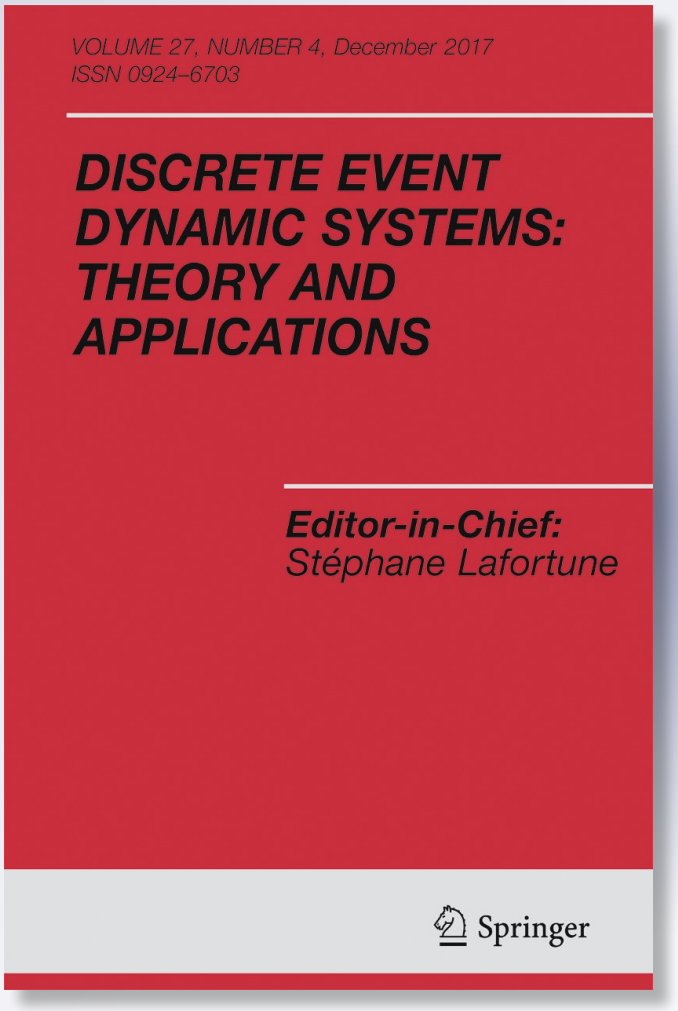
Cellular adaptive Petri net based on learning automata and its application to the vertex coloring problem

S. Mehdi Vahidipour, Mohammad Reza Meybodi & Mehdi Esnaashari

Discrete Event Dynamic Systems
Theory and Applications

ISSN 0924-6703
Volume 27
Number 4

Discrete Event Dyn Syst (2017)
27:609-640
DOI 10.1007/s10626-017-0251-z



Springer

Discrete Event Dyn Syst (2017) 27:609–640
DOI 10.1007/s10626-017-0251-z

Cellular adaptive Petri net based on learning automata and its application to the vertex coloring problem

S. Mehdi Vahidipour[1,2] · **Mohammad Reza Meybodi**[3] · **Mehdi Esnaashari**[4]

Received: 5 October 2016 / Accepted: 29 May 2017 / Published online: 4 July 2017

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 a so called controlling mechanism. Whenever a Petri net is used to represent an algorithm, the application of a different controlling mechanism results in a different instance 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. A 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 tedious 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 generated from the cooperation of these identical algorithms. To avoid having large and complex APN-LAs for cellular problems, a cellular adaptive Petri net called CAPN-LA is proposed in this paper, 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 is 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

✉ S. Mehdi Vahidipour
vahidipour@kashanu.ac.ir

[1] Faculty of Electrical and Computer Engineering, Department of Computer, University of Kashan, Kashan, Iran

[2] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), P.o.Box 19395-5746 Tehran, Iran

[3] Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran

[4] Computer Engineering Department, K. N. Toosi University of Technology, Tehran, Iran

measure of expediency is defined for comparing different CAPN-LAs according to their expected reward; a CAPN-LA which receives a 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.

Keywords Adaptive Petri net · Learning automata · Cellular algorithm · Graph-based problems · Vertex coloring (VC) problem

1 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 (Reisig 2013a).

Petri nets can also be used to represent algorithms of different classes (Reisig 2013b; Vahidipour et al. 2015). One class of algorithms is that of the cellular algorithms (Terán-Villanueva et al. 2013), which commonly follows the distributed computing model (Linial 1992). This model is appropriate for representing cellular algorithms because it is made up of cells, like points in a lattice or squares on a checker board, with local interactions. Each cell executes the same algorithm, referred to as the local algorithm, and the problem as a whole is solved by means of cooperation between neighboring cells. Petri nets can also be used to represent such cellular algorithms. For this to be possible, two steps can be taken:

1) A Petri net must be designed to represent the local algorithm to be executed in each cell. This Petri net is then replicated into all cells.
2) A mechanism must be devised to handle the required cooperation between any two neighboring cells.

In the first step, when a Petri net 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 a mechanism for making decisions. Each decision point in a Petri net arises when two or more transitions are enabled in a given marking. At this point, the decision to be made is the selection of an enabled transition for firing from the set of enabled transitions. In Petri net literature, such decision making mechanisms are referred to as controlling mechanisms (Peterson 1981). Different controlling mechanisms have previously been proposed in literature, such as a random mechanism (Murata 1989), queue regimes (Burkhard 1981), priority (Bause 1996; Bause 1997), external controller (Holloway and Krogh 1994), and APN-LA (Vahidipour et al. 2015). For a local solution, any of these controlling mechanisms may result in a specific local algorithm.

For the cooperation mechanism between any two neighboring cells, required in the second step above, a common approach which is typically adopted in the Petri net literature is to represent the cooperation within the structure of the two Petri nets designed for the cells; that is, by connecting the two Petri nets using ordinary Petri net elements such as common places and inhibitor arcs (Reisig 2013a). This results in a large Petri net for the whole algorithm, which is extremely hard to analyze and simulate (Fujimoto 2001; Chiola and Ferscha 1993).

To overcome the issue of large Petri nets, we suggest in this paper that the cooperation between the two neighboring cells should 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 allows such a property. This is due to the fact that in this mechanism, control is handled using a learning automaton (LA), and LAs have been shown to be able to cooperate with each other (Narendra and Thathachar 1989). Furthermore, APN-LA, unlike the other controlling mechanisms, is adaptive; that is, it can adapt itself to the changes in conflicts between the transitions. Such an adaptation is useful when Petri nets are used for modeling the dynamics of real-world problems (Vahidipour et al. 2015). Therefore, in this paper, a cellular adaptive Petri net called CAPN-LA is proposed in which a number of APN-LAs are organized into a cellular structure. 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 concept of learning (Esnaashari and Meybodi 2015). 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 from the set of enabled transitions. The steady-state behavior of CAPN-LA is studied, and the conditions under which a CAPN-LA becomes expedient are presented.

Finally, CAPN-LA is utilized to present various algorithms for solving the vertex coloring (VC) problem (Jensen and Toft 1994). To this end, an APN-LA is first designed to represent 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 the VC problem. In this paper, we use six different sets of controlling parameters for APN-LAs to form six different algorithms for solving the 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 2 represents the basic definitions. Section 3 reviews the subject of related work. In section 4, the proposed CAPN-LA is described. In section 5, the behavior of CAPN-LA is analyzed. In section 6, application of the CAPN-LA to the vertex coloring problem is provided. Section 7 includes the conclusion.

2 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.

2.1 Learning automata

A Learning Automaton (LA) is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment (Narendra and Thathachar 1989). 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 $E = \left\{\underline{\alpha}, \underline{\beta}, \underline{c}\right\}$, where $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ is the finite set of inputs, $\underline{\beta} = \{\beta_1, \beta_2, \ldots, \beta_r\}$ is the set of outputs i.e., the reinforcement signal, and $\underline{c} = \{c_1, c_2, \ldots, c_m\}$ is the set of 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, $(i = 1, \ldots, m)$ 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 (Narendra and Thathachar 1989). In what follows, variable structure LA, which will be used in this paper, will be briefly described. A variable structure LA is represented by $\left(\underline{\alpha}, \underline{\beta}, \underline{q}, L\right)$, where $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ is the set of actions, $\underline{\beta} = \{\beta_1, \ldots, \beta_r\}$ is the set of inputs, $\underline{q} = \{q_1, q_2, \ldots, q_m\}$ 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, defined by (1) and (2), is a linear learning algorithm which updates the action probability vector $\underline{q}$ using

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

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

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

when the taken action is penalized by the environment (i.e., $\beta(k) = 1$). In Eqs. (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 Eqs. (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 $a(n) \gg b(n)$[1]; 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.

[1] The notation $a \gg b$ means that a is **much greater than** b.

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 (Thathacher and Harita 1987; Zhang et al. 2015). Recently, several LA based approaches have been presented for improving the performance of many applications (Zhang et al. 2016; Ghavipour and Meybodi 2016; Rezvanian and Meybodi 2016).

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. In the rest of this sub-section, Cellular Learning Automata (CLA) is briefly introduced.

Cellular learning automaton (CLA) (Meybodi et al. 2003) is a combination of cellular automaton (CA) and learning automaton (LA). The basic idea of CLA is to use LA for adjusting the state transition probability of a stochastic CA. This model, which opens a new learning paradigm, is superior to CA because of its ability to learn and is also superior to single LA because it consists of a collection of LAs interacting with each other. A CLA is a CA, in which a number of LAs is assigned to every cell. Each LA, residing in a particular cell, determines its action (state) on the basis of its action probability vector. Like CA, there is a local rule that the CLA operates under. The local rule of the CLA and the actions selected by the neighboring LAs of any particular LA determine the reinforcement signal to that LA. The neighboring LAs (cells) of any particular LA (cell) constitute the local environment of that LA (cell). The local environment of an LA (cell) is non-stationary due to the fact that the action probability vectors of the neighboring LAs vary during the evolution of the CLA. The operation of a CLA could be described as the following steps: at the first step, the internal state of every cell is determined on the basis of the action probability vectors of the LAs residing in that cell and its neighboring LAs. In the second step, the local rule of the CLA determines the reinforcement signal to each LA residing in that cell. Finally, each LA updates its action probability vector based on the supplied reinforcement signal and the chosen action. This process continues until the desired result is obtained.

In the literature, several complex models have been introduced constructed atop of the CLA, such as CLA-based evolutionary computing (Rastegar et al. 2004), differential evolution-based CLA (Vafashoar et al. 2012), CLA-based particle swarm optimization (Hashemi and Meybodi 2009), and Irregular CLA (Esnaashari and Meybodi 2015). Also, there are other complex models which are designed atop the LA, such as hierarchical system of LA (Thathachar and Satstry 1997), distributed LA (DLA) (Beigy and Meybodi 2006), extended DLA (Mollakhalili Meybodi and Meybodi 2014), network of LA (Williams 1988), multi-level game of LA (Billard 1996; Billard 1997).

2.2 Petri nets

A Petri net is one of several mathematical modeling languages for describing concurrent 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 $W: ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}$ defines the interconnections of both sets 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, if $\exists p \in \dot{P}, M(p) > 0$; otherwise $\dot{P}$ is unmarked or empty in M.
- A transition t is *enabled* in marking M, denoted by $M[t>$, if $M(p) \geq W(p,t)$, $\forall p \in P$.
- 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: $M[t > M' and\ M'[t'\not>$ or $M[t' > M'' and\ M''[t\not>$.
- 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 (Peterson 1981; Murata 1989). 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 each of the following two situations: 1) All enabled transitions are concurrent and causally independent, which means that one transition may fire before, after or in parallel with the others (Murata 1989); and 2) some of the enabled transitions are in conflict, which means if any one of them is fired, none of the others can be fired (Murata 1989). In this paper, the mechanisms for selecting an enabled transition from a set of concurrent transitions are hereafter referred to as *concurrent selectors*, and the other mechanisms are referred to as *conflict resolvers*.

2.3 Adaptive Petri net based on learning automata

Several approaches are used in the literature to make PNs adaptive by fusing intelligent techniques such as neural networks and fuzzy logic systems with PNs (Zhou and Venkatesh 1998; Gao et al. 2003; Ding et al. 2016; Vahidipour and Meybodi 2013). Recently, an adaptive Petri net based on learning automata (APN-LA) was introduced in (Vahidipour et al. 2015), in which an adaptive Petri net was obtained from a fusion between Petri nets and LAs. In this Petri net, an LA is used as the conflict resolver for 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 between them. To form these clusters, we consider the transitions with the same input places as the members of a cluster. Thus, 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 (Vahidipour et al. 2015):

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 $s_i, (i = 1, \ldots, n)$ 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; 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 $s_i, (i = 1, \ldots, n)$, a transition with one input and one output place will be inserted into the Petri net. These extra elements are used to provide a way of fusing the Petri nets and learning automata. The 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 are described in (Vahidipour et al. 2015).

The firing rules of an APN-LA system differ from those of an ordinary PN due to their fusion with LAs (Vahidipour et al. 2015). The set of firing rules of an APN-LA in a given marking can be briefly described as follows:[2]

1- Determine the set of enabled transitions.
2- Specify a firing cluster from 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 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.

3 Related work

This section consists of two parts. The first part briefly reviews controlling mechanisms which have previously been used in Petri nets. In the second part, an overview is provided of the learning automata-based algorithms designed to solve the vertex coloring problem.

[2] More details on the firing rules of the APN-LA are given in (Vahidipour et al. 2015).

3.1 Controlling mechanisms in Petri nets

When a set of enabled transitions in a PN are in conflict, a controlling mechanism is needed to choose one for firing. There are a number of controlling mechanisms which have been addressed in the literature. Random selection (Peterson 1981) is the most commonly used of these controlling mechanisms. Another set of mechanisms which is applied to control the firing order in PNs is the queue regime (Burkhard 1981). 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. The enabled transition to which the highest priority has been assigned will be fired at any marking (Bause 1996; Bause 1997). The priority net can be either static (Bause 1996) or dynamic (Bause 1997). In the static case, a transition's priority level does not change. However, in the dynamic case, a transition's level of priority is fixed for a particular marking, although it differs for different markings.

Controlled Petri Nets (CtlPNs) are a type of PNs in which there is a new kind of place, known as a control place (Holloway and Krogh 1994). This control place allows an external controller 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, to prevent the CtlPN from reaching a set of forbidden markings. General methods (Holloway and Krogh 1994) such as path-based approaches and linear integer programming can be used to design this type of control policy.

A controlling mechanism was addressed in (Vahidipour et al. 2015) which is known as 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 (Vahidipour et al. 2015), APN-LA was used by the authors to solve the problem of priority assignment in queuing systems. In this case, the goal was to provide a controlling mechanism to select jobs from 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.

3.2 Vertex coloring algorithms based on learning automata

The problem of vertex coloring (VC) is a variation of the graph coloring (GC) problem (Jensen and Toft 1994; Akbari Torkestani 2009). The problem of graph coloring is a classical problem of combinatorial optimization in graph theory, and is broadly used in real-life applications such as computer air traffic flow management (Barnier and Brisset 2002), timetabling (Carter et al. 1996; Lewis and Paechter 2007), and light wavelength assignment in optical networks (Zymolka et al. 2003).

In the vertex coloring problem, each vertex of the graph is assigned a color such that there are no two adjacent vertices with the same color (Akbari Torkestani 2009). Since the vertex coloring problem is NP-hard for general graphs (Karp 1972), exact algorithms can only be used for small graphs; in practice, very large graphs are usually generated in a wide range of applications (Akbari Torkestani 2009), although in real-life applications, a near-optimal coloring of the graph will usually suffice. Therefore, a host of algorithms with polynomial time approximation have been introduced to find near-optimal solutions for the coloring problem (Jensen and Toft 1994). The approximation methods addressed in the literature can be categorized as genetic algorithms (Mabrouk et al. 2009), fuzzy-based optimizations

(Asmuni et al. 2009), evolutionary algorithms (Malaguti and Toth 2008), neural network approaches (Talavan and Yanez 2008) and so on.

A number of learning automata-based algorithms for solving the VC problem have been addressed in the literature. In a high proportion of these algorithms, as the learning process reaches its end, each vertex learns how to select a color in order to complete the graph using the smallest number of colors, and with no neighbors of the same color (Akbari Torkestani 2009). For example, an approximation algorithm was introduced in (Akbari Torkestani and Meybodi 2010) 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 several stages. Initially, the learning automata select a subset of the non-neighbor graph vertices and assign the same color to them. Then, the rest of the uncolored vertices will be selected and colored. An iteration is considered complete if all the vertices are colored. Here, if the size of the color-set (the number of colors selected) is smaller than the number of selected colors, as chosen earlier, the coloring is rewarded; otherwise it is penalized. When the algorithm reaches 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 area of graph coloring have been discussed in the literature (Golzari and Meybodi 2002; Akbari Torkestani and Meybodi 2009).

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-based algorithm is proposed in (Akbari Torkestani and Meybodi 2011). In this algorithm, the graphs to be colored are modeled by an ICLA. Each vertex in the graph corresponds to an ICLA cell, and each cell is equipped with an LA. The activation of the LA is automatic. When activated, a color will be selected for the assigned cell. To find a suitable selection of colors, the LAs can cooperate with their neighboring LAs, located in the neighboring cells. If the number of the colors in a solution is larger than the number used in the best solution found so far, or if the solution is not allowed, the selected solution will be penalized. If none of these two conditions occur, the selected solution will be rewarded. When the algorithm reaches its end, the LAs learn how to choose colors such that the graph is entirely colored in a legal way with the smallest number of colors. Several other ICLA applications in the area of graph coloring are addressed in the literature (Enayatzare and Meybodi 2009; Enami Eraghi et al. 2009).

4 The proposed cellular adaptive PN based on LA

In this section, a cellular adaptive Petri net based on learning automata is proposed, called CAPN-LA. 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 relationships, and an APN-LA is assigned to each cell (Fig. 1). The neighborhood relationship between the 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 between these local algorithms specifies the behavior of the CAPN-LA.

The APN-LA assigned to 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

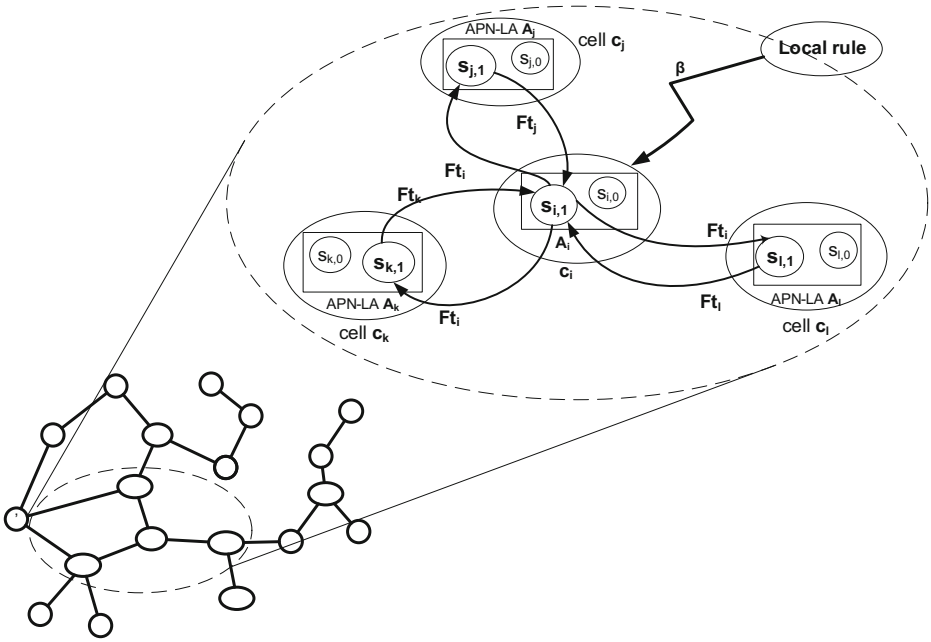


Fig. 1 The cellular adaptive Petri net based on learning automata (CAPN-LA)

automaton (LA) with a 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, from 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 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 the neighboring clusters of conflicting transitions, which in turn results in cooperation between neighboring APN-LAs. This provides a means of cooperation between local algorithms represented by neighboring APN-LAs. Formally, a CAPN-LA structure can be defined according to the following definition.

Definition 1: A cellular Adaptive Petri net based on Learning Automata (CAPN-LA) is a Structure $\mathcal{N} = \left(G < \mathbb{E}, V >, \{A_1, \ldots, A_n\}, \hat{F}\right)$ 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. $\{A_1, \ldots, A_n\}$ 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 $A_i = \left(\widehat{P}_i, \widehat{T}_i, \widehat{W}_i, S_i, LA_i\right)$ in which $\widehat{P}_i$ is the set of places, $\widehat{T}_i = T \cup \left\{t^u_{i,1}\right\}$ is a finite set of transitions and an updating transition $t^u_{i,1}$, $\widehat{W}_i$:

$\left(\left(\widehat{P}_i \times \widehat{T}_i\right) \cup \left(\widehat{T}_i \times \widehat{P}_i\right)\right) \rightarrow \mathbb{N}$ defines the interconnection of $\widehat{P}_i$ and $\widehat{T}_i$, LA_i is a learning automaton with a 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 $\widehat{T}_i$. No pair of transitions in this cluster has any input places in common, i.e., $\{\forall t_k \in s_{i,0}, \forall t_j \in s_{i,0} | \bullet t_k \cap \bullet t_j = \varnothing\}$. A random mechanism is used as the concurrent selector in this cluster.
ii. $s_{i,1}$ contains the remaining transitions in $\widehat{T}_i$ which are not in $s_{i,0}$. The set of transitions in 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. $\hat{F} = \{F_1, \ldots, F_n\}$ is a set of local rules which implies cooperation between clusters of conflicting transitions. $F_i : \underline{s}_{i,1} \rightarrow \beta_i$ is the local rule of cell c_i, where $\underline{s}_{i,1} = \{s_{j,1} | (i,j) \in E\} \cup \{s_{i,1}\}$ 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,
- $M_0 = [\mu_{1,0}, \ldots, \mu_{n,0}]$ 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 its constituent APN-LAs. The CAPN-LA is asynchronous, and each of its APN-LAs evolves independently from the others. The firing rules of the APN-LA A_i, assigned to cell c_i, in any marking M can be described as follows:

1. A transition $t \in \widehat{T}_i$ is said to be enabled if each input place p of t is marked with at least $\widehat{W}_i(p,t)$ tokens.
2. Only one enabled transition $t \in \widehat{T}_i$ can fire in each marking M. This transition is selected from a cluster of transitions referred to as the fired cluster.
3. 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, $P_{i,0} = \frac{|t' \in s_{i,0}, M[t'>|}{|t \in \widehat{T}_i, M[t>|}$ and $P_{i,1} = \frac{|t' \in s_{i,1}, M[t'>|}{|t \in \widehat{T}_i, M[t>|}$, where $|\ |$ stands for the 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. The firing of a transition $t \in \widehat{T}_i$ removes $\widehat{W}_l(p,t)$ tokens from each input place p of t, and adds $\widehat{W}_i(t,p)$ 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 $t^u_{i,1}$ fires. To generate the reinforcement signal, the last selected actions (the last fired transitions from $s_{i,1}$) of the neighboring APN-LAs are used.
7. Using β_i, LA_i updates its action probability vector using its learning algorithm.

It could be argued that the CAPN-LA is very similar to cellular learning automata (CLA); they both consist of cells, each with multiple neighboring cells, and each possess an LA. However, the main difference between these two cellular structures is their state spaces. In each cell of the CAPN-LA, there exists a Petri net, and thus the state space of the CAPN-LA is constructed by merging the markings of neighboring PNs. In contrast, in a CLA the state of a cell is equal to the state of its constituent LA.

5 Behavior of CAPN-LA

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

Definition 3: At a time instant k, the configuration of conflicting transitions 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 cluster $s_{i,1}$ and T denotes the transpose operator.

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

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

Definition 5: The evolution of conflicting transitions from a given initial configuration $\underline{q}(0) \in \mathcal{K}$is a sequence of configurations $\left\{\underline{q}(k)\right\}_{k \ge 0}$, such that $\underline{q}(k+1) = \mathcal{G}\left(\underline{q}(k)\right)$, 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 one which chooses each of its actions with equal probability. Therefore, for a pure-chance automaton with m_i-actions, the probability of j^{th}action $q_i^j, j = 1, \ldots, m_i$ is equal to $\frac{1}{m_i}$.

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

Definition 8: A CAPN-PCA is a CAPN-LA in which APN-PCAs are used 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 cell c_i, that is,

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

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

- $N_i^j, (1 \le j \le \mathcal{N}_i)$ *is the j^{th} neighboring cluster of cluster $s_{i,1}$,*
- Ft_i *denotes the last fired transition of 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 cluster $s_{i,1}$,*
- $q_i^j(k)$ *is the probability of choosing transition t_i^j from cluster $s_{i,1}$ for firing at time instant k,*
- *En(t, μ) denotes a Boolean function which indicates whether or not transition t is enabled in marking μ; it is defined in Eq. (5) as follows:*

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

- $\hat{q}_i^j(k,\mu)$ *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 Eq. (6). $\hat{q}_i^j(k,\mu) = 0$ when transition t_i^j is not enabled in marking μ.*

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

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

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

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 Eq. (7), $F_i\left(t_1, \ldots, t_{\mathcal{N}_i}; t_i^j\right)$, the local rule of cell c_i, generates the reinforcement signal $\beta_i(k)$ for a set of transitions $\left\{t_1, \ldots, t_{\mathcal{N}_i}; t_i^j\right\}$. This set consists of the 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 cluster $s_{i,1}$. The average reward of transition t_i^j is calculated by Eq. (7), a weighted sum of F_i in which the weights are the products of the probabilities of the firing transitions $\left\{t_1, \ldots, t_{\mathcal{N}_i}\right\}$. The average reward for all transitions in cluster $s_{i,1}$ is calculated for a marking set μ_i in Eq. (8). Note that by definition,

$\hat{q}_i^j(k,\mu)\neq 0$ only when transition t_i^j is enabled. In other words, the enabled transitions are only considered for calculating the average reward in cluster $s_{i,1}$.

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

Definition 11: The total average reward for the CAPN-LA for 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 concept of learning (Esnaashari and Meybodi 2015). Any model that is said to learn must at least do better than its equivalent pure-chance model. In the rest of this section, we define the concept of expediency for the CAPN-LA.

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

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

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 constituent *APN-LAs*.

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

Proof: It must be shown that a 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.

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

Lemma 1: *Consider a probabilistic configuration* $\underline{q}\in\mathcal{K}$ *which evolves using a* 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: These have both been proven in (Esnaashari and Meybodi 2015), in which $\underline{q}(k)$ represents a probabilistic matrix constructed by the action probability vector of a team of LAs at time instant k. These LAs form irregular cellular learning automata and all of them evolve with learning algorithm SL_{R-P}.

Lemma 2: *Consider a probabilistic configuration $\underline{q}\in\mathcal{K}$ which evolves using a SL_{R-P} learning algorithm. Let $q_i^{j^*}$ be the selection probability of transition t_i^j in cluster $s_{i,1}$ in the long run and let $d_i^j\left(\underline{q}^*\right)$ be the reward value generated by Eq. (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 is shown in (Beigy and Meybodi 2006) that the value of $q_i^{j^*}$ is inversely proportional to the penalty received from the environment, denoted by $rp_i^j\left(\underline{q}^*\right)$, that is, $q_i^{j^*}\propto\frac{1}{rp_i^j\left(\underline{q}^*\right)}$. Since the reward signal $d_i^j\left(\underline{q}^*\right)$ is equal to $1-rp_i^j\left(\underline{q}^*\right)$, we get

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

In Eq. (12), if the value of $d_i^j\left(\underline{q}^*\right)$ is increased, the denominator decreases; 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.

Taking Lemma 1 into consideration, it is only required to show that the inequality in (11) holds. Using the average reward definition in Eq. (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 $d_i^j(q(k))$, $\hat{q}_i^j(k)$, 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 Eq. (7), $\lim_{k\to\infty}E\left[d_i^j\left(\underline{q}(k)\right)\right]$ can be computed as follows:

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

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 is equal to $\frac{1}{\mathrm{ED}\left(\mu,s_{i,1}\right)}$, whereas in the left-hand-side convex combination, the weight of each $d_i^j\left(\underline{q}^*\right)$ equals $\hat{q}_i^{j*}$. It was shown that in Lemma 2, the value of q_i^{j*} is proportional to the value of $d_i^j\left(\underline{q}^*\right)$, and therefore $\hat{q}_i^{j*}$is also proportional to $d_i^j\left(\underline{q}^*\right)$. 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 that on the left-hand side, and thus the theorem is proved. ■

5.1 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 a measure of expediency, in terms of the firing probabilities of the transitions in CAPN-LA.

Definition 14: A measure of expediency is defined according to Eq. (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 a pure-chance automaton; 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.

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

6 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, referred to as APN-LA-VC, and then its behavior will be analyzed. Next, various algorithms will be presented for solving the VC problem using the replication of the APN-LA-VC at all vertices of the graph $\mathbb{G}$ (cells of CAPN-LA) to be colored. The structure generated is denoted by CAPN-LA-VC. Finally, multiple simulation results will be presented.

The model used for experiments is thoroughly described in Sections 6.2.1 and 6.2.2. Since the CAPN-LA consists of a number of APN-LAs, we first describe these APN-LAs, referred to as APN-LA-VCs, in Section 6.2.1, and then define the CAPN-LA-VC as a graph of APN-LA-VCs in Section 6.2.2. The input data used for the experiments is a subset of hard-to-color benchmarks reported in DIMACS (Ftp://dimacs.rutgers.edu/pub/challenge/graph/). All simulations were implemented by MATLAB 8.1 and were run on a Pentium V Core 2 Duo 2.0 GHz. Simulation parameters were set as follows: $a = b = .1$, i.e., the reward and penalty parameters, thresholds ε and τ are equal to .009 and.95 respectively. These are described in more detail later in this section.

6.1 The VC problem in a graph

The vertex coloring problem is a well-known coloring problem (Jensen and Toft 1994; Akbari Torkestani 2009) 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 the assignment of 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 the 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 $\mathbb{G}$, $\mathbb{C} = \{ \quad_1, \ldots, \quad_m\}$ 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. The VC problem can be considered a decision-making problem that aims to decide for a given graph whether or not the graph is P-colorable, and is known as the 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., $\mathcal{X}(\mathbb{G}) \leq \Delta + 1$.

6.2 *Solving the VC Problem with a* CAPN-LA

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

6.2.1 The APN-LA-VC

Each vertex v_i of the graph $\mathbb{G}$ is modeled by the simple PN depicted in Fig. 2. In this PN, $P_{i,0}$ is the decision-making place for the color which must be assigned to v_i, and each place $P_{i,j} \in \{P_{i,1}, \ldots, P_{i,m}\}$ represents a possible color for this vertex. Using the construction procedure of the APN-LA (Vahidipour et al. 2015), this PN is converted into an APN-LA called an APN-LA-VC, as shown in Fig. 3. Note that the set $\{t_{i,1}, \ldots, t_{i,m}\}$ 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; as a consequence, it is responsible for determining the color of the vertex v_i. This is done as follows: the set of actions of LA_i,referred to as $\underline{a}_i$,

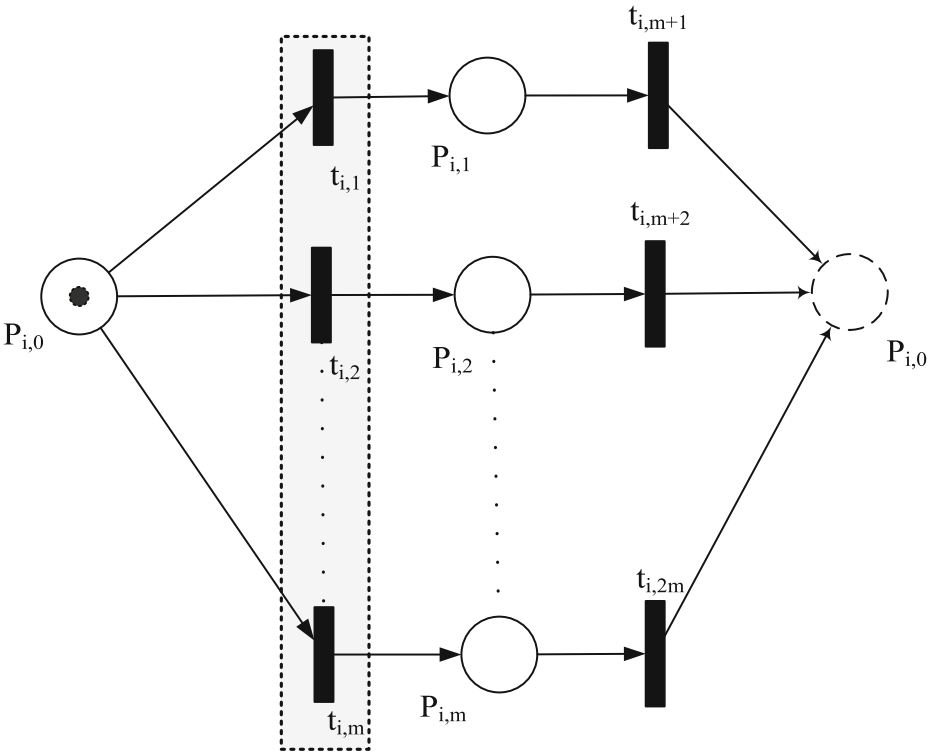


Fig. 2 A simple PN for vertex v_i. For simplicity, $P_{i,0}$ has been duplicated (indicated by the *dashed line*)

contains m actions. When $t_{i,r}$ is activated, it selects one of its actions, say $a_{i,r}$, according to its action probability vector. As a result, transition $t_{i,r}$, corresponding to the selected action, is fired and a token appears in place $p_{i,r}$. In this way, the APN-LA-VC assigns a color to the vertex.

6.2.2 The CAPN-LA-VC

A CAPN-LA-VC is a graph of a number of APN-LA-VCs such that an APN-LA-VC is assigned to each vertex of the graph. The CAPN-LA-VC system is defined as follows:

- $\mathbb{G} =< V, \mathbb{E} >$ is an undirected graph which is to be colored.
- $\{A_1, \ldots, A_n\}$ is a finite set of APN-LA-VCs, as shown in Fig. 3.
- $M_0 = [\mu_{1,0}, \ldots, \mu_{n,0}]$ is the initial marking vector of $\mathcal{N}$, where $u_{i,0} = p_{i,0} + p_{i,m+2}$ is the initial marking of A_i.
- $\hat{F} = \{F_1, \cdots, F_n\}$ is the set of local rules F_i, the local rule related to LA_i, is executed upon the firing of $t^{u}_{t,1}$ *in* A_i in 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 $s_{i,1}$ is different from the last selected transitions of all of its neighbors, then it is rewarded.
b. Otherwise, it is penalized.

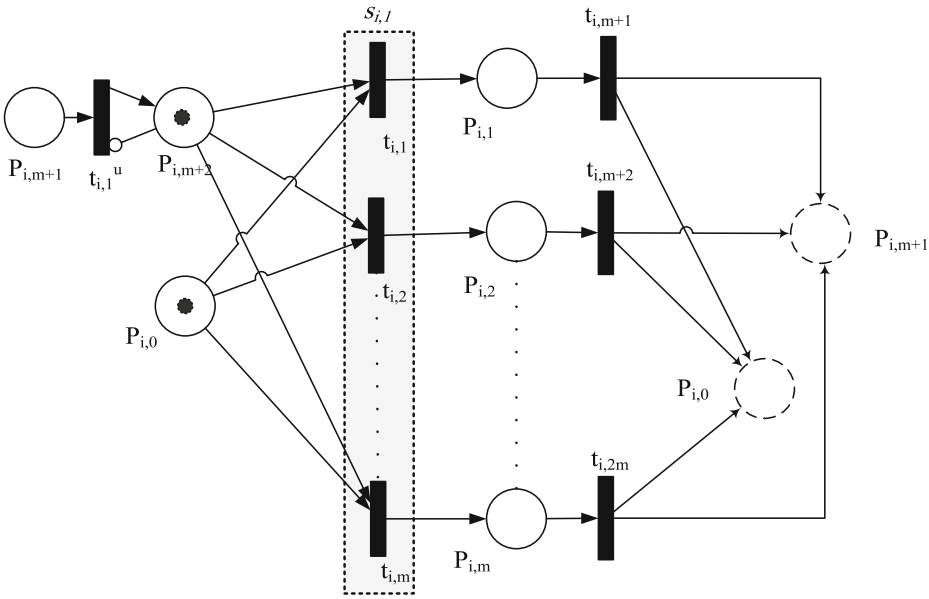


Fig. 3 An APN-LA for vertex v_i. For simplicity, $P_{i,0}$ and $P_{i,m+1}$ have been duplicated (indicated by the *dashed line*)

It should be noted that the local rule stated above is only an example; in a real scenario, any other local rule can be used instead.

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

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

A solution of the VC problem is obtained from the configuration of the conflict resolvers in the CAPN-LA-VC (refer to Definition 3). To this end, the *max* operator is used for 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 these algorithms with each other, an evaluative measure is required. For this purpose, we use the measure of expediency, as defined in Section 5.1). In the next subsection, this measure will be calculated for the CAPN-LA-VC using the results of the analysis of the proposed APN-LA-VC.

6.3 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 Eq. (8).

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

Note that according to Eq. (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 at the vertex $v_i(D_i)$ due to the differences in their local rules.

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

We first define a number of notations which are used in the rest of this section:

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

Figure 4 shows the reachability graph of A_i (Fig. 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 $\sigma_{i,j}$, $(j = 1, \ldots, m)$ states that the color j is assigned to vertex v_i.

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

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

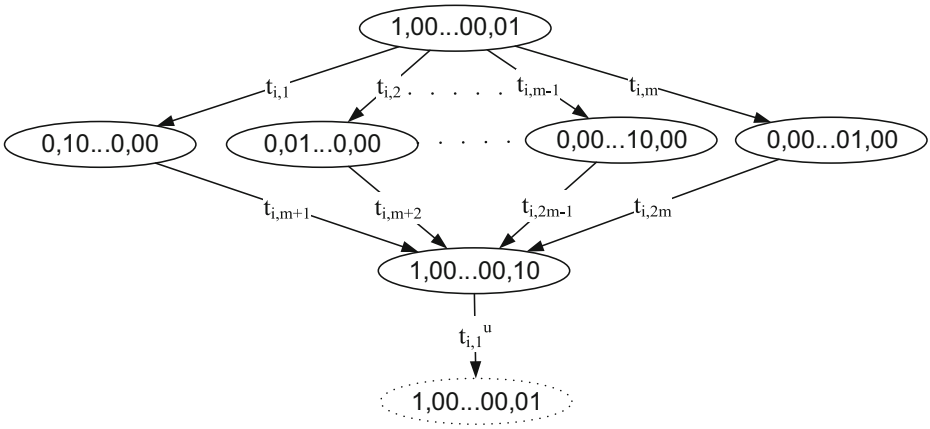


Fig. 4 The reachability graph of APN-LA-VC A_i . For simplicity, $\mu_{i,0}$ has been duplicated (indicated by the *dashed line*)

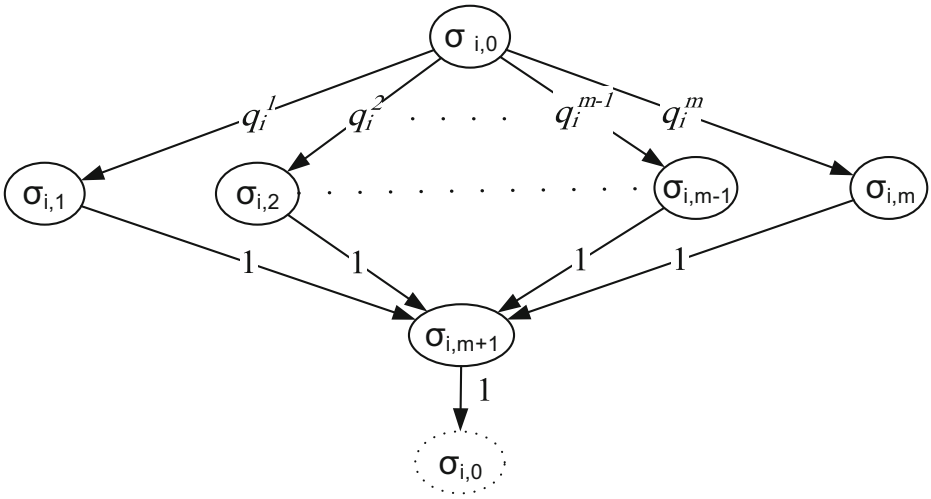


Fig. 5 The DTMC of APN-LA-VCA_i. q_i^j is the probability of assigning color j to vertex v_i. For simplicity, $\sigma_{i,0}$ has been duplicated (indicated by the *dashed line*)

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 $\sigma \in \{\sigma_{i,1}, \ldots, \sigma_{i,m}\}$, we can rewrite Eq. (21) as Eq. (22).

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

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

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

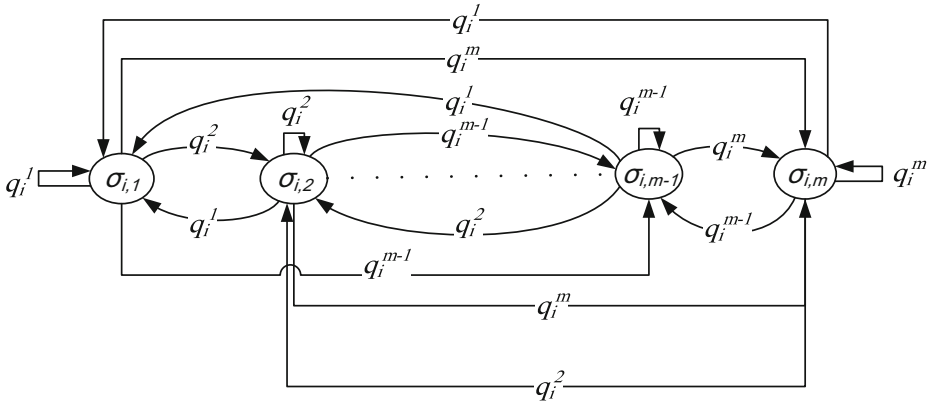


Fig. 6 Reduced DTMC for A_i

According to the above equation, the measure of expediency for the CAPN-LA-VC at time instant k is defined in Eq. (24).

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

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

6.4 A new algorithm for solving the VC problem

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

The states of the DTMC shown in Fig. 6 can be divided into two disjoint sets: 1) the set of allowable coloring states ($Y_{i,1}$); and 2) the set of forbidden 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, while $Y_{i,2}$ 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 Fig. 6 is reduced to a new DTMC with two states, as shown in Fig. 7. In the reduced DTMC, the transition-probability matrix $\mathbb{P}_i$ is as follows:

$$\begin{aligned} P_i &= \begin{bmatrix} Z_{i,2} & Z_{i,1} \\ Z_{i,2} & Z_{i,1} \end{bmatrix} \\ Z_{i,1} &= \sum_{S_{i,j} \in Y_{i,2}} q_i^j, Z_{i,2} = \sum_{S_{i,j} \in Y_{i,1}} q_i^j. \end{aligned} \tag{25}$$

The probability of being in 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 at the vertex v_i at k. This probability can be stated using the following equation:

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

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

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

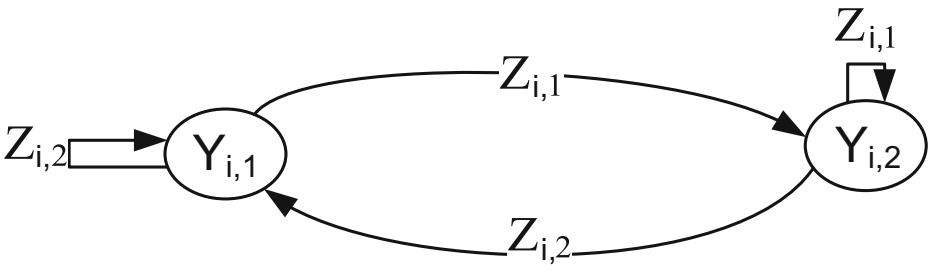


Fig. 7 Reduced DTMC for A_i

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

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

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 $\omega_i(\mathrm{k}) \geq \hat{\omega}_i(\mathrm{k})$, where $\hat{\omega}_i(\mathrm{k})$ is defined as $\hat{\omega}_i(\mathrm{k}) = \frac{1}{k}\sum_{j=1}^{k}\omega_i(\mathrm{j})$.
- 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 a better ω_i than the average of ω_i resulting from previous selections of this LA.

6.5 Simulation results

In this section, it will be shown that the CAPN-LA-VC can represent different algorithms for solving the VC problem. To compare these algorithms, two criteria introduced in (Akbari Torkestani 2013) are used. Furthermore, the proposed measure of expediency is used as a third criterion to compare the different VC algorithms. Therefore, the VC 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 of the learning automata, up to time instant $\mathbb{k}$, where $\mathbb{k}$ is defined as the smallest time instant at which the colors of all vertices in the graph are fixed,
- $M_E(\mathbb{k})$: The value of the measure of expediency defined according to Eq. (24).

To study the efficiency of these algorithms, a number of simulation studies have been conducted on a subset of hard-to-color benchmarks, as reported in DIMACS (Ftp://dimacs.rutgers.edu/pub/challenge/graph/). The rest of this section is divided into two sections:

– In the first experiment, the APN-ICLA-VC is used to implement and compare five different VC algorithms. The first four algorithms are state-of-the-art algorithms introduced in (Akbari Torkestani 2009; Akbari Torkestani 2013); in each of these, an ICLA is used to solve the VC problem. The fifth algorithm is proposed in this paper by introducing a novel local rule.
– The second experiment compares two scenarios of cooperation between APN-LAs for solving the VC problem.

6.5.1 Experiment one: Simulation of the CAPN-LA-VC

This section describes the four different algorithms introduced in (Akbari Torkestani 2009; Akbari Torkestani 2013), referred to here as VC1 to VC4, and a new algorithm proposed in this paper, referred to as VC5.

VC1: In the first algorithm, which we call VC1, it is assumed that the number of available colors for each vertex v_i or equivalently, the number of actions in the set of actions of LA_i, is equal to the number of vertices in the graph $\mathbb{G}$; that is, $m = |\underline{\alpha_i}| = |V|$. In VC1, when LA_i is activated, an action is selected from its set of actions. 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 the 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; this in turn prolongs the total running time of the algorithm. On the other hand, as described above, 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, the number of available actions in VC2 for each LA is equal to $\Delta + 1$. Due to the reduction in the number of actions of each LA, it is expected that VC2 will color the graph in less time and with a smaller number of colors compared to VC1.

VC3: In VC2, all LAs have the same number of actions. However, 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 the VC problem needs to decrease the number of colors required to color the graph, an enhanced algorithm will reduce the number of actions available to each LA where possible. In this enhanced algorithm, called VC4, learning automata with varying numbers of available action sets are used, as introduced in (Zhang et al. 2014); 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 the above algorithms, i.e., VC1 to VC4, use a simple local rule defined for VC1. In this paper, we propose a new local rule in Section 6.4 above. Here, the VC4 algorithm using this proposed local rule is referred to as VC5.

The termination condition of all the aforementioned algorithms at 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 a legal coloring of the graph. To eliminate the problem of incorrect coloring, the following modification is applied to all of the mentioned algorithms: all LAs are considered to be learning automata with varying numbers of available action sets. The actions available to 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 $\boldsymbol{k}$ is increased whenever the LA corresponding to the vertex with the maximum degree is updated. Simulation parameters are set as follows: $\boldsymbol{a} = \boldsymbol{b} = .1$, $\varepsilon = .009$, and $\tau = .95$. The results of this simulation study, which are reported in Table 1, indicate that:

- In moving from VC1 to VC4, the *CN* and $\mathcal{U}$ criteria decrease and the $M_E(\mathbb{k})$ 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 actions available to each learning automaton, its convergence rate and its accuracy increase.
- VC5 is always better or at least equal to VC4 in in terms of all the above mentioned criteria. This efficiency is the result of considering the proposed measure ω_i in the local rule of the CAPN-LA-VC system.

6.5.2 Experiment two: Cooperation scenarios

A simulation study was conducted to compare the following two scenarios: 1) handling the required cooperation between the neighboring cells within the controlling mechanisms; and 2) handling the required cooperation between the neighboring cells within the structure of the Petri net. To this end, a Petri net consisting of multiple APN-LAs is 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 approach is to use inhibitor arcs. An inhibitor arc disables the use of a color for a vertex when one of its neighbors uses this for coloring. Therefore, to solve the VC problem, the APN-LAs of the neighboring vertices are connected together by inhibitor arcs in order to disable the use of illegal colors in the coloring of vertices. Fig. 8 illustrates an APN-LA with inhibitor arcs corresponding to the vertex v_i. The APN-LA presented 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 hereafter as APN-LA-IA.

The number of inhibitor arcs required in the APN-LA-IA for coloring the graph $\mathbb{G}$ with n vertices is equal to $\sum_{i=1}^{n} m_i \times \Delta_i$ where m_i is the number of available colors for vertex v_i and Δ_i is the degree of vertex v_i. Although the use of these inhibitor arcs decreases the number of markings in the state-space of APN-LA-IA, it increases the complexity of the structure of the APN-LA-IA. To generate a legal coloring by APN-LA-IA, the lowest priority is considered for transitions $t_{i,m+1}, \ldots, t_{i,2m}, (i = 1, \ldots, n)$. In this way, the

Table 1 Results of the experiment for proposed algorithms represented by the CAPN-LA-VC

Algorithm	VC1			VC2			VC3			VC4			VC5		
Graph	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	1,044,679	.439	5	116,002	.561	5	35,679	.801	5	26,876	.842	5	12,593	.897
DSJC125_5	19	1,067,235	.494	18	124,642	.558	17	46,410	.760	17	39,055	.827	17	27,263	.851
DSJC125_9	48	1,308,101	.486	45	128,119	.579	44	55,224	.758	44	46,407	.802	44	32,004	.827
DSJC250_1	10	641,605	.563	8	84,145	.689	8	28,757	.752	8	24,925	.857	8	14,661	.874
DSJC250_5	31	4,418,776	.532	29	421,963	.672	28	130,756	.739	28	100,635	.838	28	45,645	.839
DSJC250_9	76	6,741,839	.510	75	626,042	.671	73	233,568	.699	72	150,091	.833	72	59,200	.847
DSJC500_1	14	1,110,579	.502	13	144,954	.602	12	86,799	.721	12	67,603	.870	12	50,270	.889
DSJC500_5	52	2,712,399	.492	51	345,971	.588	49	222,161	.706	49	156,358	.813	49	83,597	.814
DSJC500_9	139	6,735,701	.482	131	648,780	.578	128	324,640	.685	127	285,403	.773	127	158,908	.782
le450_15a	15	4,279,467	.561	15	480,839	.687	15	162,445	.775	15	107,575	.892	15	78,335	.910
le450_15b	15	4,629,513	.556	15	497,638	.635	15	183,004	.734	15	138,001	.798	15	59,340	.864
le450_15c	16	4,672,330	.562	15	589,748	.663	15	311,569	.762	15	265,139	.815	15	102,384	.849
le450_15d	17	5,284,660	.585	15	502,709	.663	15	299,948	.771	15	203,358	.809	15	11,965	.831
le450_25c	26	5,516,730	.469	26	621,156	.607	25	313,236	.790	25	255,099	.825	25	103,542	.830
le450_25d	27	5,737,808	.448	26	636,840	.592	25	334,672	.699	25	275,394	.786	25	157,722	.798

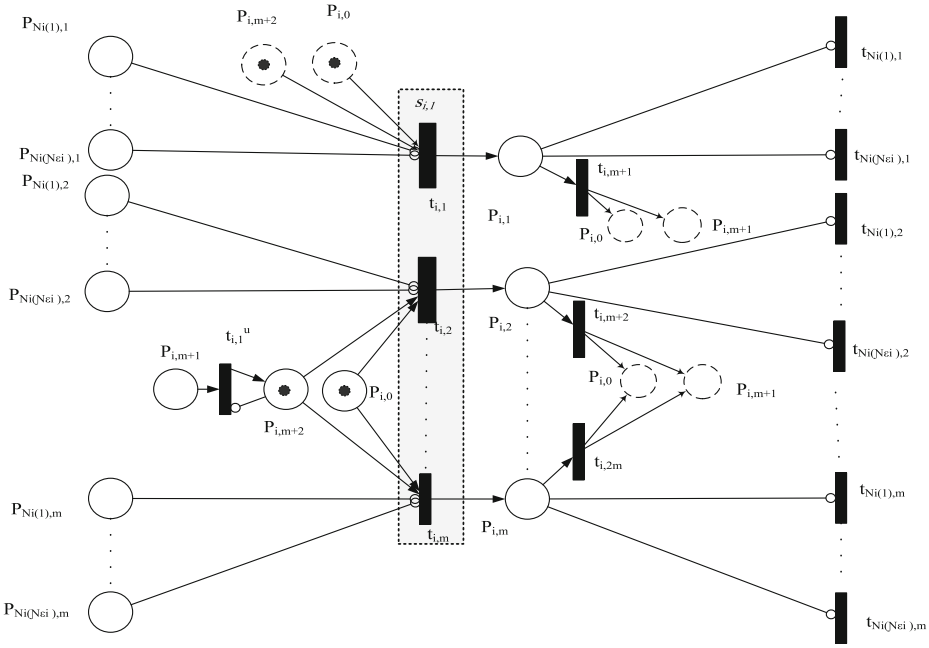


Fig. 8 An APN-LA corresponding to vertex v_i in which inhibitor arcs are used to disable illegal coloring of the neighboring vertices. The vertex v_i has $|NE_i|$ neighbors indicated by $N_i(1), \ldots, N_i(\mathcal{NE}_i)$. For simplicity, several places have been duplicated (indicated by the *dashed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

color assigned to each vertex (i.e., the token appearing in place $P_{i,1}|P_{i,2}| \ldots |P_{i,m}$) stays the same until the next color is assigned to the vertex. The algorithm represented by APN-LA-IA is referred to as VC6. In VC6, the number of actions for each vertex v_i is equal to $\Delta_i + 1$ where Δ_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 the CN, $\mathcal{U}$, and $M_E(\mathbb{K})$ criteria. In addition, this table also reports the required number of inhibitor arcs for representing the 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 Δ_i and the number of possible colors for vertex v_i is $m_i = \Delta_i + 1$. Regarding the size of the PNs (in terms of the number of places, transitions, arcs, and inhibitor arcs) used in these two scenarios, the following points can be made:

- In the CAPN-LA-VC, for each vertex v_i, there exists a Petri net (Fig. 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 in total $|\mathbb{E}| + 4n$ 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 (Fig. 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.

Table 2 Results of the experiment for VC6 represented by the APN-LA-IA. #IA is the required number of inhibitor arcs

Algorithm	VC5				VC6			
Graph	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	#IA	CN	$\mathcal{U}$	$M_E(\mathbb{K})$	#IA
DSJC125_1	5	12,593	.897	125	5	35,722	.782	20,279
DSJC125_5	17	27,263	.851	125	18	46,415	.745	495,871
DSJC125_9	44	32,004	.827	125	44	55,231	.756	1,565,939
DSJC250_1	8	14,661	.874	250	8	28,765	.750	178,900
DSJC250_5	28	45,645	.839	250	28	130,771	.734	3,974,526
DSJC250_9	72	59,200	.847	250	73	233,584	.698	1,253,204
DSJC500_1	12	50,270	.889	500	12	86,814	.719	1,289,234
DSJC500_5	49	83,597	.814	500	49	222,180	.704	31,560,758
DSJC500_9	127	158,908	.782	500	128	324,658	.685	101,382,226
le450_15a	15	78,335	.910	450	15	162,443	.777	737,068
le450_15b	15	59,340	.864	450	15	182,986	.736	730,290
le450_15c	15	102,384	.849	450	15	311,587	.761	2,709,830
le450_15d	15	119,652	.831	450	16	299,983	.767	2,730,334
le450_25c	25	103,542	.830	450	26	313,249	.773	3,091,758
le450_25d	25	157,722	.798	450	26	334,671	.699	3,098,528

From Table 2, the following conclusions can be drawn:

- VC5 outperforms VC6 in terms of the *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 the Petri net is decreased.

7 Conclusion

In this paper, a cellular adaptive Petri net based on learning automata (CAPN-LA) is proposed. The CAPN-LA consists of a graph and a number of APN-LAs, in which each APN-LA is 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 used to resolve 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 to represent a local algorithm to be executed at each vertex of the graph. This APN-LA, called APN-LA-VC, was then replicated at 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 presented. In addition, a new algorithm was also proposed which attempts 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, a measure of expediency is proposed for comparing these algorithms.

Based on this measure, a number of computer simulations were conducted and the algorithms were compared. The results of the simulations indicate that the proposed algorithm is more expedient than the state-of-the-art algorithms and is able to color graphs using a lower number of iterations and fewer of colors. In future work, we will seek to define the notion of absolute expediency and the conditions under which the CAPN-LA becomes absolutely expedient.

Acknowledgment This research was in part supported by a grant from IPM. (No. CS1396-4-21).

References

Akbari Torkestani J (2009) Channel assignment and multicast routing in mobile ad hoc networks based on learning automata, PhD dissertation, Dep. of Eng., Univ. Islamic Azad, Iran

Akbari Torkestani J (2013) A new approach to the vertex coloring problem. Cybern Syst 44(5):444–466

Akbari Torkestani J, Meybodi MR (2009) "Graph coloring problem based on learning automata", Proceedings of International Conference on Information Management and Engineering, Kuala-Lumpur, Malaysia, pp. 718–722

Akbari Torkestani J, Meybodi MR (2010) A new vertex coloring algorithm based on variable action-set learning automata. Journal of Computing and Informatics 29(3):1001–1020

Akbari Torkestani J, Meybodi MR (2011) A cellular learning automata based algorithm for solving the vertex coloring problem. Expert Syst Appl 38(8):9237–9247

Asmuni H, Burke EK, Garibaldi JM, McCollum B, Parkes AJ (2009) An investigation of fuzzy multiple heuristic orderings in the construction of University examination timetables. Comput Oper Res 36:981–1001

Barnier N, Brisset P (2002) Graph coloring for air traffic flow management. Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques, Le Croisic, France:133–147

Bause F (1996) On the analysis of Petri net with static priorities. Acta Informatica 33:669–685

Bause F (1997) Analysis of Petri nets with a dynamic priority method. Application and Theory of Petri Nets 1248:215–234

Beigy H, Meybodi MR (2006) Utilizing distributed learning automata to solve stochastic shortest path problem. International Journal of Uncertainty, Fuzziness and Knowledge- based Systems 14(5):591–617

Billard EA (1996) Stability of adaptive search in multi-level games under delayed information. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 26:231–240

Billard EA (1997) "Chaotic behavior of learning automata in multi-level games under DelayedInformation", Proc. of IEEE Intl. Conf. on Systems, Man, and Cybernetics, pp. 1412–1417

Burkhard HD (1981) Ordered firing in Petri nets. EIK (Journal of information processing and cybernetics) 17(2/3):71–86

Carter M, Laporte G, Lee S (1996) Examination timetabling: algorithmic strategies and applications. J Oper Res Soc 47:373–383

Chiola G, Ferscha A (1993) Distributed simulation of Petri nets. IEEE Concurr 3:33–50

Ding Z, Zhou Y, Zhou M (2016) Modeling self-adaptive software systems with learning Petri nets. IEEE Transactions on Systems, Man, and Cybernetics: Systems 46(4):483–498

Enami Eraghi A, Akbari Torkestani J, Meybodi MR (2009) "Cellular learning automata-based graph coloring problem", Proceedings of 2009 International Conference on Machine Learning and Computing. Perth, Australia, pp 163–167

Enayatzare M, Meybodi MR (March, 2009) "Solving graph coloring problem using cellular learning automata", Proceedings of 14th Annual CSI Computer Conference of Iran. Amirkabir University of Technology, Tehran, Iran

Esnaashari M, Meybodi MR (2015) Irregular cellular learning automata. IEEE Transactions on Cybernetics 45(8):1622–1632

Fujimoto RM (2001) Parallel simulation: parallel and distributed simulation systems. In: Proceedings of the 33nd conference on winter simulation, IEEE Computer Society, pp. 147–157

Gao M, Zhou M, Huang X, Wu Z (2003) Fuzzy reasoning Petri nets. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 33(3):314–324 May 2003

Ghavipour M, Meybodi MR (2016) An adaptive fuzzy recommender system based on learning automata. Electron Commer Res Appl 20:105–115

Golzari Sh, Meybodi MR (2002) "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
Hashemi B, Meybodi MR (2009) Cellular Pso: a Pso for dynamic environments In: Advances in Computation and Intelligence, Lecture Notes in Computer Science, vol. 5821/2009, pp. 422–433
Holloway LE, Krogh BH (1994) Controlled Petri nets: a tutorial survey. In: The 11th international conference on analysis and optimization of systems discrete event systems. Springer, Berlin Heidelberg, pp 158–168
Jensen TR, Toft B (1994) Graph coloring problems. Wiley, USA
Karp RM (1972) Reducibility among combinatorial problems. Complexity of Computer Computations, Plenum Press, USA:85–103
Lewis R, Paechter B (2007) Finding feasible timetables using group based operators. IEEE Trans Evol Comput 11(3):397–413
Linial N (1992) Locality in distributed graph algorithms. SIAM J Comput 21(1):193–201
Mabrouk BB, Hasni H, Mahjoub Z (2009) On a parallel genetic-tabu search based algorithm for solving the graph coloring problem. Eur J Oper Res 197:1192–1201
Malaguti E, Toth P (2008) An evolutionary approach for bandwidth multi-coloring problems. Eur J Oper Res 189:638–651
Meybodi MR, Beigy H, Taherkhani M (2003) Cellular learning automata and its applications. Sharif Journal of Science and Technology 19(25):54–77
Mollakhalili Meybodi MR, Meybodi MR (2014) "Extended distributed learning automata: an automata-based framework for solving stochastic graph optimization problems", Appl Intell, vol. 41, vo. 3, pp. 923–940
Murata T (1989) "Petri nets: properties, analysis and applications", Proc IEEE, vo1.77, pp. 541–580
Narendra KS, Thathachar MAL (1989) Learning automata: an introduction. Prentice-Hall, Inc., Englewood cliffs, New Jersey, USA
Peterson JL (1981) Petri net theory and the modeling of systems. Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA
Rastegar R, Arasteh AR, Harriri A, and Meybodi MR (2004) A fuzzy clustering algorithm using cellular learning automata based evolutionary algorithm. In: Proc. of the Fourth Intl. Conf. on Hybrid Intelligent Systems (HIS04), Japan, Kitakyushu, pp. 310–314
Reisig W (2013a) Understanding Petri Nets: modeling techniques, Analysis methods, case studies, Springer Science & Business. Springer Publishing Company, Springer-Verlag Berlin Heidelberg
Reisig W (2013b) The synthesis problem. In: Jensen K, van der Aalst WMP, Balbo G, Koutny M, Wolf K (eds) Transactions on Petri nets and other models of concurrency VII. Springer, Berlin Heidelberg, pp 300–313
Rezvanian A, Meybodi MR (2016) Stochastic graph as a model for social networks. Comput Hum Behav 64:621–640
Talavan PM, Yanez J (2008) The graph coloring problem: a neuronal network approach. Eur J Oper Res 191:100–111
Terán-Villanueva JD et al (2013) Cellular processing algorithms. In: Soft computing applications in optimization, control, and recognition. Springer, Berlin Heidelberg, pp 53–74
Thathachar MAL, Satstry PS (1997) A hierarchical system of learning automata that can learn the globally optimal path. Inf Sci 42(2):743–166
Thathacher MA, Harita BR (1987) Learning automata with changing number of actions. IEEE Transactions on Systems, Man and Cybernetics 17(6):1095–1100
Vafashoar R, Meybodi MR, and Momeni AH (2012) "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
Vahidipour SM, Meybodi MR (2013) "Adaptation in priority Petri net using learning automata".In: Proceedings of 11th Iranian Conference on Intelligent Systems, Kharazmi University, Tehran, Iran, February 27–28
Vahidipour SM, Meybodi MR, Esnaashari M (2015) 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 45(10):1373–1384
Williams RJ (1988) Toward a theory of reinforcement learning connectionist systems, Technical Report, Nu-ccs-88-3. Northeastern University, Boston
Zhang J, Wang C, Zhou MC (2014) Last-position elimination-based learning automata. Cybernetics, IEEE Transactions 44(12):2484–2492 Dec. 2014
Zhang J, Wang C, Zhou M (2015) Fast and epsilon-optimal discretized pursuit learning automata. IEEE transactions on cybernetics 45(10):2089–2099
Zhang J, Wang C, Zang D, Zhou M (2016) Incorporation of optimal computing budget allocation for ordinal optimization into learning automata. IEEE Trans Autom Sci Eng 13(2):1008–1017
Zhou MC, Venkatesh K (1998) Modeling, simulation and control of flexible manufacturing systems: a Petri net approach. World Scientific, Singapore
Zymolka AM, Koster CA, Wessaly R (2003) "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

S. Mehdi Vahidipour received the MSc degree in computer engineering from University of Shiraz, Shiraz, Iran, in 2003 and the Ph. D. degree in computer engineering from Amirkabir University of Technology, Tehran, Iran, in 2016. He is currently an Assistant Professor with University of Kashan, Kashan, 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 K. N. Toosi University, Tehran, Iran. His current research interests include computer networks, learning systems, soft computing, and information retrieval.