] A. M. Zymolka, C. A. Koster, and R. Wessaly, “Transparent Optical Network Design with Sparse Wavelength Conversion”, *Proceedings of the 7th IFIP Working Conference on Optical Network Design a nd Modelling*, Budapest, Hungary, pp. 61-80, 2003.
[55] R. M. Karp, “Reducibility among Combinatorial Problems”, *Complexity of Computer Computations*, Plenum Press, USA, pp. 85-103, 1972.
[56] B. B. Mabrouk, H. Hasni, and Z. Mahjoub, “On a Parallel Genetic-tabu Search Based Algorithm for Solving the Graph Coloring Problem”, *European Journal of Operational Research*, vol. 197, pp. 1192-1201, 2009.
[57] H. Asmuni, E. K. Burke, J. M. Garibaldi, B. McCollum, and A. J. Parkes, “An Investigation of Fuzzy Multiple Heuristic Orderings in the Construction of University Examination Timetables”, *Computers & Operations Research*, vol. 36, pp. 981-1001, 2009.
[58] E. Malaguti, and P. Toth, “An Evolutionary Approach for Bandwidth Multi-coloring Problems”, *European Journal of Operational Research*, vol. 189, pp. 638-651, 2008.
[59] P. M. Talavan, and J. Yanez, “The Graph Coloring Problem: A Neuronal Network Approach”, *European Journal of Operational Research*, vol. 191, pp. 100-111, 2008.
[60] J. Akbari Torkestani, and M. R. Meybodi, “A New Vertex Coloring Algorithm Based on Variable Action-Set Learning Automata”, *Journal of Computing and Informatics*, vol. 29, no. 3, pp. 1001-1020, 2010.
[61] Sh. Golzari and M. R. Meybodi, "A Parallel Graph Coloring Algorithm for Two Dimensional Cellular Automata", *Proceedings of Tenth Conference on Electrical Engineering*, University of Tabriz, vol. 1, pp. 288-299, May 2002.
[62] J. Akbari Torkestani and M. R. Meybodi, "Graph Coloring Problem Based on Learning Automata", *Proceedings of International Conference on Information Management and Engineering*, Kuala-Lumpur, Malaysia, pp. 718-722, April , 2009.
[63] J. Akbari Torkestani and M. R. Meybodi, "A Cellular Learning Automata based Algorithm for Solving the Vertex Coloring Problem", *Expert Systems with Applications*, vol. 38, no. 8, pp. 9237-9247, 2011.
[64] M. Enayatzare and M. R. Meybodi, "Solving Graph Coloring Problem using Cellular Learning Automata", *Proceedings of 14th Annual CSI Computer Conference of Iran*, Amirkabir University of Technology, Tehran, Iran, March, 2009.
[65] A. Enami Eraghi, J. Akbari Torkestani, and M. R. Meybodi, "Cellular Learning Automata-based Graph Coloring Problem", *Proceedings of 2009 International Conference on Machine Learning and Computing*, Perth, Australia, pp. 163-167, July, 2009
[66] Ftp://dimacs.rutgers.edu/pub/challenge/graph/.
[67] J. Zhang, C. Wang, and M. C. Zhou, "Last-Position Elimination-Based Learning Automata," *Cybernetics, IEEE Transactions on* , vol.44, no.12, pp.2484,2492, Dec. 2014.

S. Mehdi Vahidipour received the BSc and MSc degrees in computer engineering in Iran, in 2000 and 2003, respectively. He is also a lecturer in University of Kashan.
He is currently working toward the PhD degree in computer engineering in Amirkabir University of Technology, Iran. His research interests include distributed artificial intelligence, learning automata, and Adaptive Petri nets.

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

Mehdi Esnaashari received the B.S., M.S., and the 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.