Engineering Applications of Artificial Intelligence 77 (2019) 255–267

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

GAPN-LA: A framework for solving graph problems using Petri nets and learning automata

S.M. Vahidipour [a,*], M. Esnaashari [b], A. Rezvanian [c], M.R. Meybodi [d]

[a] Faculty of Electrical and Computer Engineering, Department of Computer, University of Kashan, Kashan, Iran
[b] Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran, Iran
[c] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[d] Soft Computing Lab, Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran

ARTICLE INFO

Keywords:
Adaptive Petri net
Learning automata
Modular algorithms
Graph-based problems

ABSTRACT

A fusion of learning automata and Petri nets, referred to as APN-LA, has been recently introduced in the literature for achieving adaptive Petri nets. A number of extensions to this adaptive Petri net have also been introduced; together we name them the APN-LA family. Members of this family can be utilized for solving problems in the domain of graph problems; each member is suitable for a specific category within this domain. In this paper, we aim at generalizing this family into a single framework, called generalized APN-LA (GAPN-LA), which can be considered as a framework for solving graph-based problems. This framework is an adaptive Petri net, organized into a graph structure. Each place or transition in the underlying Petri net is mapped into exactly one vertex of the graph, and each vertex of the graph represents a part of the underlying Petri net. A vertex in GAPN-LA can be considered as a module, which, in cooperation with other modules in the framework, helps in solving the problem at hand. To elaborate the problem-solving capability of the GAPN-LA, several graph-based problems have been solved in this paper using the proposed framework.

1. Introduction

Petri nets (PNs) are graphical tools, useful for the formal description of systems whose dynamics are characterized by concurrency, synchronization, mutual exclusion and conflict, which are typical features of distributed environments. Petri nets are utilized to model cellular automata (Zaitsev, 2015a,b), grids (Zaitsev et al., 2014; Zaitsev, 2013), queuing systems (Vahidipour et al., 2015, 2017a,b,c; Vahidipour and Esnaashari, 2017), and so on. PNs incorporate a notion of a (distributed) state and a rule for changing states that allows them to capture both the static and the dynamic characteristics of a real system (Marsan et al., 1994).

A PN comprises three structural components: places, transitions and arcs. Places are used to describe possible local system states, which may be either conditions or situations. Transitions are used to describe events that may modify the system state. Arcs specify the relations between local states and events; they indicate the local state in which an event can occur, and the local state transformations induced by that event.

The evolution of a Petri net can be described in terms of its initial state and a number of firing rules (Murata, 1989). A state or marking of a PN is specified by tokens which are indistinguishable markers that reside in places. If a place describes a condition, that condition is true if a token is present in the place and is false otherwise. If a place defines a situation, the number of tokens present in the place is used to specify the situation. Firing rules define the marking modification induced by transition firings. A transition can fire only if all its input places contain at least one token. In this case, the transition is said to be enabled. The firing of an enabled transition removes one token from each of its input places and generates one token in each of its output places. More tokens are required/generated in an input/output place if the weight of its arc is greater than one. The firing of a transition is an atomic operation. Tokens are removed from input places and deposited into output places using one indivisible action (Marsan et al., 1994).

In the evolution of a PN, two or more enabled transitions may be in conflict; this occurs when the firing of one transition results in disabling the other(s). Thus, in a PN, a controlling mechanism is needed which specifies the order for the firing of transitions, so that probable conflicts among transitions will be resolved. Different controlling mechanisms have been proposed in the literature so far, such as random selection (Peterson, 1981), queue regimes (Burkhard, 1981), priority (Bause, 1996, 1997), external controllers (Holloway and Krogh, 1994), Sleptsov net (Zaitsev, 2016), and APN-LA (Vahidipour et al., 2015) to mention only a few. Among these mechanisms, only APN-LA uses an adaptive controlling mechanism.

* Corresponding author.
E-mail addresses: vahidipour@kashanu.ac.ir (S.M. Vahidipour), esnaashari@kntu.ac.ir (M. Esnaashari), a.rezvanian@aut.ac.ir (A. Rezvanian), mmeybodi@aut.ac.ir (M.R. Meybodi).

https://doi.org/10.1016/j.engappai.2018.10.013
Received 20 September 2016; Received in revised form 15 July 2018; Accepted 9 October 2018
Available online xxxx
0952-1976/

Adaptive Petri net based on learning automata (APN-LA) has been proposed by the fusion of learning automata and ordinary Petri nets. 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; Thathachar and Sastry, 2004). In the APN-LA, learning automata are used in a Petri net as a controlling mechanism: when two or more transitions are simultaneously enabled in a given marking, a decision is made by the LA and an enabled transition is selected for firing. APN-LA is the basic adaptive Petri net from which other adaptive Petri nets have been developed.

A generalization of this type of adaptive Petri net, referred to as ASPN-LA, has been introduced in Vahidipour et al. (2015) resulting from the fusion of learning automata with stochastic Petri nets (SPN). ASPN-LA has been used for representing and analyzing a number of priority-assignment algorithms in queuing systems. ASPN-LA is also used for analyzing algorithms in the shortest path problem in stochastic graphs (Vahidipour et al., 2017a).

Another adaptive PN based on APN-LA is APN-ICLA (APN based on irregular cellular LA) (Vahidipour et al., 2017b). The idea behind constructing APN-ICLA is that in some situations, having a means of cooperation between learning automata within a single Petri net may be beneficial. This cooperation is not available in APN-LA, as in APN-LA each learning automaton acts independently from the others. The APN-ICLA can be regarded as a Petri net in which the controlling mechanism is cooperative.

Yet another adaptive PN based on APN-LA is CAPN-LA (cellular APN-LA) (Vahidipour et al., 2017c). CAPN-LA can be utilized to represent cellular algorithms in which an identical algorithm must be executed in each cell and the solution of the problem is achieved via the cooperation of all such identical algorithms. CAPN-LA consists of a cellular structure and a number of identical APN-LAs; each APN-LA represents an algorithm which must be executed in each cell. The required cooperation between the neighboring cells in CAPN-LA is handled by means of cooperation between the APN-LAs in the corresponding cells. The idea behind CAPN-LA is to mitigate the fact that designing a single APN-LA for representing cellular algorithms is tedious work and results in a large and complex model. In Vahidipour et al. (2017c) CAPN-LA has been used for representing a number of cellular algorithms for solving the vertex coloring problem.

The family of APN-LAs introduced so far can be used for modeling and problem solving in different categories of graph-based problems; each member of the family is suitable for a specific category. For instance, CAPN-LA is suitable for representing the category of cellular applications, while ASPN-LA is better suited to the category of problems with time dependency, such as queuing systems. In this paper, we propose a generalization for the family of APN-LAs, referred to as a generalized adaptive PN based on learning automata (GAPN-LA), which possesses the problem-solving capabilities of all members of the family. Indeed, GAPN-LA can be considered as a generic framework for solving different categories of graph-based problems.

More formally, a GAPN-LA is a large Petri net which is mapped into a graph structure. Places and transitions of the Petri net are mapped into vertices of the graph. This graph provides a means of visualizing the neighborhood relationships between different parts of the Petri net. From a modular perspective, each part of the Petri net, mapped into a vertex in the GAPN-LA, can be considered as a module which performs a specific task. Thus, a problem can be solved as a result of different modules of the GAPN-LA performing different tasks. In this paper, a number of graph-based problems are solved using the proposed GAPN-LA.

The rest of the paper is organized as follows: Section 2 introduces the learning automata and reviews their applications on graph problems. Section 3 reviews the different kinds of adaptive Petri nets. In Section 4, the proposed GAPN-LA is described. In Section 5, GAPN-LA is proposed as a framework for solving graph problems. Section 6 includes the conclusion.

2. Learning automata and their applications in graph problems

In this section, first, a brief review of learning automata is presented. Then, the applications of learning automata in the graph domain are briefly reviewed.

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 (Tsetlin, 1973; 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 $E = \{\underline{\alpha}, \underline{\beta}, \underline{c}\}$, 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 the penalty probabilities, where the element c_i is associated with the given action α_i. In a stationary random environment, the value of $c_i, (i = 1, \ldots, m)$ 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 $(\underline{\alpha}, \underline{\beta}, \underline{q}, L)$, 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, shown by (1) and (2), is a linear learning algorithm which updates the action probability vector $\underline{q}$ using

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

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

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

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 $a(n) \gg b(n)$; (3) L_{R-I} in which $b(n) = 0$. The notation SL_{R-P} is used in this paper when LA uses the L_{R-P} learning algorithm and operates within an S-model environment.

At the time instant$k$, an LA operates as follows: (1) selects an action $\alpha(k)$ randomly based on $q(k)$; (2) performs $\alpha(k)$ on the environment and receives $\beta(k)$; and, (3) updates $q(k)$ using the learning algorithm L. A learning automaton is called LA with a variable set of actions, if the set of available actions for the LA can be varied over the time (Thathachar, 1987).

The LA is, by design, a simple unit by which simple decision makings can be performed. The full potential of the LA will be realized when a

cooperative effort is made by a set of interconnected LAs to achieve the group synergy. In other words, LA can be used as the building blocks of more complex learning models. These complex models include hierarchical system of LA (Thathachar and Sastry, 1987), distributed LA (DLA) (Beigy and Meybodi, 2006), extended DLA (EDLA) (Meybodi and Meybodi, 2014), network of LA (Williams, 1988), multi-level game of LA (Billard, 1996, 1997), cellular LA (CLA) (Beigy and Meybodi, 2004), CLA-based evolutionary computing (Rastegar et al., 2004), differential evolution-based LA (Mahdaviani et al., 2015), differential evolution-based CLA (Vafashoar et al., 2012), CLA-based particle swarm optimization (Hashemi and Meybodi, 2009), and Irregular CLA (ICLA) (Esnaashari and Meybodi, 2015).

2.2. LA applications in graph problems

In this section, we provide an overview of learning automata applications in graph problems. A learning automaton solution to the NP-hard graph partitioning problem was presented by Oommen et al. This problem involves partitioning the nodes of a graph G into K sets of equal size, to minimize the sum of the costs of the edges having endpoints in different sets (Oommen and de St Croix, 1996). Motevalian et al. proposed an algorithm using cellular learning automata for solving the maximum independent set problem. Their simulation results showed that their algorithm outperformed the genetic algorithm (Motevallian and Meybodi, 2005). Khomami et al. investigated an LA-based algorithm for finding the maximum independent set in the graph. Based on their simulation, their learning automaton-based algorithm outperforms some heuristic algorithms in finding the maximum independent set (Khomami et al., 2016). Beigy et al. introduced an algorithm based on learning automata for solving the graph isomorphism problem. Based on a simulation, they conjectured that their algorithm has an average time complexity of order $O(n^3)$ (Beigy and Meybodi, 2002). In Enayatzare and Meybodi (2008), Enayatzare et al. proposed an algorithm for solving the maximum cut problem in graphs using cellular learning automata. This algorithm also produces better results than genetic algorithms.

Akbari Torkestani et al. introduced some learning automata-based algorithms for solving the stochastic version of the minimum connected dominating set (MCDS) problem with weighted vertices. They also showed that for their algorithms, if the optimality of the solution is particularly important, the learning rate should be selected to be as small as possible (Torkestani and Meybodi, 2012a). The authors also solved another version of the MCDS problem, i.e., the minimum weakly connected dominating set (WCDS) problem in stochastic graphs, using several LA-based algorithms where the probability distribution function of the weight of the vertices is unknown (Torkestani and Meybodi, 2010). Motivated by a mobile ad hoc networks application, in Torkestani and Meybodi (2011a,b), the mobility-based multicast routing algorithms for wireless mobile ad hoc networks are modeled with a stochastic graph and a stochastic version of the minimum Steiner connected dominating set problem in weighted networks, in which the relative mobility of each host is considered as its weight is introduced. Subsequently, they proposed some learning automata-based approximation algorithms for finding a near-optimal solution to the minimum weighted Steiner connected dominating set problem.

Akbari et al. suggested an approximation algorithm for solving the minimum vertex coloring problem based on variable action-set learning automata. Their algorithm iteratively finds different possible colorings of the graph. The proposed algorithm guarantees that the graph can be legally colored at each iteration. As the proposed algorithm approaches the end, each vertex learns how to select one of its non-adjacent vertices (Torkestani and Meybodi, 2012c). In addition, in Akbari Torkestani and Meybodi (2011), an irregular cellular learning automata-based algorithm was proposed for finding a near-optimal solution to the vertex coloring problem. The proposed CLA-based coloring algorithm is a fully distributed algorithm in which each vertex chooses its optimal color based solely on the colors selected by its adjacent vertices.

Mirsaleh and Meybodi proposed a new CLA-based memetic algorithm for finding a near-optimal solution for the vertex coloring problem (Mirsaleh and Meybodi, 2016). Their proposed algorithm is based on the irregular open CLA (IOCLA), which is an extension of ICLA in which the evolution of CLA is influenced by local and global environments. Like other IOCLA-based algorithms, the local environment of the proposed algorithm is composed of the neighboring learning automata of any cell. On the other hand, the global environment consists of a pool of memes in which each meme corresponds to a certain local search method. Each meme is represented by a set of learning automata where the history of the corresponding local search method can be extracted. Each environment produces a signal, which is used by the local rule to generate a reinforcement signal to the learning automaton residing in every cell. All learning automata update their action probability vectors on the basis of the supplied reinforcement signal generated by the local rule. This process continues until the desired result is obtained.

In some studies, a stochastic version of the shortest path problem has been investigated. Beigy et al. first introduced distributed learning automata and then proposed some iterative algorithms based on distributed learning automata for solving the stochastic shortest path problem. In these algorithms, at each stage, distributed learning automata determine which edges are to be visited and finally find a policy that determines a path from a source node to a destination node with minimal expected cost (Beigy and Meybodi, 2006). Misra et al. presented an LA-based algorithm for the dynamic all-pairs shortest path problem in stochastic environments, where edge weights change stochastically and where graph topologies undergo multiple simultaneous edge-weight updates. They reported their simulation for different graph topologies with varying sizes and edge weights (Misra and Oommen, 2009). A learning automata-based heuristic algorithm for solving the minimum spanning tree problem in a stochastic graph was proposed by Akbari et al. (Torkestani and Meybodi, 2012b). In their algorithm, at each stage, the edges that must be visited are determined by the learning automata assigned to the graph vertices. In Torkestani (2012), the stochastic degree-constrained minimum spanning tree problem was introduced, and an LA-based algorithm was proposed to solve it. The authors claimed that their algorithm concentrated and approached the edges of the minimum-cost degree-constrained spanning tree.

Mollakhalili et al. proposed a new version of distributed learning automata and a new adaptive procedure based on extended distributed learning automata (EDLA), to solve optimization problems in stochastic graphs. They presented two algorithms based on the new version of DLA, for solving SSP and SMST problems. The results indicate the superiority of the proposed EDLA-based algorithm compared with previous automata-based algorithms (Meybodi and Meybodi, 2014). Rezvanian et al. (Rezvanian and Meybodi, 2015a) presented several distributed learning automata methods for solving the maximum clique problem in stochastic graphs, for some applications in social network analysis when the probability distribution functions of the weights associated with the edges of the stochastic graph are unknown. They showed theoretically that their algorithm for finding the maximum clique converged when the learning parameters were properly chosen.

Rezvanian et al. developed multiple learning automata-based algorithms for finding the minimum vertex covering in stochastic graphs (Rezvanian and Meybodi, 2015b) where the PDF of the weight associated with the graph vertices is unknown, motivated by the application of information spreading throughout online social networks. They showed that their algorithm with two sets of learning automata significantly outperformed the other algorithm with a single learning automaton.

The summary of studies for LA applications on graph problems is given in Table 1.

3. Adaptive Petri net based on learning automata

In this section, first, a brief review of Petri nets is presented. Then, the adaptive Petri net based on learning automata is briefly reviewed. Finally, the APN-LA family is introduced.

Table 1
Summary of studies for LA applications on graph problems.

Graph problem	LA type
Shortest path problem	LA (Misra and Oommen, 2009), DLA (Beigy and Meybodi, 2006), EDLA (Meybodi and Meybodi, 2014), VDLA (Vahidipour et al., 2017a).
Graph coloring problem	LA (Torkestani and Meybodi, 2012c) (Bouhmala and Granmo, 2010), CLA (Akbari Torkestani and Meybodi, 2011), IOCLA (Mirsaleh and Meybodi, 2016)
Dominating set problem	LA (Torkestani and Meybodi, 2011a,b, 2012a), DLA (Jamehshourani et al., 2011; Khomami et al., 2018; Torkestani and Meybodi, 2011a,b)
Maximum independent set problem	CLA (Motevallian and Meybodi, 2005), DLA (Alipour and Meybodi, 2006) LA (Khomami et al., 2016)
Maximum cut problem	CLA (Enayatzare and Meybodi, 2008)
Graph isomorphism	LA (Beigy and Meybodi, 2002)
Graph partitioning	LA (Oommen and de St Croix, 1996), CLA (Farshbaf et al., 2011)
Minimum spanning tree problem	DLA (Torkestani and Meybodi, 2011a,b, 2012b), LA (Torkestani, 2012), EDLA (Meybodi and Meybodi, 2014)
Maximum clique problem	LA (Soleimani-Pouri et al., 2012), DLA (Rezvanian and Meybodi, 2015a)
Minimum vertex covering problem	LA (Mousavian et al., 2013), DLA (Rezvanian and Meybodi, 2015b), CLA (Mousavian et al., 2014)
Steiner tree problem	LA (Noferesty and Meybodi, 2008), DLA (Nourollah and Meybodi, 2007)

3.1. Petri nets

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

- For each element x, either a place or a transition, its pre-set is defined as $\bullet x = \{y \in P \cup T | W(y, x) > 0\}$ and its post-set is defined as $x\bullet = \{y \in P \cup T | W(x, y) > 0\}$.
- A marking of a PN is a function $M: P \rightarrow \mathbb{N}$ where $M(p_i)$ denotes the number of tokens in p_i. A PN along with an initial marking M_0creates a PN system denoted by(N, M_0).
- A set $\dot{P} \subseteq P$ is marked in a marking M, $if f \exists p \in \dot{P}, M(p) > 0$; otherwise $\dot{P}$ is unmarked or empty in M.
- A transition t is *enabled* in marking$M$, denoted by$M[t>, if f M(p) \geq W(p, t), \forall p \in P$.
- 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 (Murata, 1989; Peterson, 1981). In other words, this set (1) determines the set of enabled transitions from available transitions; (2) selects a transition from the set of enabled transitions for firing; and, (3) generates a new marking by firing the selected transition. The second step, within the above steps, is the one where a decision making needs to be performed for selecting an enabled transition for firing. This is where controlling mechanism is used.

A controlling mechanism must be able to select an enabled transition for firing in any of the following two situations: (1) All enabled transitions are concurrent and they are causally independent, which means one transition may fire before, after or in parallel with the others (Murata, 1989) and (2) some of the enabled transitions are in conflict, which means if any of them is fired, none of the others can be fired (Murata, 1989). In this paper, the mechanisms of selecting an enabled transition among a set of concurrent transitions are referred to as *concurrent selectors* and the other mechanisms are referred to as *conflict resolvers*, hereafter.

3.2. Adaptive Petri net based on learning automata

An adaptive Petri net based on learning automata (APN-LA) is a Petri net obtained from the fusion between Petri nets and learning automata (Vahidipour et al., 2015). In this Petri net, an LA is used as the conflict resolver in the controlling mechanism and a random mechanism is used as the concurrent selector. The LA is assigned to a cluster of transitions to resolve conflicts among them. To form such clusters, we consider transitions with the same input places as members of a cluster. In this way, the conflicting transitions form one or more clusters. The remaining transitions, which are not in conflict with any other transitions, form another cluster, in which the concurrent selector, or in other words the random mechanism, is used to select an enabled transition for firing.

Like other Petri nets, APN-LAs can be designed to represent an algorithm for solving a problem. To this end, the construction procedure can be 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 $s_0, s_1, \ldots, s_n$. 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 and also two transitions in two different clusters do not have any input place in common. Note also 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 for providing a way of achieving fusion between Petri nets and learning automata. Inserted transitions are immediate transitions, except that when they fire, the internal structure of the LA 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 given in (Vahidipour et al., 2015).

The firing rules of an APN-LA system differ from those of an ordinary PN due to the fusion with learning automata (Vahidipour et al., 2015). The set of firing rules of an APN-LA, in a given marking, can be briefly described as follows.

1- Determine the set of enabled transitions.
2- Specify a firing cluster among the set of enabled clusters. A cluster of transitions is enabled in a given marking when at least one of its transitions is enabled. This selection is performed randomly.
3- If the firing cluster is the cluster s_0, then the concurrent selector is used to select a transition in s_0 for firing.

4- Otherwise, the conflict resolver of the firing cluster is used to select a transition in that cluster for firing.
5- The selected transition is fired, and a new marking is generated.
6- If the fired transition is an updating transition, then the internal state of the related conflict resolver is updated.

The interested reader may refer to (Vahidipour et al., 2015) for more details on the firing rules of the APN-LA.

3.3. The APN-LA family

A number of models based on APN-LA have been proposed in the literature so far. In this paper, we refer to all these models as the APN-LA family. The first member of this family is ASPN-LA (Vahidipour et al., 2015). ASPN-LA is an APN-LA where the underlying Petri net is a stochastic Petri net (SPN). This PN is used for representing and analyzing a number of priority-assignment algorithms in queuing systems (Vahidipour et al., 2015). The ASPN-LA is also used for analyzing algorithms in the shortest path problem in stochastic graphs (Vahidipour et al., 2017a).

Another member of the family is APN-ICLA (Vahidipour et al., 2017b). The idea behind constructing APN-ICLA is that in some situations, having a means of cooperation between learning automata within a single Petri net may be beneficial. This cooperation is not available in APN-LA, as in APN-LA each learning automaton acts independently from the others. APN-ICLA can be regarded as a Petri net, in which the controlling mechanism is cooperative. APN-ICLA has been successfully used for solving the vertex coloring problem.

The last member of the family is CAPN-LA (Vahidipour et al., 2017c), which can be used for representing cellular algorithms. In this model, a number of identical APN-LAs are organized into a cellular structure and each APN-LA represents an algorithm which must be executed in each cell. APN-LAs in neighboring cells cooperate with each other in order to solve the whole problem. The idea behind CAPN-LA is to mitigate the fact that designing a single APN-LA for representing cellular algorithms is tedious work and results in a large and complex model. This member of the APN-LA family has been used for representing a number of cellular algorithms for solving the vertex coloring graph problem (Vahidipour et al., 2017c).

4. The proposed generalized adaptive PN based on learning automata

In this section, a generalized adaptive Petri net based on learning automata, called GAPN-LA, is proposed. A GAPN-LA is a large Petri net which is mapped into a graph structure. In this mapping, places and transitions of the Petri net are mapped into vertices of the graph: each place or transition is mapped into exactly one of the vertices and each vertex forms a part of the places and transitions of the Petri net. This graph provides a means of visualizing neighborhood relationships between different parts of the Petri net. Each part of the Petri net can be an APN-LA, and conflicts among the conflicting transitions are resolved using an LA.

From a modular perspective, each vertex in the GAPN-LA can be considered as a module which performs a specific task. Thus, a problem can be solved as a result of different modules performing different tasks of the GAPN-LA. From this perspective, four problem-solving paradigms can be considered for GAPN-LA.

1. Solitary: in this paradigm, modules perform solely independently of each other.
2. Local cooperation (LC): in the local cooperation paradigm, neighboring modules (vertices) cooperate with each other for solving the problem.
3. Global cooperation (GC): problem solving in this paradigm occurs by means of cooperation between all modules of the GAPN-LA.
4. Composite: any combination of the above paradigms can be used for problem solving in the composite paradigm. For example, some of the modules may cooperate locally while the rest of the modules cooperate globally.

Using the above-mentioned problem-solving paradigms, GAPN-LA becomes a powerful framework which can be used for solving any graph-based problem.

The sets of places and transitions mapped into a vertex v_i, are denoted by P_i and T_i respectively. As in other Petri nets, the controlling mechanism in vertex v_i has two parts: (1) a concurrent selector and (2) a conflict resolver. The concurrent selector in v_i is a random mechanism, whereas its conflict resolver is a learning automaton (LA) with a varying number of actions. This LA is denoted LA_i and is assigned to a cluster of transitions, i.e., $s_{i,1}$, which may conflict with each other in some markings. For these markings, LA_i selects an enabled transition from among the enabled transitions in $s_{i,1}$ for firing.

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

Definition 1. A generalized adaptive Petri net based on learning automata (GAPN-LA) is a structure $\mathcal{N} = (G\langle V, \mathbb{E}\rangle, \widehat{P}, \widehat{T}, \widehat{W}, \{S_1, \ldots, S_n\}, \{LA_1, \ldots, LA_n\}, \widehat{F})$, where:

I. G is an undirected graph, with $V = \{v_0, v_1, \ldots, v_n\}$ as the set of vertices (cells) and $\mathbb{E}$ is the set of edges (adjacency relations). Furthermore, v_0 has no edges.
II. $\widehat{\mathrm{P}} = \bigcup_{i=0,1,\ldots,n} P_i$ is a finite set of places. $P_i, (i = 0, \ldots, n)$ is the set of places assigned to the vertex v_i.
III. $\widehat{T} = T^u \cup \{\bigcup_{i=0,1,\ldots,n} T_i\}$ is a finite set of transitions, where:
   (a) $T^u = \bigcup_{j=1}^{k} t_j^u, 1 \le k \le n$ is a set of updating transitions. An updating transition t_j^u is an immediate transition, except that when it fires, the internal structures of one or more learning automata are updated. The set of learning automata related to t_j^u is denoted by $H(t_j^u)$.
   (b) $T_i, (i = 0, \ldots, n)$ is the set of transitions assigned to vertex v_i.
IV $\widehat{\mathrm{W}}: \left(\left(\widehat{\mathrm{P}} \times \widehat{\mathrm{T}}\right) \cup \left(\widehat{\mathrm{T}} \times \widehat{\mathrm{P}}\right)\right) \to \mathbb{N}$ defines the interconnection of $\widehat{\mathrm{T}}$ and $\widehat{\mathrm{P}}$.
V. $S_i = \{\mathrm{s_{i,0}}, \mathrm{s_{i,1}}\}$ $(i = 1, \ldots, n)$ is the set of clusters of transitions in vertex v_i where:
   (a) $s_{i,0}$ contains concurrent transitions from T_i. No pair of transitions in this cluster have any input places in common, i.e., $\{\forall t_k, t_j \in s_{i,0} | \bullet \mathrm{t_k} \cap \bullet \mathrm{t_j} = \varnothing\}$. A random mechanism is used as the concurrent selector in this cluster.
   (b) $s_{i,1}$ contains the remaining transitions in T_i which are not in $s_{i,0}$. The set of transitions in the cluster $s_{i,1}$ may conflict with each other in some markings, and thus a conflict resolver is required to select an enabled transition for firing in this cluster.
VI. $LA_i (i = 1, \ldots, n)$ is a learning automaton with a varying number of actions, which is assigned to cluster $s_{i,0}$ in cell v_i. LA_i is the conflict resolver of the cluster $s_{i,1}$.
VII. $\widehat{F} = \bigcup_{j=1}^{k} F_j, 1 \le k \le n$ is a set of signal generator functions. Function F_j is executed when the updating transition t_j^u is fired, and then generates signal β_j. Upon generation of β_j, learning automata in the set $H(t_j^u)$ update their action probability vectors using their learning algorithms.

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

i. $\mathcal{N}$ is a GAPN-LA structure,

ii. $M_0 = [\mu_{0,0}, \mu_{1,0}, \ldots, \mu_{n,0}]$ is the initial marking vector of $\mathcal{N}$, such that $\mu_{i,0}, (i = 1, \ldots, n)$ is the initial marking of cell υ_i and $\mu_{0,0}$ are the initial markings of the other places which are not in any $\upsilon_i, (i = 1, \ldots, n)$. M is used to represent the set of markings of $\mathcal{N}$ and $\mu_i, (i = 1, \ldots, n)$ represents the set of markings of υ_i. The term $\mu_{i,j}$ represents the jth marking in υ_i.

A GAPN-LA system evolves from the current marking vector to a new marking vector according to the firing rules constituting the Petri net. The firing rules of the GAPN-LA, in any marking M, can be described as follows.

1. A transition $t \in \widehat{T}$ is said to be enabled if each input place p of t is marked with at least $\widehat{W}(p,t)$ tokens.
2. Only one enabled transition $t \in \widehat{T}$ can fire in each marking M.
3. Before selecting an enabled transition t for firing, a set of transitions must be selected, referred to as the fired set. T^u is selected as the fired set with probability $\frac{|t_j^u \in T^u, M[t_j^u>|}{|t \in T, M[t>|}$ or $T_i, (i = 0, 1, \ldots, n)$ is selected as the fired set with probability $\frac{|t' \in T_i, M[t'>|}{|t \in T, M[t>|}, i = 0, 1, \ldots, n$, where $|\ |$ stands for the norm operator.
4. If T^u or T_0 is the fired set, then an enabled transition t from the fired set is randomly selected for firing. Otherwise an enabled transition t is selected from the fired set T_i according to the following steps:

   4.1 t is selected from the cluster $s_{i,0}$(or $s_{i,1}$) 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 $P_{i,0} + P_{i,1} = 1$.

   4.2 If $s_{i,0}$ is the fired cluster, then an enabled transition t is randomly selected from $s_{i,0}$ for firing. Otherwise, LA_i is activated. The available action set of the activated LA_i consists of the actions corresponding to the enabled transitions. The activated LA selects an action from its available action set according to its action probability vector and then the corresponding transition is selected for firing.

5. Firing of a transition $t \in \hat{\mathrm{T}}$ removes $\widehat{W}(p,t)$ tokens from each input place p of t, and adds $\widehat{W}(t,p)$ tokens to each output place p of t.
6. The function $F_j \in \hat{F}$ is used to generate the reinforcement signal β_j for $H(t_j^u)$ when the updating transition t_j^u fires. Using β_j, learning automata in $H(t_j^u)$ update their action probability vectors using their learning algorithms.

Remark 1. Considering the different problem-solving paradigms for GAPN-LA, we assume the following three different types of function F_j.

(a) If function F_j uses the whole marking of the GAPN-LA to generate the reinforcement signal, the reinforcement signal is globally generated and F_j is a global function. If all functions in $\hat{F}$ are global functions, then the GAPN-LA operates in the GC paradigm.
(b) If function F_j uses the marking of a vertex to generate the reinforcement signal for the LA in that vertex, the reinforcement signal is generated in a solitary manner and F_j is a solitary function. If all functions in $\hat{F}$ are solitary functions, then the GAPN-LA operates in the solitary paradigm.
(c) If function F_j uses markings of a vertex and its neighboring vertices to generate the reinforcement signal for the LA in that cell, the reinforcement signal is locally generated and F_j is a local function. If all functions in $\hat{F}$ are local functions, then the GAPN-LA operates in the LC paradigm.

Remark 2. GAPN-LA can be considered as a generalization of three adaptive Petri nets APN-LA, APN-ICLA and CAPN-LA. This is clarified by the following points.

i. If $s_{i,0} = \{\varnothing\}, (i = 1, \ldots, n)$ in the GAPN-LA and the GAPN-LA operates in the solitary paradigm, then GAPN-LA becomes an APN-LA.
ii. If the GAPN-LA operates in the LC paradigm such that its learning automata form an ICLA interconnected structure, then the GAPN-LA becomes an APN-ICLA.
iii. If $P_o = \{\varnothing\}$ and $T_o = \{\varnothing\}$ in the GAPN-LA and the GAPN-LA operates in the LC paradigm, then the GAPN-LA becomes a CAPN-LA.

Remark 3. The next section illustrates that how well the proposed framework can be utilized in graph-based problems. However, following marks need to be considered before any decision for using GAPN-LA:

(a) Utilizing GAPN-LA for solving graph-based problems efficiently requires a good familiarity with both cellular algorithms and Petri nets.
(b) Before using this framework, the design process of GAPN-LA is needed.
(c) If the underlying graph of a problem would be very large, then a large Petri net needs to be analyzed and executed which takes huge amount of running time.

5. GAPN-LA: A framework for solving graph problems

In this section, the proposed GAPN-LA is used to solve several graph problems. These are the vertex coloring (VC) problem, the minimum vertex covering (MVC) problem and the minimum dominating set (MDS) problem. We first describe each problem in a graph $\mathbb{G}$. Next, a GAPN-LA is constructed for solving the problem, referred to by a compound name, i.e., GAPN-LA-[the abbreviation of the problem]. For example, we solve the VC problem, the problem of coloring graph vertices using m colors such that neighboring vertices have different colors, using the GAPN-LA-VC framework. The GAPN-LA-VC framework represents and simulates different LA-based algorithms for solving the VC problem in a graph $\mathbb{G}$.

The input data used for the experiments are a subset of DIMACS benchmarks (Johnson and Trick, 1996). All simulations were implemented in MATLAB 8.1 and run on a Pentium V Core 2 Duo 2.0 GHz computer. Simulation parameters are set as follows: $a = b = 0.1$, i.e., the reward and penalty parameters, thresholds ϵ and τ are equal to 0.009 and 0.95 respectively, as described in more detail later in this section.

5.1. Vertex Coloring (VC) problem

In this section, a GAPN-LA framework is introduced for solving the VC problem in a graph. The vertex coloring problem is a well-known coloring problem (Jensen and Toft, 2011) in which a color is assigned to each vertex of the graph. A legal vertex coloring of graph $\mathbb{G}\langle V, \mathbb{E}\rangle$, where $V(\mathbb{G})$ is a set of $|V| = n$ vertices and $\mathbb{E}(\mathbb{G})$ is an edge set, is when distinct colors are assigned 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 $\mathbb{G}$, $\mathbb{C} = \{\mathfrak{c}_1, \ldots, \mathfrak{c}_m\}$ denotes the set of colors assigned to the vertices and $H : V \to \mathbb{C}$ is the coloring function that assigns a color to each vertex such that $F(u) \neq F(v)$ for every $(u, v) \in \mathbb{E}$. Graph $\mathbb{G}\langle V, \mathbb{E}\rangle$ is P-colorable if it can be legally colored with at most P different colors. The VC problem can be considered as a decision-making problem that aims at deciding, for a given graph, whether or not the graph is P-colorable. This is called the P-coloring problem. The chromatic number $\mathfrak{X}(G)$ is the minimum number of colors required for coloring the graph, and a graph $\mathbb{G}$ is said to be P-chromatic if $\mathfrak{X}(\mathbb{G}) = P$. 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., $\mathfrak{X}(\mathbb{G}) \leq \Delta + 1$.

5.1.1. Solving the VC problem with a GAPN-LA

In this section, a GAPN-LA is designed which uses a local cooperation paradigm to solve the VC problem. A vertex uses the neighboring information to select an allowable color from a set of colors. To describe this GAPN-LA, first, an adaptive Petri net is introduced to represent a local module for each vertex of graph $\mathbb{G}$. This Petri net is referred to as the APN-LA-VC. Then, to construct the GAPN-LA-VC, the APN-LA-VC is replicated into all vertices of the graph $\mathbb{G}$.

The GAPN-LA-VC

Each vertex $v_i, i = 1, \ldots, n$ of the graph $\mathbb{G}$ is modeled by an APN-LA, as shown in Fig. 1. In this PN, $P_{i,0}$ is the place of making a decision on 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. 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, and therefore is responsible for determining the color of the vertex v_i. This is achieved as follows. The action set of LA_i, referred to as $\underline{\alpha}_i$, contains m actions. When LA_i is activated, it selects one of its actions, say $\alpha_{i,r}$, according to its action probability vector. As a result, the transition $t_{i,\mathrm{r}}$ corresponding to the selected action is fired and a token appears in the place $P_{i,r}$. Hence, a color is assigned to the vertex.

The GAPN-LA-VC is achieved when the APN-LA shown in Fig. 1 is assigned to each vertex of the graph. The GAPN-LA-VC uses the local cooperation paradigm, and the GAPN-LA-VC system is as defined as described below.

I. G is an undirected graph which is to be colored, with $V = \{v_0, v_1, \ldots, v_n\}$ as the set of vertices (cells) and $\mathbb{E}$ as the set of edges (adjacency relations). Note that v_0 has no edges.

II. $\hat{\mathrm{P}} = \bigcup_{i=0,1,\ldots,n} P_i$ is a finite set of places and $P_i = \{P_{i,0}, \ldots, P_{i,m+2}\}$, $(i = 0, \ldots, n)$ is the set of places assigned to the vertex v_i.

III. $\hat{T} = T^u \cup \{\bigcup_{i=0,1,\ldots,n} T_i\}$ is a finite set of transitions, where:

- $T^u = \bigcup_{j=1}^{n} t^u_{j,1}$ is a set of updating transitions, and updating transition $t^u_{j,1}$ is assigned to the vertex v_j.
- $T_i = \{t_{i,0}, \ldots, t_{i,2m}\}, (i = 0, \ldots, n)$ is the set of transitions assigned to the vertex v_i.

VI. $\hat{\mathrm{W}}: \left(\left(\hat{\mathrm{P}} \times \hat{\mathrm{T}}\right) \cup \left(\hat{\mathrm{T}} \times \hat{\mathrm{P}}\right)\right) \rightarrow \mathbb{N}$ defines the interconnection of $\hat{\mathrm{T}}$ and $\hat{\mathrm{P}}$. Places and transitions assigned to the vertex v_i are interconnected according to Fig. 1.

V. $S_i = \{\mathrm{s_{i,0}}, \mathrm{s_{i,1}}\}$ $(i = 1, \ldots, n)$ is the set of clusters of transitions in vertex v_i, where:

- $s_{i,0}$ contains concurrent transitions from T_i. No pair of transitions in this cluster has any input places in common, i.e., $\{\forall t_k, t_j \in s_{i,0} | \bullet \mathrm{t_k} \cap \bullet \mathrm{t_j} = \varnothing\}$. A random mechanism is used as the concurrent selector in this cluster.
- The cluster $s_{i,1}$ contains the remaining transitions in T_i which are not in $s_{i,0}$. The transitions in the cluster $s_{i,1}$ may conflict with each other in some markings, and thus a conflict resolver is required to select an enabled transition for firing in this cluster.

VI. $LA_i (i = 1, \ldots, n)$ is a learning automaton with a varying number of actions, which is assigned to cluster $s_{i,0}$ in cell v_i. LA_i is the conflict resolver of the cluster $s_{i,1}$.

VII. $\hat{\mathrm{F}} = \{\mathrm{F_1}, \ldots, \mathrm{F_n}\}$ is the set of local functions. F_i, the local function related to LA_i, is executed upon the firing of $t^u_{i,1}$ in the A_i and generates the reinforcement signal β_i for LA_i using the set of previous firing transitions of all neighboring clusters of $s_{i,1}$. A simple local function can be described as follows.

- If the selected transition of $\mathrm{s_{i,1}}$ is different from the previously selected transitions of all its neighbors, then it is rewarded.
- Otherwise, it is penalized.

It should be noted that the local function stated above is just an example. In real scenarios, any other function can be used instead. Based on the proposed GAPN-LA-VC, different algorithms can be designed by:

1. substituting different rules in the definition of a local function, or
2. defining different action sets for learning automata, or
3. using different mechanisms for selecting an action from the available action set of the learning automata.

A solution of the VC problem is obtained from the configuration of the conflict resolvers in the GAPN-LA-VC (refer to Definition 1). To this end, the *max* operator is utilized over the action probability vector $\underline{q}_i$ to determine the color of vertex v_i. The color related to the action with the highest value in $\underline{q}_i$ is assigned to v_i.

5.1.2. Simulation results

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

- CN: Number of colors required for coloring the graph,
- $\mathfrak{U}$: Total number of updates of the learning automata up to the time instant $\mathbb{k}$, where $\mathbb{k}$ is defined to be the smallest time instant at which the colors of all vertices in the graph are fixed,

To study the efficiency of the algorithms, a number of simulation studies have been conducted on a subset of hard-to-color benchmarks reported in DIMACS (Johnson and Trick, 1996). In the following, APN-ICLA-VC is used to implement and compare five different VC algorithms. The first four algorithms were introduced in Akbari Torkestani (2013); in each of these an ICLA is used to solve the VC problem (referred to as VC1 to VC4). The fifth algorithm was proposed in Vahidipour et al. (2017c), and is referred to as VC5.

VC1: In the first algorithm, which we call VC1, it is assumed that the number of available colors for coloring each vertex v_i or equivalently the number of actions in the action set of LA_i is equal to the number of vertices in the graph $\mathbb{G}$; that is $m = \left|\underline{\alpha}_i\right| = |V|$. In the VC1, when LA_i is activated, an action from its action set is selected. All learning automata in the GAPN-LA-VC update their action probability vectors using an L_{R-I} learning algorithm. The local function of the GAPN-LA-VC in VC1 algorithm is actually just simple function, defined as follows:

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

VC2: In the first algorithm, the number of available actions for each LA is high. This significantly prolongs the time required by each LA to find a suitable action which in turn, prolongs the total running time of the algorithm. On the other hand, as mentioned before, it is shown that an arbitrary graph can be colored with at most $\Delta + 1$ colors, where Δ denotes the maximum degree of the vertices in the graph. To obtain a faster algorithm, in VC2, the number of available actions for each LA is equal to $\Delta + 1$. Due to the reduction in number of actions of each LA, it is expected that the VC2 colors the graph in less time and with a smaller number of colors compared to the VC1.

VC3: In VC2, all LAs have the same number of actions. But it is obvious that a vertex v_i and its neighbors can be colored using at most $\Delta_i + 1$ different colors, where Δ_i is the degree of vertex v_i. Therefore, the number of actions of each vertex v_i can be reduced to $\Delta_i + 1$. This algorithm is referred to as VC3.

VC4: Since a suitable mechanism for solving VC problem needs to decrease the number of required colors for coloring the graph, an enhanced algorithm reduces the number of available actions in each LA, where possible. In this enhanced algorithm, called VC4, learning

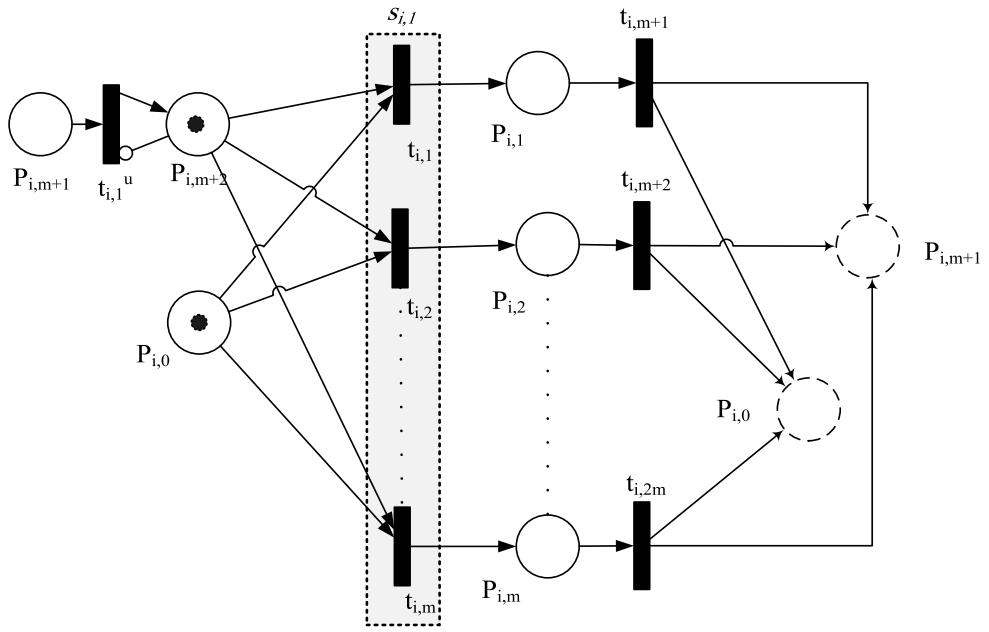

Fig. 1. An APN-LA for vertex v_i. For simplicity, $P_{i,0}$ and $P_{i,m+1}$ have been duplicated (indicated by dashed-line).

automata with varying number of available action set are used, which is 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 above algorithms, i.e. VC1 to VC4, utilize the simple local function defined for VC1. In Vahidipour et al. (2017c), using measure $\omega_i(k)$ defined for vertex v_i at iteration number k, a new local function has been proposed. $\omega_i(k)$ is the ratio between the probability of selection an allowable color in vertex v_i at iteration k and the number of allowable colors in v_i at time instant k. Using $\omega_i(k)$, the following function is defined in Vahidipour et al. (2017c):

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

In other words, the selected action of LA_i is rewarded if (1) it is different from the selected actions of its neighbors and (2) this selection results in better ω_i than the average of ω_i resulted from previous selections of this LA. Here, the VC4 algorithm with this proposed function is referred to as VC5.

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

One problem with algorithms VC1 to VC4 is that they cannot be guaranteed to find a legal coloring of the graph. To eliminate the problem of improper coloring, the following modification is applied to all the mentioned algorithms. All learning automata are considered to be learning automata with a varying number of available action sets. The available actions for an LA at any time instant k are those which are not selected by any neighboring learning automata at that time instant. This property allows each LA to pick only legal colors.

In this simulation study, the time instant k is increased whenever the LA corresponding to the vertex with the maximum degree is updated. Simulation parameters are set as follows: $a = b = 0.1$, $\varepsilon = 0.009$ and $\tau = 0.95$. The results of this simulation study, which are reported in Table 2, indicate the following.

- By 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. This is because decreasing the number of available actions for each learning automaton increases its convergence rate as well as its accuracy.
- VC5 is always better than, or at least equal to, VC4, in terms of all the mentioned criteria. This efficiency is the result of considering the proposed measure ω_i in the signal generator function of the GAPN-LA-VC system.

5.2. Minimum vertex covering (MVC) and minimum dominating set (MDS) problems

In this section, first, a GAPN-LA framework is introduced for finding the minimum vertex covering (MVC) in a graph, and then this framework is used with minor modifications for finding the minimum dominating set (MDS) in a graph. Before discussing the framework, a brief description of these problems is given.

The MVC problem is a well-known graph problem where a set of vertices is chosen from the graph. Let graph $\mathbb{G}\langle V, \mathbb{E}\rangle$ be the input graph, where $V(\mathbb{G}) = \{v_1, v_2, \ldots, v_n\}$ is the set of $|V| = n$ vertices and $\mathbb{E}(\mathbb{G})$ is the edge set. A vertex cover C is a subset of vertices such that, for every $e(v_i, v_j) \in \mathbb{E}$, at least one of the vertices v_i or v_j is an element of C of the graph $\mathbb{G}$. The MVC of graph G is the vertex cover with minimum size among all possible vertex covers of graph $\mathbb{G}$ (Rezvanian and Meybodi, 2015b). The MVC problem is known to be NP-hard (Karp, 1972).

The dominating set problem is a general class of well-known graph optimization problems. A dominating set in a given graph $\mathbb{G}$ is a subset $S \subseteq V(\mathbb{G})$, such that every vertex $v_i \subseteq V(\mathbb{G})$ is either in the dominating set S or adjacent to a vertex of S. A vertex of S is said to dominate itself and all its adjacent vertices. The dominating set with the minimum cardinality is said to be the minimum dominating set (MDS). The MDS problem is known to be NP-hard (Kolaitis and Thakur, 1994).

5.2.1. Solving the MVC problem with a GAPN-LA

In this section, a GAPN-LA is designed which uses a global cooperation paradigm to solve the MVC problem. All vertices cooperate to select a minimum vertex covering set from the set of allowable vertex covering sets. To describe this GAPN-LA, we first describe the Petri net assigned to each vertex $v_i, i = 1, \ldots, n$, and then describe the Petri net assigned to

Table 2
Results of experiments for proposed algorithms represented by CAPN-LA-VC.

Graph	Algorithm									
	VC1		VC2		VC3		VC4		VC5	
	CN	$\mathcal{U}$	CN	$\mathcal{U}$	CN	$\mathcal{U}$	CN	$\mathcal{U}$	CN	$\mathcal{U}$
DSJC125_1	5	1 044 679	5	116 002	5	35 679	5	26 876	5	12 593
DSJC125_5	19	1 067 235	18	124 642	17	46 410	17	39 055	17	27 263
DSJC125_9	48	1 308 101	45	128 119	44	55 224	44	46 407	44	32 004
DSJC250_1	10	641 605	8	84 145	8	28 757	8	24 925	8	14 661
DSJC250_5	31	4 418 776	29	421 963	28	130 756	28	100 635	28	45 645
DSJC250_9	76	6 741 839	75	626 042	73	233 568	72	150 091	72	59 200
DSJC500_1	14	1 110 579	13	144 954	12	86 799	12	67 603	12	50 270
DSJC500_5	52	2 712 399	51	345 971	49	222 161	49	156 358	49	83 597
DSJC500_9	139	6 735 701	131	648 780	128	324 640	127	285 403	127	158 908
le450_15a	15	4 279 467	15	480 839	15	162 445	15	107 575	15	78 335
le450_15b	15	4 629 513	15	497 638	15	183 004	15	138 001	15	59 340
le450_15c	16	4 672 330	15	589 748	15	311 569	15	265 139	15	102 384
le450_15d	17	5 284 660	15	502 709	15	299 948	15	203 358	15	11 965
le450_25c	26	5 516 730	26	621 156	25	313 236	25	255 099	25	103 542
le450_25d	27	5 737 808	26	636 840	25	334 672	25	275 394	25	157 722

vertex v_0 in GAPN-LA. The resulting GAPN-LA is called GAPN-LA-MVC, as shown in Fig. 2.

Each vertex v_i of the graph $\mathbb{G}$ in GAPN-LA-MVC is modeled by the simple PN depicted in Fig. 2. In this PN, $P_{i,0}$ is the place of making the decision for selecting vertex v_i as a member of the covering set and each place $\{P_{i,1}, P_{i,2}\}$ represents a possible decision for this vertex. A token in $P_{i,1}$ represents "selecting v_i to be a member of the vertex cover", and a token in $P_{i,1}$ represents "not selecting v_i to be a member of the vertex cover". Note that the set $\{t_{i,1}, t_{i,2}\}$ is a cluster of conflicting transitions forming the cluster $s_{i,1}$. Therefore, a conflict resolver, LA_i, is responsible for resolving effective conflicts, and therefore is responsible for selecting the vertex v_i in the vertex cover. When a token appears in $P_{i,3}$, then LA_i is activated for selecting an action from its action set. The action set of LA_i is referred to as $\underline{\alpha}_i = \{\alpha_{i,1}, \alpha_{i,2}\}$, where action α_i^1 is "selecting v_i to be a member of the vertex cover", and action α_i^2 is "not selecting v_i to be a member of the vertex cover".

Fig. 3 indicates the Petri net assigned to vertex v_0 in the GAPN-LA-MVC. In this Petri net, when n tokens appear in $P_{0,0}$, the updating transition t^u must be fired (this means that all learning automata in GAPN-LA-MVC decide the covering set of the graph). The term $\#P_{0,1}$ (the number of tokens in place $P_{0,1}$) indicates the size of the vertex covering set and $\#P_{0,2}$ is the number of non-members of the vertex covering set. When transition $t_{0,3}$ is fired, then all learning automata in the GAPN-LA-MVC are activated for the next iteration. Tokens appear in places $P_{i,3}, i = 1, \ldots, n$. Before this, tokens in places $P_{0,1}$ and $P_{o,2}$ disappear, using firing transitions $t_{0,1}$ and $t_{0,2}$ respectively.

GAPN-LA-MVC is a structure that contains a graph and a number of Petri nets, such that the Petri net indicated in Fig. 2 is assigned to each vertex $v_i, i = 1, \ldots, n$ and the Petri net depicted in Fig. 3 is assigned to vertex v_0. GAPN-LA-MVC uses the global cooperation paradigm and the GAPN-LA-MVC system is defined as described below.

I. $\mathbb{G}$ is an undirected graph, with $V = \{v_0, v_1, \ldots, v_n\}$ as the set of vertices and $\mathbb{E}$ as the set of edges.

II. $\hat{\mathrm{P}} = \bigcup_{i=0,1,\ldots,n} P_i$ is a finite set of places. $P_i = \{P_{i,0}, \ldots, P_{i,3}\}, (i = 1, \ldots, n)$ is the set of places assigned to the vertex $v_i, i = 1, \ldots, n$ (Fig. 2) and $P_0 = \{P_{0,0}, \ldots, P_{0,5}\}$ (Fig. 3).

III. $\hat{T} = t^u \cup \{\bigcup_{i=0,1,\ldots,n} T_i\}$ is a finite set of transitions where:

- t^u is the updating transition. Updating transition t^u is considered for updating all learning automata residing in all vertices.

IV. $T_i = \{t_{i,1}, \ldots, t_{i,4}\}, (i = 1, \ldots, n)$ is the set of transitions assigned to the vertex $v_i, i = 1, \ldots, n$ (Fig. 2) and $T_0 = \{t_{0,1}, \ldots, t_{0,3}\}$ (Fig. 3).

V. $\hat{\mathrm{W}}: \left(\left(\hat{\mathrm{P}} \times \hat{\mathrm{T}}\right) \cup \left(\hat{\mathrm{T}} \times \hat{\mathrm{P}}\right)\right) \rightarrow \mathbb{N}$ defines the interconnection of $\hat{\mathrm{T}}$ and $\hat{\mathrm{P}}$. Places and transitions assigned to the vertex v_i are interconnected according to Figs. 2 and 3.

VI. $S_i = \{\mathrm{s_{i,0}}, \mathrm{s_{i,1}}\}$ $(i = 1, \ldots, n)$ is the set of clusters of transitions in vertex v_i where

- $s_{i,0} = \{t_{i,3}, t_{i,4}\}$. A random mechanism is used as the concurrent selector in this cluster.
- Similarly, $s_{i,1} = \{t_{i,1}, t_{i,2}\}$. The set of transitions in the cluster $s_{i,1}$ may conflict with each other in some markings, and thus a conflict resolver is required to select an enabled transition for firing in this cluster.

VII. $LA_i (i = 1, \ldots, n)$ is an LA with two actions which is assigned to cluster $s_{i,1}$ in v_i; LA_i is the conflict resolver of the cluster $s_{i,1}$.

VIII. $\hat{\mathrm{F}} = \{\mathrm{F_1}\}$ is a signal generator function. Function $\mathrm{F_j}$, related to all learning automata, is executed upon the firing of t^u and generates the reinforcement signal β for all learning automata using the set of previous firing transitions of all clusters of $s_{i,1}$ (those learning automata choosing action $\alpha_{i,1}$). A simple signal generator function can be described as follows.

- If a candidate set of vertices creates a vertex covering set and $\#P_{0,1} < \theta$ (the stored value in the function), then learning automata are rewarded and $\theta = \#P_{0,1}$.
- Otherwise, all learning automata are penalized.

Based on the proposed GAPN-LA-MVC, different algorithms can be designed by:

1. substituting different rules in the definition of signal generator functions (using different mechanisms for generating the reinforcement signal β) or
2. defining different strategies and rules for adjustment of the learning rate for the learning algorithm of learning automata.

5.2.2. Solving the MDS problem with a GAPN-LA

The introduced GAPN-LA-MVC can be used for solving the MDS problem if the signal generator function is defined as follows.

- If a candidate set of vertices creates a dominating set and $\#P_{0,1} < \theta$ (the stored value in the function), then learning automata are rewarded and $\theta = \#P_{0,1}$.
- Otherwise, all learning automata are penalized.

The GAPN-LA-MVC framework with this modification is referred to as a GAPN-LA-MDS framework. In the next subsection, we report the simulation results for these two frameworks.

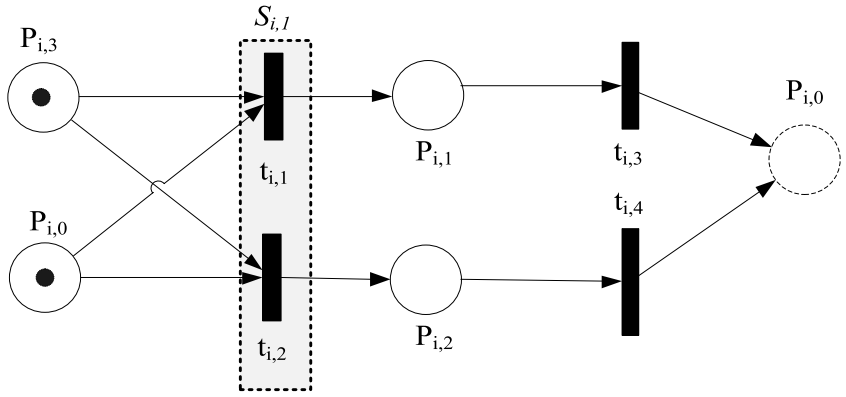

Fig. 2. The Petri net assigned to vertex $v_i, i = 1, \ldots, n$ of the GAPN-LA-MVC.

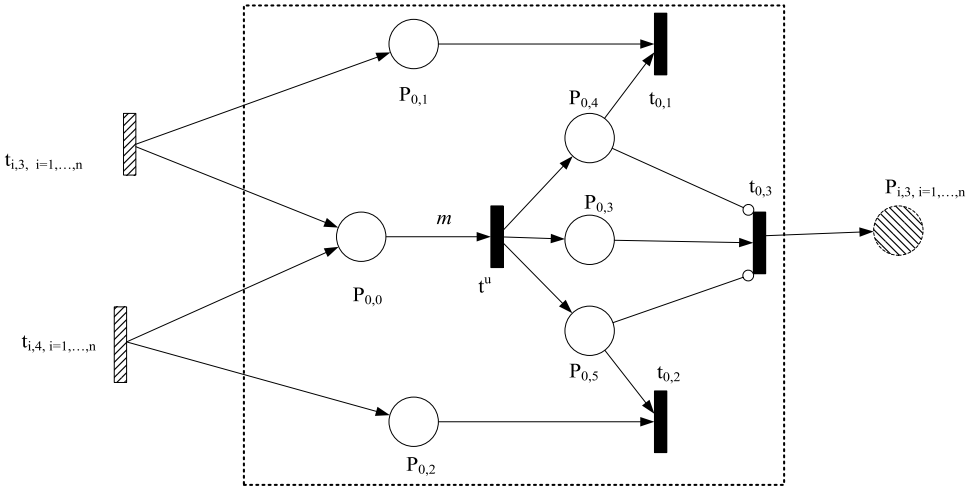

Fig. 3. The Petri net assigned to the vertex v_0 of the GAPN-LA-MVC.

5.2.3. Simulation results

In this section, it will be shown that GAPN-LA-MVC can represent different algorithms for solving the MVC problem and that GAPN-LA-MDS can represent different algorithms for solving the MDS problem. These algorithms are compared with each other in terms of the following criteria:

- MS: minimum size of vertex covering set for the graph found by GAPN-LA-MVC,
- $\mathcal{U}$: total number of updates of the learning automata up to the time instant $\mathbb{k}$, where $\mathbb{k}$ is defined to be the smallest time instant at which the sizes of MVC (MDS) in the graph found by GAPN-LA-MVC (GAPN-LA-MDS) are fixed.

To study the performance of the algorithms, a number of simulation studies were conducted on a subset of DIMACS benchmarks (Johnson and Trick, 1996). In the following, GAPN-LA-MVC is used at first to implement and compare five different MVC algorithms. Then, GAPN-LA-MDS is used to implement these five algorithms for solving the MDS problem.

MVC1: In the first algorithm, which we call MVC1, it is assumed that the learning rate of the learning algorithm for updating the action probability vectors of all learning automata remains fixed. All learning automata in the GAPN-LA-MVC update their action probability vectors using an L_{R-I} learning algorithm. The following simple signal generator function is utilized in MVC1.

- If candidate set of vertices creates a vertex covering set and $\#P_{0,1} < \theta$ (the stored value in the function), then learning automata are rewarded and $\theta = \#P_{0,1}$.
- Otherwise, all learning automata are penalized.

MVC2: In MVC1, all learning automata use the same learning rate. This means that the algorithm treats all the vertices equally during the solving of the graph. This significantly prolongs the time required by each LA to find an optimal action, which in turn prolongs the total running time of the algorithm. It seems that one way to speed up the learning process is to use a higher learning rate for learning automata in those vertices which have a more promising role in finding the MVC. To obtain a faster algorithm in MVC2, we use $a_i = c.p_{i,j}$ as the learning rate of learning automaton LA_i, where c is a constant value in the range $[0, 1]$ and $p_{i,j}$ is the probability of action $\alpha_{i,j}$.

MVC3: Since the performance of any LA-based algorithm is very sensitive to the learning rate of the LA, a large value for the learning rate results in an increased speed of convergence and a decrease in the accuracy of the algorithm, while a small value for the learning rate results in increased accuracy and a decrease in the speed of convergence of the algorithm. To compromise between the speed of convergence and the accuracy of the algorithm, we use $a_i = c.d_{i,j}$ as the learning rate of LA_i, where c is a constant value in the range $[0, 1]$ and $d_{i,j}$ is an estimation of the reward probability for action $\alpha_{i,j}$. The value of $d_{i,j}$ is computed by dividing the number of times that selected action $\alpha_{i,j}$ is rewarded by the number of times that LA_i is activated. In MVC3, the vertices which are more important for MVC are updated with the adaptive value of the learning rate.

MVC4: All the above algorithms, i.e., MVC1 to MVC3, utilize the simple signal generator function defined for MVC1. To obtain a better algorithm in MVC3, one may use a dynamic threshold (DT) measure, as introduced by Rezvanian and Meybodi (2015b) to define a better

Table 3
Results of experiments for the proposed algorithms represented by GAPN-LA-MVC.

Graph	Algorithm							
	MVC1		MVC2		MVC3		MVC4	
	MS	$\mathcal{U}$	MS	$\mathcal{U}$	MS	$\mathcal{U}$	MS	$\mathcal{U}$
DSJC125_1	60	482 811	59	470 061	59	465 430	58	458 967
DSJC125_5	20	294 607	19	275 638	19	272 128	19	248 765
DSJC125_9	13	327 826	13	315 563	12	314 855	12	300 923
DSJC250_1	77	871 231	75	835 143	76	817 809	75	797 736
DSJC250_5	21	623 000	21	618 761	21	614 113	21	606 891
DSJC250_9	13	570 404	13	567 240	13	563 356	13	551 121
DSJC500_1	76	916 142	76	915 509	76	911 147	75	905 443
DSJC500_5	25	847 547	24	828 990	24	826 117	23	814 813
DSJC500_9	13	678 598	13	663 338	13	666 194	13	659 334
le450_15a	106	968 056	106	954 849	106	952 401	105	937 046
le450_15b	97	731 571	97	733 666	96	727 136	95	712 452
le450_15c	55	555 200	55	546 518	55	545 864	54	518 715
le450_15d	61	653 304	61	655 481	60	640 991	59	624 001
le450_25c	64	530 149	63	519 439	63	515 987	63	510 868
le450_25d	62	667 323	61	653 414	61	645 404	60	629 578

Table 4
Results of experiments for the proposed algorithms represented by GAPN-LA-MDS.

Graph	Algorithm							
	MDS1		MDS2		MDS3		MDS4	
	MS	$\mathcal{U}$	MS	$\mathcal{U}$	MS	$\mathcal{U}$	MS	$\mathcal{U}$
DSJC125_1	37	297 354	34	270 877	39	269 513	32	256 789
DSJC125_5	11	162 034	11	159 580	10	143 225	9	117 836
DSJC125_9	5	126 087	4	114 231	4	114 078	4	109 030
DSJC250_1	44	497 846	42	467 680	42	451 947	40	447 852
DSJC250_5	14	377 576	10	327 387	11	328 243	9	325 120
DSJC250_9	6	292 515	6	261 803	5	270 844	4	249 376
DSJC500_1	40	765 365	39	757 743	39	754 133	38	752 062
DSJC500_5	10	565 031	9	555 127	9	556 183	9	559 368
DSJC500_9	6	313 199	6	306 156	5	305 034	5	301 893
le450_15a	61	795 841	60	772 115	58	772 037	58	761 186
le450_15b	51	549 486	50	540 255	50	541 024	50	535 678
le450_15c	29	292 742	29	288 164	29	287 819	29	278 569
le450_15d	33	353 427	32	343 859	32	341 862	32	338 441
le450_25c	33	273 358	33	272 087	33	270 279	33	268 942
le450_25d	32	344 425	30	331 290	31	327 992	30	314 789

signal generator function. The dynamic threshold DT $= \frac{1}{k}\sum_{j=1}^{k}$ MVC(j) is defined as the average of the sizes of all vertex covers found up to iteration number k. Using DT, the simple signal generator function can be described as follows.

- If a candidate set of vertices creates a vertex covering set and $\#P_{0,1} < DT$ (the stored value in the function), then learning automata are rewarded and $= \frac{(k-1)DT+\#P_{0,1}}{k}$.
- Otherwise, all learning automata are penalized.

In other words, LA_i is rewarded if the resulting MVC is better than the average of all MVCs so far. Here, the MVC3 algorithm with this proposed improvement is referred to as MVC4.

The termination condition of all the aforementioned algorithms in each vertex $v_i, i = 1, \ldots, n$ is fulfilled when the action probability of one of the actions of LA_i, say action $\alpha_{i,j}$, reaches a predetermined threshold τ. From this time on, the selection of vertex v_i as a member of MVC will be fixed.

Remark. To solve the MDS problem, the aforementioned algorithms can be applied within the GAPN-LA-MDS framework. The proposed algorithms for solving the MDS problem in GAPN-LA-MDS are called MDS1, MDS2, MDS3 and MDS4. According to this modification, the criteria MS and $\mathcal{U}$ are redefined for GAPN-LA-MDS:

– MS: minimum size of dominating set for the graph found by the GAPN-LA-MDS,

$\mathcal{U}$: total number of updates on the learning automata up to the time instant $\mathbb{k}$, where $\mathbb{k}$ is defined to be the smallest time instant at which the sizes of MDS in the graph found by GAPN-LA-MDS are fixed.

Simulation parameters are set as follows: $a = 0.01$, $b = 0.001$ and $\tau = .95$. The results of the simulation for solving MVC and MDS are reported in Tables 3 and 4 respectively. From Table 3, one can observe that by moving from MVC1 to MVC4, the CN and $\mathcal{U}$ criteria decrease at least for 12 out of the 15 graphs. In other words, MVC4 is the best and MVC1 is the worst algorithm for finding the minimum vertex covering in graphs. The reason for the good performance of MVC4 may be found in the fact that it takes advantage of an adaptive strategy for adjusting the learning rate in the algorithm and the dynamic threshold. From Table 4, similar results for solving MDS using other algorithms can also be obtained, verifying the superiority of an adaptive strategy for adjusting the learning rate and the dynamic threshold.

6. Conclusion

In this paper, GAPN-LA, as a general framework for solving different categories of graph-based problems, was proposed. This framework is a generalization of the APN-LA family and possesses the problem-solving capabilities of all members of the family. It is suitable for representing the categories of both cellular applications and problems with time dependency. For solving any graph-based problem, the proposed GAPN-LA should be mapped into the graph structure of the problem. Places and transitions of the Petri net are mapped into the vertices of the graph. The graph provides a means of visualizing the neighborhood relationships between different parts of the Petri net. From a modular perspective, each vertex in GAPN-LA, can be considered as a module which performs a specific task. In order to elaborate the capabilities of GAPN-LA, we designed and presented several algorithms for solving

the vertex coloring problem, the minimum vertex covering problem and the minimum dominating set problem in graphs. Several computer simulations were conducted to prove the efficiency of the proposed algorithms compared with existing state-of-the-art algorithms.

Acknowledgment

This research was in part supported by a grant from IPM, Iran (No. CS1397-4-79).

References

Akbari Torkestani, J., 2013. A new approach to the vertex coloring problem. Cybern. Syst. 44, 444–466.

Akbari Torkestani, J., Meybodi, M.R., 2011. A cellular learning automata-based algorithm for solving the vertex coloring problem. Expert Syst. Appl. 8, 9237–9247.

Alipour, M., Meybodi, M., 2006. Solving maximal independent set problem using distributed learning automata. In: 14th Iranian Electrical Engineering Conference, ICEE2006. Tehran, Iran, pp. 1–11.

Bause, F., 1996. On the analysis of Petri nets with static priorities. Acta Inform. 33, 669–685.

Bause, F., 1997. Analysis of Petri nets with a dynamic priority method. Appl. Theory Petri Nets 1997, 215–234.

Beigy, H., Meybodi, M.R., 2002. A learning automata based graph isomorphism algorithm. In: JCIS. Presented at the International Joint Conference on Information Science, pp. 344–348.

Beigy, H., Meybodi, M.R., 2004. A mathematical framework for cellular learning automata. Adv. Complex Syst. 3, 295–319.

Beigy, H., Meybodi, M.R., 2006. Utilizing distributed learning automata to solve stochastic shortest path problems. Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 14, 591–615.

Billard, E.A., 1996. Stability of adaptive search in multi-level games under delayed information. IEEE Trans. Syst. Man Cybern.-Part Syst. Hum. 26, 231–240.

Billard, E.A., 1997. Chaotic behavior of learning automata in multi-level games under delayed information. In: IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. IEEE, Orlando, USA, pp. 1412–1417.

Bouhmala, N., Granmo, O.-C., 2010. Stochastic learning for SAT-encoded graph coloring problems. Int. J. Appl. Metaheuristic Comput. 1, 1–19.

Burkhard, H.-D., 1981. Ordered firing in Petri nets. EIK J. Inf. Process. Cybern. 17, 71–86.

Enayatzare, M., Meybodi, M.R., 2008. An algorithm based on cellular learning automata for solving maximum cut problem in graph. In: Second Iranian Data Mining Conference. Tehran, Iran, pp. 1–11.

Esnaashari, M., Meybodi, M.R., 2015. Irregular cellular learning automata. IEEE Trans. Cybern. 45, 1622–1632.

Farshbaf, M., Feizi-Derakhshi, M.-R., Roshanpoor, A., 2011. Multi-objective optimization using hybrid genetic algorithm and cellular learning automata applying to graph partitioning problem. In: 11th International Conference on Hybrid Intelligent Systems, (HIS). IEEE, pp. 722–727.

Hashemi, A., Meybodi, M.R., 2009. Cellular PSO: A PSO for dynamic environments. In: Cai, Z. (Ed.), Advances in Computation and Intelligence. In: Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 422–433.

Holloway, L.E., Krogh, B.H., 1994. Controlled Petri nets: A tutorial survey. In: 11th International Conference on Analysis and Optimization of Systems Discrete Event Systems. Springer, pp. 158–168.

Jamehshourani, P., Meybodi, M.R., Torkestani, J.A., 2011. Solving Steiner connected dominating set using distributed learning automata. In: Proceedings of IEEE International Conference on Intelligent Computing and Intelligent Systems, ICIS 2011, pp. 477–480.

Jensen, T.R., Toft, B., 2011. Graph Coloring Problems. John Wiley & Sons.

Johnson, D.S., Trick, M.A., 1996. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. American Mathematical Society.

Karp, R.M., 1972. Reducibility among combinatorial problems. Complex Comput. Comput. 40, 85–103.

Khomami, M.M.D., Bagherpour, N., Sajedi, H., Meybodi, M.R., 2016. A new distributed learning automata based algorithm for maximum independent set problem. In: Artificial Intelligence and Robotics (IRANOPEN). IEEE, Qazvin, Iran, pp. 12–17.

Khomami, M.M.D., Rezvanian, A., Bagherpour, N., Meybodi, M.R., 2018. Minimum positive influence dominating set and its application in influence maximization: a learning automata approach. Appl. Intell. 48 (3), 570–593. http://dx.doi.org/10.1007/s10489-017-0987-z.

Kolaitis, P.G., Thakur, M.N., 1994. Logical definability of NP optimization problems. Inform. and Comput. 115, 321–353.

Mahdaviani, M., Kordestani, J.K., Rezvanian, A., Meybodi, M.R., 2015. LADE: Learning automata based differential evolution. Int. J. Artif. Intell. Tools 24, 1550023.

Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G., 1994. Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons, Inc..

Meybodi, M.R.M., Meybodi, M.R., 2014. Extended distributed learning automata. Appl. Intell. 41, 923–940.

Mirsaleh, M.R., Meybodi, M.R., 2016. A new memetic algorithm based on cellular learning automata for solving the vertex coloring problem. Memetic Comput. 8, 211–222.

Misra, S., Oommen, B.J., 2009. An efficient pursuit automata approach for estimating stable all-pairs shortest paths in stochastic network environments. Int. J. Commun. Syst. 22, 441–468.

Motevallian, A., Meybodi, M.R., 2005. Solving maximum independent set problem using cellular learning automata. In: 10th Annual CSI Computer Conference of Iran. Tehran, Iran, pp. 1–10.

Mousavian, A., Rezvanian, A., Meybodi, M.R., 2013. Solving minimum vertex cover problem using learning automata. In: 13th Iranian Conference on Fuzzy Systems, IFSC 2013, pp. 1–5.

Mousavian, A., Rezvanian, A., Meybodi, M.R., 2014. Cellular learning automata based algorithm for solving minimum vertex cover problem. In: 2014 22nd Iranian Conference on Electrical Engineering (ICEE). IEEE, pp. 996–1000.

Murata, T., 1989. Petri nets: Properties, analysis and applications. Proc. IEEE 77, 541–580.

Narendra, K.S., Thathachar, M.A., 1989. Learning Automata: An Introduction. Prentice-Hall.

Noferesty, S., Meybodi, M.R., 2008. A learning automata-based algorithm for solving Steiner tree problem. CSI J. Comput. Sci. Eng. 6, 53–59.

Nourollah, A., Meybodi, M.R., 2007. Solving minimum steiner tree problem using generalized distributed learning automata. In: First Iranian Data Mining Conference. Tehran, Iran, pp. 1–9.

Oommen, B.J., de St Croix, E.V., 1996. Graph partitioning using learning automata. IEEE Trans. Comput. 45, 195–208.

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

Rastegar, R., Arasteh, A.R., Hariri, A., Meybodi, M.R., 2004. A fuzzy clustering algorithm using cellular learning automata based evolutionary algorithm. In: Fourth International Conference on Hybrid Intelligent Systems. IEEE, pp. 310–314.

Rezvanian, A., Meybodi, M.R., 2015a. Finding maximum clique in stochastic graphs using distributed learning automata. Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 23, 1–31.

Rezvanian, A., Meybodi, M.R., 2015b. Finding minimum vertex covering in stochastic graphs: A learning automata approach. Cybern. Syst. 46, 698–727.

Soleimani-Pouri, M., Rezvanian, A., Meybodi, M.R., 2012. Solving maximum clique problem in stochastic graphs using learning automata. In: 2012 Fourth International Conference on Computational Aspects of Social Networks, CASoN., pp. 115–119.

Thathachar, M., 1987. Learning automata with changing number of actions. IEEE Trans. Syst. Man Cybern. 17, 1095–1100.

Thathachar, M.A.L., Sastry, P.S., 1987. A hierarchical system of learning automata that can learn die globally optimal path. Inf. Sci. 42, 143–166.

Thathachar, M.A.L., Sastry, P.S., 2004. Networks of Learning Automata: Techniques for Online Stochastic Optimization. Kluwer Academic Publishers.

Torkestani, J.A., 2012. Degree-constrained minimum spanning tree problem in stochastic graph. Cybern. Syst. 43, 1–21.

Torkestani, J., Meybodi, M.R., 2010. Learning automata-based algorithms for finding minimum weakly connected dominating set in stochastic graphs. Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 18, 721–758.

Torkestani, J.A., Meybodi, M.R., 2011a. Learning automata-based algorithms for solving stochastic minimum spanning tree problem. Appl. Soft Comput. 11, 4064–4077.

Torkestani, J.A., Meybodi, M.R., 2011b. Weighted Steiner connected dominating set and its application to multicast routing in wireless MANETs. Wirel. Pers. Commun. 60, 145–169.

Torkestani, J.A., Meybodi, M.R., 2012a. Finding minimum weight connected dominating set in stochastic graph based on learning automata. Inf. Sci. 200, 57–77.

Torkestani, J.A., Meybodi, M.R., 2012b. A learning automata-based heuristic algorithm for solving the minimum spanning tree problem in stochastic graphs. J. Supercomput. 59, 1035–1054.

Torkestani, J.A., Meybodi, M.R., 2012c. A new vertex coloring algorithm based on variable aaction-set learning automata. Comput. Inform. 29, 447–466.

Tsetlin, M.L., 1973. Automaton Theory and the Modelling of Biological Systems. Academic Press, New York and London.

Vafashoar, R., Meybodi, M.R., Momeni Azandaryani, A.H., 2012. CLA-DE: a hybrid model based on cellular learning automata for numerical optimization. Appl. Intell. 36, 735–748.

Vahidipour, S.M., Esnaashari, M., 2017. Priority assignment in queuing systems with unknown characteristics using learning automata and adaptive stochastic Petri nets. J. Comput. Sci. http://dx.doi.org/10.1016/j.jocs.2017.08.009.

Vahidipour, S.M., Meybodi, M.R., Esnaashari, M., 2015. Learning automata-based adaptive Petri net and its application to priority assignment in queuing systems with unknown parameters. IEEE Trans. Syst. Man Cybern. Syst. 45, 1373–1384.

Vahidipour, S.M., Meybodi, M.R., Esnaashari, M., 2017a. Finding the shortest path in stochastic graphs using learning automata and adaptive stochastic Petri nets. Internat. J. Uncertain. Fuzziness Knowledge-Based Systems 25, 427–455.

Vahidipour, S.M., Meybodi, M.R., Esnaashari, M., 2017b. Adaptive Petri net based on irregular cellular learning automata with an application to vertex coloring problem. Appl. Intell. 46, 272–284.

Vahidipour, S.M., Meybodi, M.R., Esnaashari, M., 2017c. Cellular adaptive Petri net based on learning automata and its application to the vertex coloring problem. Discrete Event Dyn. Syst. 1–32.

Williams, R.J., 1988. Toward a Theory of Reinforcement-Learning Connectionist Systems. Northeastern University.

Zaitsev, D.A., 2013. Verification of computing grids with special edge conditions by infinite Petri nets. Autom. Control Comput. Sci. 47, 403–412.
Zaitsev, D.A., 2015a. Universality in infinite Petri nets. In: Proceedings of 7th International Conference. MCU 2015. In: Lecture Notes in Computer Science, vol. 9288, pp. 180–197.
Zaitsev, D.A., 2015b. Simulating cellular automata by infinite synchronous Petri nets. In: 21st Annual International Workshop on Cellular Automata and Discrete Complex Systems. Turku. Finland. TUCS Lecture Notes. Vol. 24, pp. 91-100.
Zaitsev, D.A., 2016. Sleptsov nets run fast. IEEE Trans. Syst. Man Cybern. Syst. 46, 682–693.
Zaitsev, D.A., Zaitsev, I.D., Shmeleva, T.R., 2014. Infinite Petri nets as models of grids. In: Khosrow-Pour, Mehdi (Ed.), Encyclopedia of Information Science and Technology, Vol. 10, third ed. pp. 187–204, (Chapter 19).
Zhang, J., Wang, C., Zhou, M., 2014. Last-position elimination-based learning automata. IEEE Trans. Cybern. 44, 2484–2492.

S. Mehdi Vahidipour received the BSc and MSc degrees in computer engineering in Iran, in 2000 and 2003, respectively. He received the Ph.D. degree in computer engineering at the Computer Engineering & Information Technology Department from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2016. Currently, he is an Assistant Professor in Electrical and Computer Engineering department, University of Kashan. His research interests include distributed artificial intelligence, learning automata, and Adaptive Petri nets.

Mehdi Esnaashari received the B.S., M.S., and the Ph.D. degrees in computer engineering, all from the Amirkabir University of Technology, Tehran, Iran, in 2002, 2005, and 2011, respectively. Prior to his current position, he was an Assistant Professor with Iran Telecommunications Research Center, Tehran. He is currently an Assistant Professor with faculty of computer engineering, K. N. Toosi University of Technology, Tehran, Iran. His current research interests include computer networks, learning systems, soft computing, and information retrieval.

Alireza Rezvanian was born in Hamedan, Iran, in 1984. He received the B.Sc. degree from Bu-Ali Sina University of Hamedan, Iran, in 2007, the M.Sc. degree in Computer Engineering with honors from Islamic Azad University of Qazvin, Iran, in 2010, and the Ph.D. degree in Computer Engineering at the Computer Engineering & Information Technology Department from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2016. Currently, he works as a researcher in School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. Prior to the current position, he joined the Department of Computer Engineering and Information Technology at Hamedan University of Technology, Hamedan, Iran as a lecturer. He has authored or co-authored more than 60 research publications in reputable peer-reviewed journals and conferences including IEEE, Elsevier, Springer, Wiley and Taylor & Francis. He has organized various special sessions in international conferences in his area of expertise in Iran and abroad. He is a member of Technical Program Committees (TPCs) of various IEEE sponsored conferences in Iran and abroad. He has been a reviewer of many reputable conferences and journals. His research activities include soft computing, evolutionary algorithms, complex social networks, and learning automata.

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.