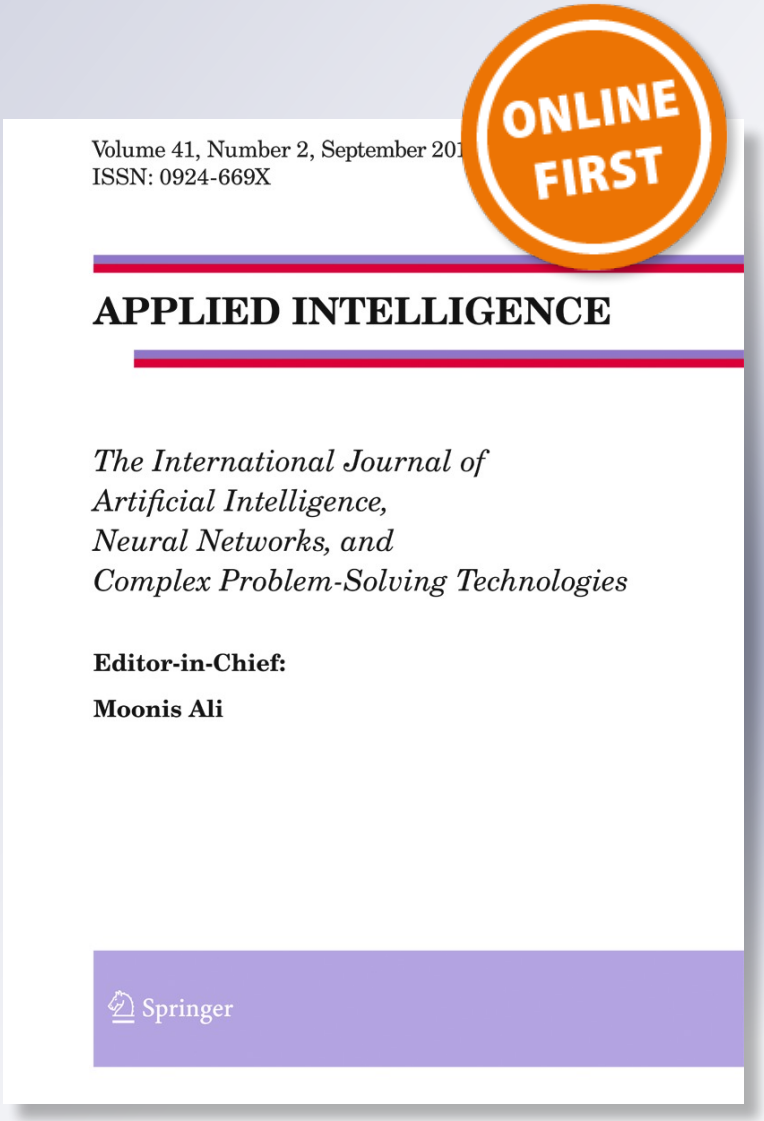
Extended distributed learning automata

Mohammad Reza Mollakhalili Meybodi & Mohammad Reza Meybodi

Applied Intelligence
The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies

ISSN 0924-669X

Appl Intell
DOI 10.1007/s10489-014-0577-2



Springer

Appl Intell
DOI 10.1007/s10489-014-0577-2

Extended distributed learning automata

An automata-based framework for solving stochastic graph optimization problems

Mohammad Reza Mollakhalili Meybodi · Mohammad Reza Meybodi

Abstract In this paper, a new structure for cooperative learning automata called extended learning automata (eDLA) is introduced. Based on the new structure, an iterative randomized heuristic algorithm using sampling is proposed for finding an optimal subgraph in a stochastic edge-weighted graph. Stochastic graphs are graphs in which the weights of edges have an unknown probability distribution. The proposed algorithm uses an eDLA to find a policy that leads to a subgraph that satisfy some restrictions such as minimum or maximum weight (length). At each stage of the proposed algorithm, the eDLA determines which edges should be sampled. The proposed eDLA-based sampling method may reduce unnecessary samples and hence decrease the time required for finding an optimal subgraph. It is shown that the proposed method converges to an optimal solution, the probability of which can be made arbitrarily close to 1 by using a sufficiently small learning parameter. A new variance-aware threshold value is also proposed that can significantly improve the convergence rate of the proposed eDLA-based algorithm. It is further shown that our algorithm is competitive in terms of the quality of the solution.

M. R. Mollakhalili Meybodi (✉)
Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
e-mail: Mollakhalili@maybodiau.ac.ir

M. R. Meybodi
Soft Computing Laboratory, Amirkabir University of Technology, Tehran, Iran
e-mail: mmeybodi@aut.ac.ir

Present Address:
M. R. Mollakhalili Meybodi
Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran

Keywords Distributed learning automata (DLA) · Extended distributed learning automata (eDLA) · Learning automata (LA) · Stochastic graph · Stochastic subgraph · Sampling

1 Introduction

Automata models for learning systems were introduced in the 1960s and were popularized by Narendra in [1] as learning automata (LA). There have since been many advances in the theory and application of these learning models [2]. The most important among these are groups of LAs that form teams or feed forward networks and have been shown to converge to desired solutions under a proper choice of learning rate [2, 3].

Learning automata are essentially simple agents for doing simple things. Recently, LAs have been applied to a wide range of science and engineering applications (e.g., [4, 5]). The full potential of LAs is realized when a group of automata interacts with each other to solve a given problem. One of the interconnected structures of learning automata is (DLA), the complete version of which was introduced and applied in [6, 7].

As a network of learning automata which cooperates collectively in a random environment to solve a particular problem, DLAs have repeatedly been used to solve various problems, especially those related to random graphs with weighted edges or vertices [3, 7, 8].

One of the obvious drawbacks of DLAs as distributed multi-agent systems is that the next active agent is directly selected by the action taken by the current active agent. In other words, in a DLA only one LA is active at a given time and is able to take action on the environment. The next active LA specified by the selected action of the

Published online: 23 August 2014

current active LA. The LA activation mechanism in DLAs has limited application; however, this network has been able to solve various problems in random graphs such as finding shortest paths [6, 8–10], and problems in other areas such as web document clustering[11], web page ranking [12], link prediction in adaptive web sites [13], user modeling in adaptive hypermedia [14], web mining [15, 16] and so on.

In this paper, the extended approach to the activation mechanism of LAs in a DLA is introduced and is referred to as eDLA. Then, an algorithm based on eDLA is proposed to find optimal subgraphs in stochastic graphs. The rest of the paper is organized as follows. In Section 2, LAs and graph theory including stochastic graphs are briefly discussed and eDLA is introduced. In Section 3, a new eDLA-based proposed algorithm is presented. Section 4 is aimed at proving the convergence of the proposed algorithm. Section 5 evaluates the performance of the proposed algorithm through simulation experiments. Finally, in the last section the findings of this paper are discussed.

2 Preliminaries

To better understand the algorithms discussed in this paper, we briefly describe the concepts of stochastic graphs, LAs, and DLAs in this section.

2.1 Stochastic graphs

A stochastic graph G is denoted by a triple $G = (V, E, Q)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of nodes, $E \subseteq V \times V$ is a set of edges, and the $n \times n$ matrix $Q_{n\times n}$ is the probability distribution describing the statistics of edge lengths where n is the number of nodes. For each edge $e_{(i,j)} \in E$ the associated weight w_{ij} is a positive random variable with the probability density function (PDF) q_{ij}, which is assumed to be unknown in this paper. As a result, any proposed algorithm in this paper is based on the assumption that q_{ij} is not known a priori.

2.2 Stochastic learning automata

A stochastic LA is an adaptive decision making unit that improves its performance by learning how to choose the optimal action from a finite set of allowable actions through repeated interactions with a random environment.

At each instant, the automaton chooses an action from its available actions, based on a probability distribution over the action set. The selected action serves as the input to the random environment. The environment responds to the action with a reinforcement signal. Based on the reinforcement feedback from the environment, the action probability vector is updated using a learning process that is governed by the learning algorithm. The objective of this process is to find the optimal action from the action-set so that the average penalty received from the environment is minimized. Stochastic LAs can be classified into two main families: *fixed structure* (FSLAs) and *variable structure* (VSLAs).

A VSLA is represented by a triple $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, T\}$ where:

- $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of actions (or outputs of the automaton). The output or action of the automaton at the instant n, denoted by $\alpha(n)$, is an element of the finite set $\boldsymbol{\alpha}$.
- r is the number of available actions.
- $\boldsymbol{\beta} = \{0, 1\}$ is a set of responses from the environment (or inputs to the automaton).
- T denotes a learning algorithm, which is a recurrence relation used to modify the action selection probability vector of the automaton.

The relationship between an LA and its random environment is shown in Fig. 1

At each step k, the automaton samples an action $\alpha_i(k) \in \boldsymbol{\alpha}$ according to the action selection probability vector $\mathbf{p}(k)$. The selected action serves as the input to the environment. The environment responds to the input with a stochastic response $\beta(k) \in \boldsymbol{\beta}$. Based on $\beta(k)$, $\mathbf{p}(k)$ is updated and a new action is chosen at the next step. The environment rewards ($\beta(k) = 1$) an action α_i of the automaton with reward probability d_i ($c_i = 1 - d_i$ denotes the penalty probability of action α_i).

The recurrence equations shown in (1) and (2) is a *linear learning algorithm* that is used to update the action selection probability vector $\mathbf{p}(k)$:

$$p_j(k+1) = \begin{cases} (1-a) \times p_j(k) + a & j = i \\ (1-a) \times p_j(k) & \forall j \neq i \end{cases} \tag{1}$$

when the action $\alpha_i(k)$ is rewarded by the environment (i.e. $\beta(k) = 1$), and

$$p_j(k+1) = \begin{cases} (1-b) \times p_j(k) & j = i \\ (1-b) \times p_j(k) + \frac{b}{r-1} & \forall j \neq i \end{cases} \tag{2}$$

when the action $\alpha_i(k)$ is penalized by the environment (i.e. $\beta(k) = 0$).

If $a = b$, then the recurrence equations (1) and (2) are called a linear reward-penalty (L_{R-P}) algorithm. If $a \gg b$,

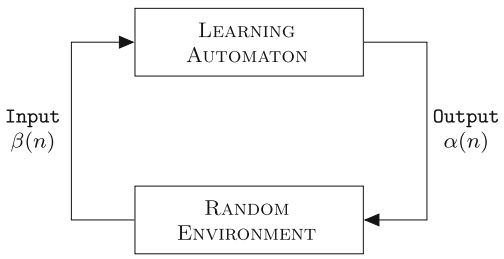


Fig. 1 Relationship between the LA and its random environment

then the given equations are called a linear reward-ϵ penalty ($L_{R-\epsilon P}$) algorithm, and finally if $b = 0$ then they are called a linear reward-inaction (L_{R-I}) algorithm. In the latter case, the action probability vectors remain unchanged when the action taken is penalized by the environment.

2.2.1 Variable action set learning automaton

If the number of actions of an LA varies over time, it is called a *variable action set LA*. The absolute expediency and ϵ-optimality of this type of LA under the L_{R-I} reinforcement scheme is given in [17].

Assume that $V(n) \subset \boldsymbol{\alpha}$ is a non-empty subset of the actions of the LA at time n. $V(n)$ represents the available (or selectable) actions at time n, called *active actions*. Selecting the elements of $V(n)$ is done randomly by an external factor. The procedure of selecting an action and updating the action probability vector in this type of LA can be described as follows.

Let $V(n)$ be the set of active actions at time n, and let $K(n) = \sum_{\alpha_i \in V(n)} p_i(n)$ represent the sum of probabilities of the active actions. Before the selection of an action, the active actions probability vector is *scaled* according to (3):

$$\forall \alpha_i \in V(n) : \widehat{p}_i(n) = \frac{p_i(n)}{K(n)} \tag{3}$$

The LA randomly then selects an action according to the scaled action probability vector $\widehat{\mathbf{p}}(n) = \{\widehat{p}_i(n) \mid \alpha_i \in V(n)\}$. According to the response received from the environment, the scaled action probability vector $\widehat{\mathbf{p}}(n)$ is updated. Finally the active action probability vector $\widehat{\mathbf{p}}(n)$ is *rescaled* following (4)

$$\forall \alpha_i \in V(n) : \widehat{p}_i(n) = p_i(n) \times K(n) \tag{4}$$

2.2.2 Distributed learning automata

DLAs are networks of automata that cooperate to solve particular problem. A DLA can be modelled by a directed graph in which the set of nodes constitutes the set of automata and the set of outgoing edges for each node constitutes the set of actions for the corresponding automaton. When an automaton selects one of its actions, the automaton at the other end of the edge corresponding to the selected action is activated. An example of DLA is given in Fig. 2. In this example, if automaton LA_1 selects action $\alpha 13$, then automaton LA_3 will be activated. The activated automaton then chooses one of its actions and so on. At any time only one automaton in the network is active.

Formally, a DLA can be embedded in a graph and can be defined as a 4-tuple $(\mathbf{A}, E, T, A^0)$, where $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$is the set of LAs, $E \subseteq A \times A$ is the set of the edges with the edge $e_{(i,j)}$ corresponding to the action a_{ij} of automaton A_i. T is the set of learning schemes with

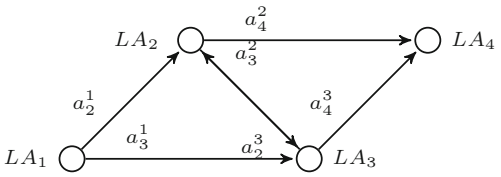


Fig. 2 Distributed Learning Automata

which the LAs update their action probability vectors, and A_0 is the root automaton of the DLA from which activation is started. The operation of a DLA can be described as follows:

- On each activation of the root node, one of its outgoing edges (one action of the root automaton) is chosen using the corresponding action probability vector.
- The selected edge activates the LA at the other end of the selected edge.
- This automaton also selects an action that results in activation of another automaton.
- This process is repeated until a leaf node is reached.
- The leaf node interacts with the environment.

A restricted version of DLA is introduced in [18], where the underlying graph is a directed acyclic graph and an L_{R-I} algorithm with decaying reward parameters is used as the learning algorithm. Meybodi and Beigy introduce a DLA in which the underlying graph is not necessarily acyclic, but to restrict an LA from appearing more than once in any path, a varying number of actions are used [7].

3 Extended distributed learning automata

An eDLA is a network of interconnected cooperative LAs supervised by a set of communication rules governing the order of operation of the LAs. In an eDLA, each LA has an activity level that changes according to the problem to be solved by the eDLA and the communication rules. At any time, only one automaton has a high activity level and can perform an action on the environment.

Formally, an eDLA can be embedded in a graph and defined by a 7-tuple

$$eDLA = \{A, E, S, P, S^0, F, C\}$$

where A is the set of vertices and E is the set of edges of $G = (A, E)$, called the *communication graph*, and $S = \{s_1, s_2, \ldots, s_n\}$ is a set of activity levels corresponding to each LA in the eDLA with the activity level of A_i denoted by s_i. Each automaton can have one of the following activity levels: ***Passive***, **Active**, ***Fire*** and ***Off***, represented by **Pa**, **Ac**, **Fi** and **Of** respectively.

Fi is the highest level of activity, and *Pa* and *Of* are the lowest levels of activity. The difference between *Pa* and *Of*

is that when an LA downgrades to the *Of* level of activity, it cannot change to other levels, whereas in the *Pa* level it is possible upgrade to higher levels. At any time, only one of the automata in the eDLA will be in the *Fi* level of activity. This automaton is determined by the function C. In an eDLA, there is a finite set of rules, P, that governs the activity levels of each automaton. These rules are based on the current activity level of each automaton and its parents or adjacent automata in G. Such rules determine the next automaton activity level. These rules may vary depending on the problem being solved by the eDLA.

$S^0 = (s_1^0, s_2^0, \ldots, s_n^0)$ is the *initial state* of the eDLA, and $F = \{S^F | S^F = (s_1^F, s_2^F, \ldots, s_n^F)\}$ is *final conditions*. F is a set of circumstances in terms of automata activity levels such that, if at least one is realized, the eDLA is transferred to the final state. Obviously F has at least one element where the activity level of all automata in the eDLA is *Of*. C is a special function that selects one automaton in the eDLA. This selection is based on the current activity level of each automaton in the eDLA and the associated problem that the eDLA is designed to solve. This function is called the *Fire Function*, and is explained in the following discussion.

The number of actions for a particular automaton is equal to the degree (if G is an undirected graph) or out degree (if G is a directed graph) of the corresponding node in the communication graph.

The eDLA works as follows:

- First, a network of LAs that is isomorphic to an input graph is created. In this network, each node is an LA and each edge (or outgoing edge) of this node is an actions of this LA.
- Second, the eDLA starts from state S^0 and, based on the rules P, the activity level of each LA in the eDLA changes. This changes the state of eDLA from S^0 to S^1. Transitions from state S^i to S^{i+1} continue until the eDLA reaches its final state(s).
- At each time, a node with high activity level, selects an action from its available actions and applies it to the environment. The *Fi* automaton then switches to a low activity level *Of*. Moreover, based on the communication rules, a set of adjacent automata with the lowest level of activity (*Pa*) upgrades to the next level of activity (*Ac*). The next *Fi* automaton is selected from the set of automata with the *Ac* level of activity. This selection is often done randomly.

Run Run is the change of automaton activity level in an eDLA to *Of* level until at least one of the final conditions is realized. The activity level change is based on the activity change cycle Pa $\rightarrow$ Ac $\rightarrow$ Fi $\rightarrow$ Of. At the beginning of each run, there is at least one automaton with *Ac* level of activity called the *root*.

Instantaneous Description This is an ordered 4-tuple

$$D^t = (D_{Of}^t, D_{Fi}^t, D_{Ac}^t, D_{Pa}^t)$$

where D_{Of}^t, D_{Fi}^t, D_{Ac}^t and D_{Pa}^t denote the sets of automata with *Of*, *Fi*, *Ac* and *Pa* levels of activity respectively.

When an LA in the eDLA performs one of its actions, the instantaneous description changes based on set of rules P. Moreover, the instantaneous description changes when the activity level of an LA in the eDLA changes to the *Fi* level. The initial instantaneous description of the eDLA, or D^0, is defined as

$$D^0 = (D_{Of}^0, D_{Fi}^0, D_{Ac}^0, D_{Pa}^0) = (\phi, \phi, \{A_0\}, A - \{A_0\})$$

and the default final instantaneous description is defined as: $D^{final} = (D_{Of}^{final}, D_{Fi}^{final}, D_{Ac}^{final}, D_{Pa}^{final}) = (A, \phi, \phi, \phi)$. Based on the above definitions and notation, a run in an eDLA can be described by a sequence of instantaneous descriptions as follows:

$$Run \equiv D^0 \succ^{fire} D^1 \succ^{action} D^2 \succ^{fire} \ldots \succ D^n \in F$$

Thus, an eDLA is a dichotomous structure consisting of a fire function and a network of LAs that cooperate to solve a specific problem. The fire function differs depending on the problem, but in general the task of the fire function is to determine the automaton that should perform an action in the random environment.

3.1 Solving the stochastic subgraph problem by eDLA

Stochastic graphs are a suitable tool for modelling many real world problems. For example, communication links in computer networks between switching elements such as routers and switches can be modelled as a random edge-weighted graph. Suppose we want to find a subgraph with certain restrictions in such a stochastic graph. For example we might find a subgraph that is a spanning tree with minimum total weight. Finding an optimal subgraph in a stochastic graph when the probability distribution function of the edge weight is unknown, in general, a #P-hard problem. This problem is called the stochastic subgraph problem.

One solution to this problem is sampling from the edges to estimate the deterministic mean value of edge weights and then to solve the problem in a deterministically equivalent graph obtained by replacing stochastic edges by deterministic edges. This requires a large number of samples. Another solution to this problem is purposeful sampling. One such method is sampling using an eDLA. By using an eDLA, sampling is done from more efficient edges. This sampling method may result in better running time for the algorithm by decreasing the number of unnecessary

samples. The general procedure used by the eDLA-based algorithm is shown in Algorithm 1.

Algorithm 1 eDLA-Based Algorithm

Input: Stochastic Graph $G = (V, E)$, P_s; K_s;
Output: Sub-graph s;

Assign an LA to each node of G and construct an eDLA=(V,E)
Let s denote the constructed sub-graph;
Let W_s denotes the weight of the constructed sub-graph;
$P \leftarrow 0$;
$K \leftarrow 0$;
repeat
  $s \leftarrow \phi$;
  $W_s \leftarrow 0$;
  v=`ROOT()`; ▷ ROOT() determines root
  $A_P \leftarrow V - \{v\}$; ▷ A_P = set of LAs with Pa level
  $A_A \leftarrow v$; ▷ A_P = set of LAs with Ac level
  $A_O \leftarrow \phi$; ▷ A_O = set of LAs with Of level
  $A_F \leftarrow$ `Fire`$(eDLA)$; ▷ A_F has Fi level
  while $A_F \neq \phi$ **do**
    $\alpha \leftarrow$ `SelectAction`(A_F);
    disable action α in adjacent active LA;
    $s \leftarrow s \cup \text{edge}(A_F, \alpha)$;
    $W_s \leftarrow W_s +$ `SampledWeight`$(\text{edge}(A_F, \alpha))$;
    $A_O \leftarrow A_O \cup A_F$;
    $A_F \leftarrow$ `Fire`$(eDLA)$;
  end while
  $K \leftarrow K + 1$;
  response=`Evaluate`$(W_s$,Threshold,K);
  for all $LA \in A_O$ **do**
    if response = REWARD **then**
      `reward` selected action by LA; ▷ L_{R-I}
    end if
    enable all disabled actions; ▷ for next run
  end for
  Threshold=`Update`$(W_s$,Threshold,K);
  $q = \prod P_{\alpha_n}^{A_m} \quad \forall \text{edge}(A_m, \alpha_n) \in s$;
until $(K \geq K_s \vee q \geq P_s)$

As can be seen in the pseudo-code of Algorithm 1, several functions are used by the algorithm:

- `ROOT`: As mentioned in the description of eDLA, at the beginning of each run, there is at least one automaton with the *Ac* level of activity called *root* and determined by the `ROOT` function. The mechanism for determining *root* is dependent on the underlying problem.
- `Fire`: This function indicates at each stage one automaton for which to upgrade the activity level to fire. This done according to the activity level of each automaton and the communication rules used by eDLA (e.g. it randomly selects an automaton from those with the *Ac* level of activity). Furthermore, based on the rules P of the eDLA and the fire automaton, adjacent automata with the *Pa* level of activity upgraded to the *Ac* level.
- `SelectAction`: This function selects an action from the available actions set of the current automaton with the highest level of activity (Fire). This selection is done according to the selection action probability vector for the specified automaton. The activity level of the specified automaton is then downgraded to *Of*. After selecting an action, the actions that can cause loops must be disabled in the actions set for the other automata with the *Ac* level of activity. The algorithm used to avoid the creation of loops is described later.
- `Evaluate`: This function is used by the environment to evaluate the action selected by the eDLA. The action of an eDLA is a sequence of automaton actions that represents a particular subgraph in the stochastic graph. The environment uses the sample length of this subgraph (W_s) to produce its response. This response, depending on whether it is unfavorable or favorable, causes the actions corresponding to subgraph edges to be penalized or rewarded respectively.

  After evaluation of an eDLA action, all of the disabled actions are enabled and the activity level of all automata in the eDLA is set to *Pa*.
- `Update` function: Assume that the subgraph s is selected at stage k. The average weight of all previously constructed subgraphs is called the *dynamic threshold* and is denoted by T. At each stage $n > 1$, the dynamic threshold T_n is computed as

$$T = T_n = \frac{1}{n}\sum_{j=1}^{n} W_s(j) = \frac{W_s(n) + (n-1)T_{n-1}}{n} \quad (5)$$

  where $W_s(j)$ denotes the weight of the sampled subgraph s at the j^{th} iteration of the algorithm and is defined as:

$$W_s(j) = \sum_{\forall e_{(t,u)} \in s} w_{(t,u)}(j)$$

  $w_{(t,u)}(j)$ is a positive random variable and shows the sampled weight of edge $e_{(t,u)}$ at the j^{th} iteration of the algorithm.

The `Fire` and `SelectAction` functions are repeated until all automata in the eDLA downgrade to the *Of* activity level. The length of the subgraph is then computed and compared with the dynamic threshold. Depending on the underlying problem, the constraints to be satisfied by the subgraph and the result of this comparison, all the LAs with the *Of* activity level update their action selection probability vector (`Evaluate` function) and the dynamic threshold is updated according to the update function described above.

The process of constructing subgraphs is repeated until the stopping conditions are reached. At this point, the last constructed subgraph is the subgraph that satisfies specific constraints. The stop conditions can be either the probability of choosing edges of the subgraph, called the subgraph probability, being greater than a certain pre-defined threshold (denoted by P_s in Algorithm 1) or selecting a

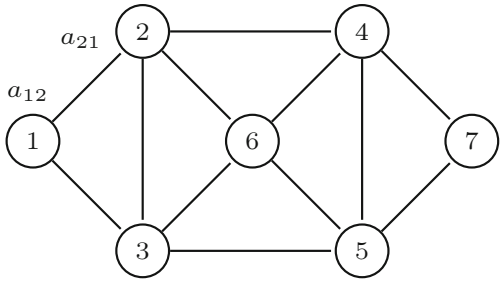


Fig. 3 Example graph for the stochastic minimum spanning tree problem

pre-specified number (denoted by K_s in Algorithm 1) of subgraphs.

To better understand the proposed algorithm and some concepts related to eDLAs, Fig. 3 represents an application of the algorithm to finding a stochastic minimum spanning tree (SMST). Initially, all automata are in the *Pa* level of activity. Before starting the run in the minimum spanning tree problem, the function `ROOT()` randomly selects an automaton from those with the *Pa* level of activity and upgrades the activity level of the selected automaton to *Ac*. The function `Fire()` randomly selects an automaton from those with the *Ac* level of activity and upgrades the activity level of the selected automaton to *Fi*. The result of a run is shown in (Fig. 4).

$$
\begin{aligned}
&(\phi, \phi, \phi, \{1,3,4,5,6,7\}) \succ^{\text{ROOT}} \\
&(\phi, \phi, \{2\}, \{1,3,4,5,6,7\}) \succ^{\text{fire(2)}} \\
&(\phi, \{2\}, \{1,3,4,6\}, \{5,7\}) \succ^{\text{action}(a_{23})} \\
&(\{2\}, \phi, \{1,3,4,6\}, \{5,7\}) \succ^{\text{fire(1)}} \\
&(\{2\}, \{1\}, \{3,4,6\}\{5,7\}) \succ^{\text{action}(a_{12})} \\
&(\{2,1\}, \phi, \{3,4,6\}, \{5,7\}) \succ^{\text{fire(6)}} \\
&(\{2,1\}, \{6\}, \{3,4,5\}, \{7\}) \succ^{\text{action}(a_{64})} \\
&(\{2,1,6\}, \phi, \{3,4,5\}, \{7\}) \succ^{\text{fire(3)}} \\
&(\{2,1,6\}, \{3\}, \{4,5\}, \{7\}) \succ^{\text{action}(a_{35})} \\
&(\{2,1,6,3\}, \phi, \{4,5\}, \{7\}) \succ^{\text{fire(4)}} \\
&(\{2,1,6,3\}, \{4\}, \{5,7\}, \phi) \succ^{\text{action}(a_{42})} \\
&(\{2,1,6,3,4\}, \phi, \{5,7\}, \phi) \succ^{\text{fire(7)}} \\
&(\{2,1,6,3,4\}, \{7\}, \{5\}, \phi) \succ^{\text{action}(a_{74})} \\
&(\{2,1,6,3,4,7\}, \phi, \{5\}, \phi) \succ^{\text{fire(5)}} \\
&(\{2,1,6,3,4,7\}, \{5\}, \phi, \phi) \succ^{\text{no-action}} \\
&(\{2,1,6,3,4,7,5\}, \phi, \phi, \phi)
\end{aligned}
\tag{6}
$$

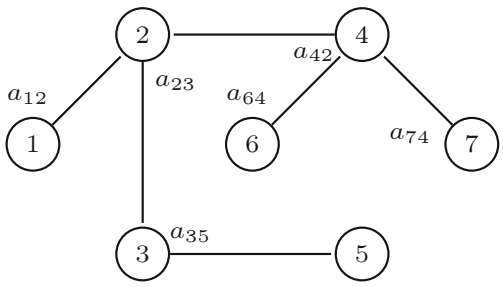


Fig. 4 Constructed spanning tree for the graph in Fig. 3 and the run shown in (6). The selected actions are displayed besides each automata

3.2 Improvement

Our experiments show that as the dynamic threshold value approaches the weight of the optimal subgraph, the eDLA learning ability decreases. To deal with this problem, we use a different criterion rather than the dynamic threshold, namely the average of the obtained subgraph weights. This criterion should be able to take into account the mean and variance of the weights of the sampled subgraphs. We reduce the computational complexity of the proposed improvement and improve its applicability as follows:

I- First, we use a simple estimate, such as the stochastic gradient, for the mean and variance.
II- Second, the weight of individual edges is not observable but the total weight, calculated as the sum of the weights of all edges, is observable.

Thus, the stochastic graph is considered to be a black box and at each step, any proposed improvement only has access to the weight of the current sampled subgraph and the estimated mean and variance of the previously sampled subgraphs. Therefore, no additional assumption is imposed on the stochastic graph and the simplicity of the LAs is not violated.

To obtain a new criterion, at each stage k we do the following:

- Err_k is calculated as the difference between the weight of the subgraph (W_k) obtained at stage k and the estimated mean of the previously obtained subgraph weights (T_{k-1}). That is, $Err_k = W_k - T_{k-1}$.
- The estimated value of the mean is updated by $T_k = T_{k-1} + \alpha \times Err_k$.
- The estimated value of the variance is updated by

$$Var_k = Var_{k-1} + \beta \times (|Err_k| - Var_{k-1})$$

It is clear that Var_k estimates the mean deviation rather than the standard deviation, and we have:

$$
\begin{aligned}
\text{mdev}^2 &= \left(\sum |W - T|\right)^2 \\
&\geq \sum \left(|W - T|^2\right) = \delta^2 = \text{sdev}^2
\end{aligned}
\tag{7}
$$

This shows that the mean deviation is greater than the standard deviation and hence is more conservative than the standard deviation.

To evaluate the eDLA action in the proposed algorithm, we use a linear combination of the estimated mean and estimated variance. In minimization problems, ${}^{T_k}/_2 + 2 \times Var_k$ is used as an upper bound for comparison, and in maximization problems, ${}^{T_k}/_2 - 2 \times Var_k$ is used as a lower bound for comparison.

4 Asymptotic behavior of proposed eDLA-based algorithm

Learning automata can automatically improve their behavior based on a response from a random stationary environment, but when connected with each other, their behavior becomes complex and hard to analyze [18]. In this section, we analyze the behavior of the eDLA-based proposed algorithm (Algorithm 1) for finding an optimal subgraph in stochastic graph. For this purpose, we use a method similar to that proposed by [1] and [19] for studying the convergence of a linear learning algorithm and in [7] for studying the asymptotic behavior of DLAs.

There are essentially two types of convergence for reinforcement schemes reported in the literature [1]. In the first type of convergence that typically occurs in expedient schemes (e.g. L_{R-P}), the distribution functions for the sequence of action probabilities converge to a distribution function. For example, it can be shown that when the L_{R-P} algorithm is used, the selection action probability, $p(n)$, converges to a random variable with a continuous distribution and computable variance, and hence can be made as small as desired by the proper choice of parameters.

The second and stronger type of convergence occurs in the case of ϵ-optimal schemes (such as L_{R-I} schemes). By using the martingale convergence theorem, it can be proved that the sequence of action probabilities converges to a limiting random variable with probability one [19].

In [7], the authors introduce the concept of path probability in DLAs. Using a method similar to that in [20], they prove that if a stochastic graph has a unique shortest path, the path probability of the shortest path converges to one in the proposed algorithm and consequently the algorithm converges to the shortest path with probability close to unity. In this section, the proposed method in [7] is used to prove convergence of our proposed algorithm.

Throughout this section, the following notation is taken into account. Assume that, in the proposed algorithm (Algorithm 1) up to stage k, subgraph s_i (for $i = 1, 2, \ldots, r$ where r is the number of subgraphs) is selected k_i times, where $k = \sum_{i=1}^{r} k_i$. Also, assuming that subgraph s_i is selected at stage $k+1$. Let $W_{s_i}(j)$ denote the weight of s_i (for $i = 1, 2, \ldots, r$) at the j^{th} sampling time (for $j = 1, 2, \ldots, k$) in the proposed algorithm, we have:

$$d_i(k) = \Pr[W_{s_i}(k+1) \text{ satisfies } T_k] \tag{8}$$
$$c_i(k) = \Pr[W_{s_i}(k+1) \text{ does not satisfy } T_k] = 1 - d_i(k)$$

where $c_i(k)$ and $d_i(k)$ denote the penalty and reward probabilities for subgraph s_i at stage k respectively, and T_k is the dynamic threshold value as described in (5) (Section 3.1). The probability of subgraph s_i is defined as the product of the probabilities of selecting each edge of s_i and is denoted by q_i. $q_i(k)$ represents the probability of choosing subgraph s_i at stage k.

Theorem 1 *Assume that s_i is the optimal subgraph. If $\mathbf{q} = [q_i]_{i=1,2,\ldots,r}$ is updated according to the proposed algorithm, then* $\lim_{n\to\infty} q_i(n) \overset{a.s.}{=} 1$.

In other words, Theorem 1 asserts that the proposed algorithm converges to the optimal subgraph with probability as close to 1 as desired.

Proof This theorem is proved in multiple stages. First, $q_i(k)$ is computed as a function of the probabilities of actions of the LAs that construct subgraph s_i. Second, it is shown that, for large enough k, choosing the optimal subgraph by the proposed algorithm is a sub-martingale process. In the next stage, using the martingale convergence theorem, convergence of the proposed algorithm to the optimal subgraph is shown. The proposed algorithm uses an L_{R-I} scheme as the learning algorithm for the automaton. In this scheme, the action probability vector of the eDLA, $\mathbf{q}(n)$, converges to the set of absorbing states with probability one (this is shown in the previous stage). It is noted that only one of these states is the desired one. We can only say that $\mathbf{q}(n)$ converges to the desired state with a positive probability. In the final stage of the proof, this probability is quantified by the method proposed by [20].

Lemma 1 *If $\mathbf{q}$ is updated according to the proposed algorithm, and $c_i(k)$ and $d_i(k)$ denote the penalty and reward probabilities for subgraph s_i at stage k as defined by (8), then we have*

$$\mathbb{E}[q_i(k+1)|\mathbf{q}(k)] = \sum_{j=1}^{r} q_j(k) \left[c_j(k) q_i(k) + d_j(k) \prod_{e_{(m,n)} \in s_i} \delta_n^m(k) \right] \tag{9}$$

where $\mathbb{E}[.]$ denotes mathematical expectation, and

$$\delta_n^m(k) = \begin{cases} p_n^m(k+1) = p_n^m(k)(1-a) + a & e_{(m,n)} \in s_j \\ p_n^m(k+1) = p_n^m(k)(1-a) & e_{(m,n)} \notin s_j \end{cases} \tag{10}$$

□

Proof The proposed algorithm uses the L_{R-I} reinforcement scheme to evaluate eDLA action. Assume that at stage k, subgraph s_j is selected by the eDLA. If s_j is penalized by the random environment (with probability $c_j(k)$),then the probability of choosing each subgraph in the graph remains unchanged. If (s_j) is rewarded by the random environment

(with probability $d_j(k)$), then for any other subgraph s_i, two cases are possible:

I- s_j and s_i have no common edge. In this case, all the edges of s_i are penalized and the probability of choosing s_i decreases.

II- s_j and s_i have some common edges. In this case, the common edges are rewarded and all other edges of s_i are penalized.

By the above description, we have:

$$\mathbb{E}[q_i(k+1)|\mathbf{q}(k)] = \sum_{j=1}^{r} \mathbb{E}\left[q_i(k+1)|\mathbf{q}(k), \alpha_j\right] \times p(\alpha_j|\mathbf{q}(k)) \tag{11}$$

In (11), α_j denotes all actions taken by the LAs that resulted in the selection of subgraph s_j by the eDLA.

However:

$$p(\alpha_j|\mathbf{q}(k)) = q_j(k) \tag{12}$$

and

$$\mathbb{E}\left[q_i(k+1)|\mathbf{q}(k), \alpha_j\right] = c_j(k)q_i(k) + d_j(k) \prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \left(p_n^m(k)\uparrow\right) \prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\notin s_j}} \left(p_n^m(k)\downarrow\right) \tag{13}$$

where $p_j^i(k)$ represents the probability of selecting edge $e_{(i,j)}$ or equivalently the probability that automaton A_i chooses action α_j (A_i has the highest level of activity or *Fi*) at stage k. In addition, $p_j^i(k)\downarrow$ and $p_j^i(k)\uparrow$ denote the decrease and increase of $p_j^i(k)$ respectively by the L_{R-I} learning algorithm, defined as follows:

$$\begin{aligned} p_j^i(k)\downarrow &= p_j^i(k) \times (1-a) \\ p_j^i(k)\uparrow &= p_j^i(k) \times (1-a) + a \end{aligned} \tag{14}$$

By substituting (12) and (13) into (11), we obtain:

$$\begin{aligned} \mathbb{E}[q_i(k+1)|\mathbf{q}(k)] &= \sum_{j=1}^{r} q_j(k)[c_j(k)q_i(k)] \\ &+ \sum_{j=1}^{r} \left[d_j(k) \prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} (p_n^m(k)\uparrow) \prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\notin s_j}} (p_n^m(k)\downarrow) \right] \\ &= \sum_{j=1}^{r} q_j(k) \left[c_j(k)q_i(k) + d_j(k) \prod_{e_{(m,n)}\in s_i} \delta_n^m(k) \right] \end{aligned} \tag{15}$$

This completes the proof. □

Lemma 2 *For a given stochastic graph, if s_i is the optimal subgraph and $\mathbf{q}(k)$ is updated according to the proposed algorithm, then the increment in the conditional expectation of $q_i(k)$ subject to $\mathbf{q}$,*

$$\triangle q_i(k) = \mathbb{E}[q_i(k+1) - q_i(k) \mid \mathbf{q}(k)]$$

is always non-negative.

Proof It is observed that the proposed algorithm chooses at most $(n-1)$ edges of the stochastic graph and forms one of r distinct subgraphs. After some algebraic operations, we obtain

$$\begin{aligned} \triangle q_i(k) &= \mathbb{E}[q_i(k+1) - q_i(k)|\mathbf{q}(k)] \\ &= \prod_{e_{(m,n)}\in s_i} \left(\mathbb{E}\left[p_n^m(k+1)|p_n^m(k)\right] - p_n^m(k)\right) \\ &= \prod_{e_{(m,n)}\in s_i} \triangle p_n^m(k) \end{aligned} \tag{16}$$

where

$$\triangle p_u^t(k)_{e_{(t,u)}\in s_i} = a\, p_u^t(k) \sum_{s\neq u}^{r_i} p_s^t(k)(c_s^t(k) - c_u^t(k)) \tag{17}$$

$\mathbf{p}^m(k)$ denotes the action selection probability vector for LA A_m in the eDLA, and $p_n^m(k)$ denotes the probability of selecting edge $e_{(m,n)}$ at stage k. In addition, $c_n^m(k)$ and $d_n^m(k) = 1 - c_n^m(k)$ denote the penalty and reward probabilities for edge $e_{(m,n)}$ respectively.

For all t, u we have $0 < p_u^t(k) < 1$. Since we assume that $e_{(t,u)} \in s_i$, from central limit theorem we conclude that, for large values of k,

$$c_v^t(k) - c_u^t(k) > 0 \quad (\forall v \neq u : \ e_{(t,v)} \notin s_i)$$

Therefore, for large values of k, the right hand side of (17) consists of non-negative quantities which implies that the right hand side of (16) is non-negative, completing the proof of Lemma 2. □

Corollary 1 *The only absorbing states of the Markov process $\{q_i(k)\}$ are 0 and 1.*

Proof Lemma 2 shows that $\{\mathbf{q}(n)\}$ is a sub-martingale. Since $\{\mathbf{q}(n)\}$ is non-negative and uniformly bounded, martingale theorems imply that

$$\lim_{k\to\infty} q_i(k) = q_i^*$$

exists with probability one. Furthermore, $\mathbf{q}(k+1) = \mathbf{q}(k)$ with probability one if and only if $q_i^* = 0$ or $q_i^* = 1$. Hence $\{0, 1\}$ constitutes the absorbing set for the Markov process $\{q_i(k)\}$. □

Let

$$S_r = \left\{ \mathbf{q}(k) \middle| q_i(k) \in [0,1];\ \sum_{i=1}^{r} q_i(k) = 1 \right\}$$

and

$$S_r^0 = \left\{ \mathbf{q}(k) \middle| q_i(k) \in (0,1);\ \sum_{i=1}^{r} q_i(k) = 1 \right\}$$

Corollary 1 implies that the set of all unit vectors in $S_r - S_r^0$ forms the set of all absorbing states of the Markov process $\{q_i(k)\}$. We will show that under some conditions for the optimum subgraph s_i, we have $q_i^* = 1$. Our method is very similar to methods given in [20] and [7, 21, 22].

Assume that $\mathbf{q}_i^* \in V_r$ denotes the state to which $\{q_i(k)\}$ converges, where $V_r = \{\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_r}\}$ denotes the set of all absorbing states for process $\{q_i(k)\}$. Define

$$\Gamma_i[\mathbf{q}] = \Pr\left[\mathbf{q}^* = \mathbf{e_i} | \mathbf{q}(0) = \mathbf{q}\right]$$

Here, $\Gamma_i[\mathbf{q}]$ denotes the probability of convergence of the proposed algorithm to a unit vector $\mathbf{e_i} \in V_r$ with initial probability vector $\mathbf{q}$. Assume that $D(S_r) : S_r \to \mathbb{R}$ is the class of all differentiable functions with bounded derivation on S_r. Such functions are necessarily continuous. If $\Psi(.) \in D$, then the operator

$$U\Psi(\mathbf{q}) = \mathbb{E}[\Psi(\mathbf{q}(k+1)) | \Psi(\mathbf{q}(k)) = \Psi(\mathbf{q})]$$

is linear and positive [19, 20]

It has been shown that $\Gamma_i[q]$ is the only continuous solution of $U\Gamma_i[\mathbf{q}] = \Gamma_i[\mathbf{q}]$ that satisfies the following boundary conditions [19, 20, 23]:

$$\Gamma_i[\mathbf{e_i}] = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{18}$$

However, the functional equation $U\Gamma_i[\mathbf{q}] = \Gamma_i[\mathbf{q}]$ is extremely difficult to solve. Hence the next best thing that can be done is to establish upper and lower bounds on $U\Gamma_i[\mathbf{q}]$. These can be computed by finding two functions $\Phi_1(.)$ and $\Phi_2(.)$ such that

$$\begin{aligned} U\Phi_1(p) &\geq \Phi_1(p) \\ U\Phi_2(p) &\leq \Phi_2(p) \end{aligned} \tag{19}$$

for all $p \in [0, 1]$ with appropriate boundary conditions.

The inequalities (19) imply that $\Phi_2(p) \leq \Gamma_i(p) \leq \Phi_1(p)$, and the functions $\Phi_1(.)$ and $\Phi_2(.)$ are called sub-regular and super-regular respectively.

It has been shown that these functions can have the following form [20]:

$$\Phi_i(p) = \frac{e^{x_i p} - 1}{e^{x_i} - 1} \quad i = 1, 2 \tag{20}$$

From Lemma 2, we know that if q_i^* denotes the probability of selecting optimal subgraph s_i by the eDLA then $q_i^* \in 0, 1$. Define

$$\Phi_i[x, \mathbf{q}] = \frac{e^{-\frac{xq_i}{a}} - 1}{e^{-\frac{x}{a}} - 1} = \frac{\Theta_i[x, \mathbf{q}] - 1}{\Theta_i[x, 1] - 1} \tag{21}$$

where $x > 0$ is to be chosen. $\Phi_i[x, \mathbf{q}] \in D(S_r)$ satisfies boundary conditions (18).

In the last stage of proof we show that $\Phi_i[x, \mathbf{q}]$ is sub-regular, and thus $\Phi_i[x, \mathbf{q}]$ qualifies as a lower bound on $\Gamma_i[\mathbf{q}]$.

We now determine conditions under which $\Phi_i[x, \mathbf{q}]$ is sub-regular. It is easy to see that the classes of super-regular and of sub-regular are closed under addition and multiplication by non-negative constants [20]. Furthermore, all constant functions are regular and hence both super- and sub-regular. For any $x > 0$, $\Theta_i[x, 1] > \Theta_i[x, 0] = 1$ and therefore $\Phi_i[x, \mathbf{q}]$ is super-regular (or sub-regular) if $\Theta_i[x, \mathbf{q}]$ is sub-regular (or super-regular). Most of our effort in the rest of this section goes into the proof of the next lemma.

Lemma 3 *Let s_i denote the optimum subgraph and $\Theta_i[x, \mathbf{q}] = e^{-xq_i/a}$ where $q_i = \prod_{e_{(m,n)} \in s_i} p_n^m$ is the probability of selecting s_i and a is the learning rate of the eDLA. Then there is a positive x such that $\Theta_i[x, \mathbf{q}]$ is super-regular.*

Proof From the definition of U, we have

$$U\Theta_i[x, \mathbf{q}] = \mathbb{E}[e^{-xq_i(k+1)/a} | \mathbf{q}(k) = \mathbf{q}] \tag{22}$$

From the definition of mathematical expectation and (22), we have

$$\begin{aligned} &\mathbb{E}[e^{-xq_i(k+1)/a} | \mathbf{q}(k) = \mathbf{q}] \\ &= \sum_{j=1}^{r} \mathbb{E}\left[e^{\frac{-x}{a}q_i(k+1)} | \mathbf{q}(k) = \mathbf{q}, s_j\right] \times \Pr[s_j | \mathbf{q}(k)] \\ &= q_j(k) \left\{ \sum_{j=1}^{r} d_j(k) e^{\frac{-x}{a}q_i(k+1)} + \sum_{j=1}^{r} (c_j(k)) e^{\frac{-x}{a}q_i(k)} \right\} \\ &= q_j(k) \{\mathbb{A} + \mathbb{B}\} \end{aligned} \tag{23}$$

For the sake of simplicity, e^x is written as $\exp(x)$, and $q_j(k)$ and $d_j(k)$ as q_j and d_j respectively. Algebraic simplification of (23) leads to (24) and (27):

$$\begin{aligned} q_j(k)\mathbb{A} &= \sum_{j=1}^{r} \exp\left\{-\frac{x}{a} q_i(k+1)\right\} d_j q_j \\ &= \exp\left\{ \frac{-x}{a} \prod_{e_{(m,n)} \in s_i} \{p_n^m + a(1 - p_n^m)\} \right\} q_j d_j \\ &\quad + \sum_{\substack{j=1 \\ j \neq i}}^{r} \exp\left\{ \frac{-x}{a} \mathbb{A}_1 \mathbb{A}_2 \right\} q_j d_j \end{aligned} \tag{24}$$

$$\mathbb{A}_1 = \prod_{\substack{e_{(m,n)} \in s_i \\ e_{(m,n)} \in s_j}} \{p_n^m + a(1 - p_n^m)\} \tag{25}$$

$$\mathbb{A}_2 = \prod_{\substack{e_{(m,n)} \in s_i \\ e_{(m,n)} \notin s_j}} \{p_n^m (1 - a)\} \tag{26}$$

and

$$q_j(k)\mathbb{B} = \sum_{j=1}^{r} \exp\left\{\frac{-x}{a} q_i(k)\right\} q_j(1-d_j)$$
$$= \Theta_i[x,\mathbf{q}] \sum_{j=1}^{r} q_j(1-d_j)$$
$$= \Theta_i[x,\mathbf{q}] - \Theta_i[x,\mathbf{q}] \sum_{j=1}^{r} q_j d_j \quad (27)$$

In (26), we have:

$$\mathbb{A}_2 = \prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\notin s_j}} \{p_n^m(1-a)\}$$
$$= \frac{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\notin s_j}} \{p_n^m(1-a)\} \prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)\}}{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)\}} \quad (28)$$
$$= \frac{\prod_{e_{(m,n)}\in s_i} \{p_n^m(1-a)\}}{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)\}}$$

Considering that $q_i = \prod_{e_{(m,n)}\in s_i} p_n^m$ along with (23) through (28), we have:

$$U\Theta_i[x,\mathbf{q}] = q_i d_i \exp\left\{-\frac{x}{a}(q_i + a(1-q_i))\right\}$$
$$+ \sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j \exp\left\{\frac{-x}{a} q_i(1-a) \frac{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)+a\}}{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)\}}\right\}$$
$$+\Theta_i[x,\mathbf{q}] - \Theta_i[x,\mathbf{q}] \sum_{j=1}^{r} q_j d_j \quad (29)$$

It is clear that

$$\frac{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)+a\}}{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)\}}$$
$$= \prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \frac{p_n^m(1-a)+a}{p_n^m(1-a)} \geq 1 \quad (30)$$

Considering (30) and the fact that e^{-x} is a decreasing function (for $x > 0$), we obtain:

$$\exp\left\{-\frac{x}{a} q_i(1-a) \frac{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)+a\}}{\prod_{\substack{e_{(m,n)}\in s_i \\ e_{(m,n)}\in s_j}} \{p_n^m(1-a)\}}\right\} \quad (31)$$
$$\leq \exp\{-\frac{x}{a} q_i(1-a)\}$$

From (30) and inequality (31), we obtain (32):

$$U\Theta_i[x,\mathbf{q}] - \Theta_i[x,\mathbf{q}] \leq q_i d_i \exp\left\{-\frac{x}{a}(q_i + a(1-q_i))\right\}$$
$$+ \sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j \exp\left\{-\frac{x}{a} q_i(1-a)\right\} - \Theta_i[x,\mathbf{q}] \sum_{j=1}^{r} q_j d_j \quad (32)$$

However,

$$\Theta_i[x,\mathbf{q}] \sum_{j=1}^{r} q_j d_j = q_i d_i \Theta_i[x,\mathbf{q}] + \Theta_i[x,\mathbf{q}] \sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j$$

and hence

$$U\Theta_i[x,\mathbf{q}] - \Theta_i[x,\mathbf{q}]$$
$$\leq_i d_i \left\{\exp\left\{-\frac{x}{a}(q_i + a(1-q_i))\right\} - \Theta_i[x,\mathbf{q}]\right\}$$
$$+ \sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j \left\{\exp\left\{-\frac{x}{a} q_i(1-a)\right\} - \Theta_i[x,\mathbf{q}]\right\}$$
$$= \Theta_i[x,\mathbf{q}] \{q_i d_i \{\exp\{-x(1-q_i)\} - 1\}\} \quad (33)$$
$$+ \Theta_i[x,\mathbf{q}] \left\{\sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j \{\exp\{x q_i\} - 1\}\right\}$$
$$= -x q_i \Theta_i[x,\mathbf{q}] \left\{\begin{array}{l} d_i(1-q_i)\frac{\exp\{-x(1-q_i)\}-1}{-x(1-q_i)} \\ -\sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j \frac{\exp\{x q_i\}-1}{x q_i} \end{array}\right\} \quad (34)$$

We define

$$V[x] = \begin{cases} \dfrac{\mathrm{e}^x - 1}{x} & x \neq 0 \\ 1 & x = 0 \end{cases} \quad (35)$$

Substituting $V[x]$ into (33), we have:

$$U\Theta_i[x,\mathbf{q}] - \Theta_i[x,\mathbf{q}]$$
$$\leq -x q_i \Theta_i[x,\mathbf{q}] \left\{\begin{array}{l} d_i(1-q_i)V[-x(1-q_i)] \\ -\sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j V[x q_i] \end{array}\right\} \quad (36)$$

From (36), we conclude that $\Theta_i[x,\mathbf{q}]$ is super-regular if and only if

$$\left\{d_i(1-q_i)V[-x(1-q_i)] - \sum_{\substack{j=1 \\ j\neq i}}^{r} q_j d_j V[x q_i]\right\} \geq 0 \quad (37)$$

which implies that

$$\frac{V[-x(1-q_i)]}{V[xq_i]} \geq \frac{\sum_{\substack{j=1\\ j\neq i}}^{r} q_j d_j}{(1-q_i)d_i} = \frac{\sum_{\substack{j=1\\ j\neq i}}^{r} q_j d_j}{\sum_{\substack{j=1\\ j\neq i}}^{r} q_j d_i} = \frac{\sum_{\substack{j=1\\ j\neq i}}^{r} q_j \frac{d_j}{d_i}}{\sum_{\substack{j=1\\ j\neq i}}^{r} q_j} \tag{38}$$

However,

$$\min_{j\neq i} \frac{d_j}{d_i} \leq \frac{\sum_{\substack{j=1\\ j\neq i}}^{r} q_j \frac{d_j}{d_i}}{\sum_{\substack{j=1\\ j\neq i}}^{r} q_j} \leq \max_{j\neq i} \frac{d_j}{d_i} \tag{39}$$

It follows that $\Theta_i[x, \mathbf{q}]$ is super-regular if

$$\frac{V[-x(1-q_i)]}{V[xq_i]} \geq \max_{j\neq i} \frac{d_j}{d_i} \tag{40}$$

It can be shown that [20]

$$\frac{1}{V[x]} \leq \frac{V[-x(1-q_i)]}{V[xq_i]} \leq V[-x] \tag{41}$$

From (40) and (41) we conclude that $\Theta_i[x, \mathbf{q}]$ is super-regular if:

$$\max_{j\neq i} \frac{d_j}{d_i} \leq \frac{1}{V[x]} \leq 1 \tag{42}$$

Since $\frac{1}{V[x]}$ is a continuous and strictly decreasing function, there exists a value $x = x^*$ such that, if $d_i > d_j (\forall j \neq i)$, then $1/V[x] = \max_{j\neq i} \frac{d_j}{d_i}$. For all $x \in (0, x^*]$, the inequality $\frac{1}{V[x]} \geq \max_{j\neq i} \frac{d_j}{d_i}$ holds, from which (37) is true and consequently $\Theta_i[x, \mathbf{q}]$ is a super-regular function. This completes the proof. □

$\Phi_i[x, \mathbf{q}]$ is a sub-regular function satisfying the boundary conditions given in (18) and $\Phi_i[x, \mathbf{q}] \leq \Gamma_i[\mathbf{q}] \leq 1$. By considering the definition of $\Phi_i[x, \mathbf{q}]$ in (21), we see that, for any given $\epsilon > 0$, there exists a positive constant $a^* < 1$ for the learning rate of the eDLA so that the inequality $1 - \epsilon \leq \Phi_i[x, \mathbf{q}] \leq \Gamma_i[\mathbf{q}] = \Pr[\mathbf{q}^* = \mathbf{e_i}|\mathbf{q}(0) = \mathbf{q}] \leq 1$ holds for all positive $a < a^*$.

This completes the proof of Theorem 1 that

$$\lim_{n\to\infty} q_i(n) \overset{a.s.}{=} 1$$

We have shown that there exists x^* such that for all $x < x^*$, $\Phi_i[x, \mathbf{q}]$ is sub-regular and

$$1 - \epsilon \leq \Phi_i[x, \mathbf{q}] \leq \Pr[\mathbf{q}^* = \mathbf{e_i}|\mathbf{q}(0) = \mathbf{q}] \leq 1$$

The quantity $1 - \epsilon$ denotes the confidence level for $\Pr(|w^* - w_i| \leq \delta)$, and x^* satisfies $\frac{1}{V[x^*]} = \max_{j\neq i} \left(\frac{d_j}{d_i}\right)$. Accordingly, the learning rate of the proposed algorithm can be determined for each error parameter $\epsilon \in (0, 1)$ as follows.

1. The parameter $x = ay$ can be obtained from (43):

$$\Phi_i[x, \mathbf{q}] = \frac{1 - e^{-q_i y}}{1 - e^{-y}} = 1 - \epsilon \tag{43}$$

After simplification, (43) can be rewritten as

$$(e^{-y})^{q_i} + e^{-y}(1 - \epsilon) - \epsilon = 0 \tag{44}$$

which can be solved by numerical methods.

2. For a specific parameter x, the eDLA learning rate, a, is calculated by solving (45):

$$\frac{1}{V[y]} = \max_{j\neq i} \left(\frac{d_j}{d_i}\right) = \frac{ax}{e^{ax} - 1} \tag{45}$$

In the rest of this section, the efficiency of the proposed eDLA-based algorithm and of the standard sampling method are compared.

We know from sampling theory that, to obtain an optimal subgraph in a stochastic graph with certain confidence level, each edge of this subgraph must be sampled a certain number of times [24, 25].

Lemma 4 *Suppose that in the proposed eDLA-based algorithm we wish subgraph s_i to be sampled n times. Then the proposed algorithm must be run n' times, and*

$$\mathbb{E}[n'] = \sum_{t=1}^{n} \mathbb{E}\left[\frac{1}{q_i(t)}\right]$$

where $q_i(t)$ represents the probability of choosing s_i at iteration t.

Proof Assume that the probability of selecting subgraph s_i at step t is $q_i(t)$. To obtain exactly one sample of s_i, the algorithm must be run $n_i(t)$ times, and we have

$$\mathbb{E}[n_i(t)|\mathbf{q}(t)] = \sum_{h=1}^{\infty} h q_i(t)(1 - q_i(t))^{h-1} = \frac{1}{q_i(t)} \tag{46}$$

Taking the expectation on both sides of (46), we have

$$\mathbb{E}[\mathbb{E}[n_i(t)|\mathbf{q}(t)]] = \mathbb{E}[n_i(t)] = \mathbb{E}\left[\frac{1}{q_i(t)}\right] \tag{47}$$

and finally

$$n' = \sum_{t=1}^{n} \mathbb{E}[n_i(t)] = \sum_{t=1}^{n} \mathbb{E}\left[\frac{1}{q_i(t)}\right] \tag{48}$$

This completes the proof. □

Remark 1 From Jensen's inequality, we know that if X is a random variable and φ is a convex function, then $\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(X)]$. Since $\frac{1}{x}$ is a convex function, we have $\mathbb{E}[n_i(t)] = \mathbb{E}\left[{}^1/_{q_i(t)}\right] \geq {}^1/_{\mathbb{E}[q_i(t)]}$.

From Lemma 2 we know that for the optimal subgraph s_i, the inequality $\mathbb{E}[q_i(t+1)] \geq \mathbb{E}[q_i(t)]$ is true and therefore we conclude that $\mathbb{E}[n_i(t+1)] \leq \mathbb{E}[n_i(t)]$.

Let $\xi(i,n) = \frac{1}{n}\sum_{k=1}^{n}\mathbb{E}[n_i(k)]$. In the next lemma, we show the decreasing property of $\xi(i,n)$ for the optimal subgraph s_i.

Lemma 5 *Suppose that s_i is the optimal subgraph. For any $n_0, n_1 \in N : n_0 \leq n_1$, we have*

$$\xi(i,n_1) \leq \xi(i,n_0)$$

Proof We show that, for any $n \in N$, we have $\xi(i,n+1) \leq \xi(i,n)$. From the definition of $\xi(i,n)$, we have

$$\begin{aligned}\xi(i,n+1) &= \frac{\sum_{k=1}^{n+1}\mathbb{E}[n_i(k)]}{n+1} = \frac{n\xi(i,n)+\mathbb{E}[n_i(n+1)]}{n+1} \\ &= \xi(i,n)+\frac{1}{n+1}\left(\mathbb{E}[n_i(n+1)]-\xi(i,n)\right) \quad (49)\end{aligned}$$

However, from the previous lemma we know that for the optimal subgraph s_i, the inequality $\mathbb{E}[n_i(t+1)] \leq \mathbb{E}[n_i(t)]$ holds and thus

$$\begin{aligned}\xi(i,n) &= \frac{1}{n}\sum_{k=1}^{n}\mathbb{E}[n_i(k)] \geq \frac{\sum_{k=1}^{n}\mathbb{E}[n_i(n+1)]}{n} \quad (50)\\ &= \mathbb{E}\left[\mathbb{E}[n_i(n+1)]\right]\end{aligned}$$

From the two inequalities (49) and (50), we conclude that

$$\xi(i,n+1) \leq \xi(i,n)$$

This completes the proof. □

Theorem 2 *Suppose that we wish to construct a $100(1-\alpha)\%$ confidence interval for the weight of an optimal subgraph. For sufficiently small values of α, the total number of sampled edges in the standard sampling method is greater than that in the proposed eDLA-based algorithm.*

Proof Suppose that the standard sampling method has to sample each edge n times to guarantee the specified confidence level for the weight of the optimal subgraph. In addition, suppose that $\tau(G)$ denotes the total number of subgraphs. Then the standard sampling method has to run $n \times \tau(G)$ times and $n \times \tau(G) \times (|V|-1)$ edges have to be sampled. To guarantee the same confidence level, the proposed eDLA-based algorithm has to sample at least n times from the optimal subgraph. From lemma 4, we know that to obtain n samples from s_i, the proposed algorithm has to run $n \times \xi(i,n)$ times. This implies that $n \times \xi(i,n) \times (|V|-1)$ edges have to be sampled.

Assume that η denotes the ratio of the total number of sampled edges in the standard sampling method (n_{standard}) to that in the proposed eDLA-based algorithm (n_{eDLA}). Then we have

$$\eta = \frac{n_{\text{standard}}}{n_{\text{eDLA}}} = \frac{n \times \tau(G) \times (|V|-1)}{n \times \xi(i,n) \times (|V|-1)} = \frac{\tau(G)}{\xi(i,n)}$$

Since $\xi(i,n)$ is a decreasing function, there exists a value k such that, for all $n > k$,

$$\xi(i,n) \leq \tau(G)$$

This completes the proof that for high confidence levels, $\eta > 1$ or $n_{\text{standard}} > n_{\text{eDLA}}$. □

5 Experiments

To study the feasibility of the proposed algorithm, two sets of experiments are conducted on some stochastic graphs. The stochastic shortest path problem and stochastic minimum spanning tree problem are denoted by SSPP and SMSTP, respectively. The former is solved by the eDLA-based algorithm on the directed stochastic graph, Graph2, taken from [7]. The latter is solved by the eDLA-based algorithm on the undirected stochastic graph Alex1-a taken from [26].

In [7], the authors proposed a DLA-based algorithm to solve the SSPP. At each stage in this method, the source automaton (corresponding to the source node) selects one of its actions (as a sample realization of its action probability vector). The selected action activates another LA. The process of choosing an action and activating an automaton is repeated until the destination automaton is reached. The weight of a constructed path is used to reward or penalize the actions selected by each LA in the DLA.

In [27], the authors proposed an algorithm based on a colony of LAs to solve SMSTP. At each stage in this algorithm, a node in the graph is randomly selected and the associated LA selects an action corresponding to an edge of this node that prevents loop construction. Consequently, at the end of each stage the selected edges form a tree. The weight of the constructed tree is used to reward or penalize the action chosen by the LA.

It has been shown that the proposed algorithm is more efficient than other existing methods for finding the minimum spanning tree in stochastic graphs.

In this section, we use an eDLA to solve these problems. At each stage, an LA is selected as the root. In the SSPP case, the root is the source automaton and in SMSTP each node can be selected randomly as the root. The activity level of the root is set to *Ac*.

Each automaton with the *Fi* level of activity selects one of its available actions based on the selection action probability vector. In SMSTP, the next *Fi* automaton is selected

Table 1 Differences between SSPP and SMSTP solving mechanisms in the eDLA-based algorithm

	Problem	
	SSP between s and d in $G = (V, E)$	**SMST** of $G = (V, E)$
Selection of ROOT	Deterministic: s (source)	Stochastic: uniformly random from V
Selection of FIRE	Deterministic: by current selected action	Stochastic: uniformly random from A_A
Termination condition	$A_F = \{d\}$	$A_O = V$

randomly from the set of automata with the *Ac* level of activity. In SSPP, the current action determines a specific LA in the set of automata with the *Ac* level of activity that should be upgraded to the *Fi* level of activity. This process (fire and action) continues until the destination node is reached (the destination automaton downgrades to the *Of* level of activity) or a tree is constructed (all of the LAs downgrade to the *Of* level of activity) in SSPP and SMSTP respectively.

Algorithm 2 eDLA-Based SSP Algorithm

Input: Stochastic Graph $G = (V, E)$, $s, d \in V$, P_p; K_p;
Output: Shortest path P between s and d;

Assign a LA to each node of G and make an eDLA=(V,E)
Let p denotes the constructed path between s and d;
Let W_p denotes the weight of constructed path;
$K \leftarrow 0$; $WTH \leftarrow 0$;$Var \leftarrow 0$;$Err \leftarrow 0$;
repeat
  $p \leftarrow \phi$; $W_p \leftarrow 0$;
  $A_P \leftarrow V - \{s\}$; ▷ A_P = set of LAs with *Passive* level
  $A_A \leftarrow s$; ▷ A_P = set of LAs with *Active* level
  $A_O \leftarrow \phi$; ▷ A_O = set of LAs with *Off* level
  $A_F \leftarrow$ `Fire`(eDLA); ▷ A_F has *Fi* level
  while $A_F \neq d$ **do**
    $a \leftarrow$ `SelectAction`(A_F);
    disable action a in adjacent active LA;
    $s \leftarrow s \cup \text{edge}(A_F, a)$;
    $W_p \leftarrow W_p +$ `SampledWeight`$(\text{edge}(A_F, a))$;
    $A_O \leftarrow A_O \cup A_F$;
    $A_F \leftarrow$ `Fire`(eDLA);
  end while
  $Err \leftarrow W_p - WTH$;
  $WTH \leftarrow WTH + 0.1 \times Err$;
  $Var \leftarrow Var + 0.2 \times (\mid Err \mid - Var)$;
  if $W_p \leq \frac{WTH}{2} + 2 \times Var$ **then**
    response=REWARD;
  end if
  $K \leftarrow K + 1$;
  for all $LA \in A_O$ **do**
    if response = REWARD **then**
      `reward` selected action by LA; ▷ L_{R-I}
    end if
    enable all disabled actions; ▷ for next run
  end for
  $q = \prod P_{\alpha_n}^{A_m} \quad \forall \text{edge}(A_m, a_n) \in p$;
until $(K \geq K_p \vee q \geq P_p)$

The process of finding a path or a spanning tree is repeated until the stopping criterion is reached, at which point the last sampled path or spanning tree has the minimum expected weight. The algorithm stops if the product of the probabilities of selecting an edge of the sampled subgraph is greater than a predefined threshold, or if the number of sampled subgraphs reaches a certain number. The modified algorithms for SSPP and SMSTP are presented as Algorithm 2 and Algorithm 3 respectively. The differences between the mechanisms used by the eDLAs to solve each of the above problems are described in Table 1.

Algorithm 3 eDLA-Based SMST Algorithm

Input: Stochastic Graph $G = (V, E)$, P_s; K_s;
Output: Spanning Tree s;

Assign a LA to each node of G and make an eDLA=(V,E)
Let s denotes the constructed spanning tree;
Let W_s denotes the weight of constructed spanning tree;
$K \leftarrow 0$; $WTH \leftarrow 0$;$Var \leftarrow 0$;$Err \leftarrow 0$;
repeat
  $s \leftarrow \phi$; $W_s \leftarrow 0$;
  Select one of the automaton at random and call it v;
  $A_P \leftarrow V - \{v\}$; ▷ A_P = set of LAs with *Pa* level
  $A_A \leftarrow v$; ▷ A_P = set of LAs with *Ac* level
  $A_O \leftarrow \phi$; ▷ A_O = set of LAs with *Of* level
  $A_F \leftarrow$ `Fire`(eDLA); ▷ A_F has *Fi* level
  while $A_F \neq \phi$ **do**
    $a \leftarrow$ `SelectAction`(A_F);
    disable action a in adjacent active LA;
    $s \leftarrow s \cup \text{edge}(A_F, a)$;
    $W_s \leftarrow W_s +$ `SampledWeight`$(\text{edge}(A_F, a))$;
    $A_O \leftarrow A_O \cup A_F$;
    $A_F \leftarrow$ `Fire`(eDLA);
  end while
  $Err \leftarrow W_s - WTH$;
  $WTH \leftarrow WTH + 0.1 \times Err$;
  $Var \leftarrow Var + 0.2 \times (\mid Err \mid - Var)$;
  if $W_s \leq \frac{WTH}{2} + 2 \times Var$ **then**
    response=REWARD;
  end if
  $K \leftarrow K + 1$;
  for all $LA \in A_O$ **do**
    if response = REWARD **then**
      `reward` selected action by LA; ▷ L_{R-I}
    end if
    enable all disabled actions; ▷ for next run
  end for
  $q = \prod P_{\alpha_n}^{A_m} \quad \forall \text{edge}(A_m, a_n) \in s$;
until $(K \geq K_s \vee q \geq P_s)$

Graph2 is a directed graph with 15 nodes, 42 edges, $v_s = 1$, $v_d = 15$ and optimal path $\pi^* = (1, 4, 12, 14, 15)$. The edge cost distribution is given in Table 2.

Table 2 Weight Distribution of Graph2

Edge	Weights				Probabilities			
(1,2)	19	25	36		0.6	0.3	0.1	
(1,3)	21	24	25	29	0.5	0.2	0.2	0.1
(1,4)	11	13	16		0.4	0.4	0.2	
(2,11)	24	28	31		0.5	0.3	0.2	
(2,5)	21	30			0.7	0.3		
(3,6)	18	24			0.7	0.3		
(2,6)	13	37	39		0.6	0.2	0.2	
(2,3)	11	20	24		0.6	0.3	0.1	
(3,7)	23	30	34		0.4	0.3	0.3	
(3,8)	14	23	34		0.5	0.4	0.1	
(3,4)	22	30			0.7	0.3		
(4,9)	35	40			0.6	0.4		
(4,12)	16	19	37		0.5	0.4	0.1	
(5,13)	28	35	37	40	0.4	0.3	0.2	0.1
(5,15)	29	32			0.7	0.3		
(5,6)	18	25	29		0.5	0.3	0.2	
(5,10)	27	33	40		0.4	0.3	0.3	
(5,7)	15	17	19	26	0.3	0.3	0.3	0.1
(6,13)	21	23			0.5	0.5		
(6,7)	12	23	31		0.5	0.3	0.2	
(7,10)	19	23	37		0.6	0.2	0.2	
(7,8)	12	15	22	24	0.3	0.3	0.3	0.1
(8,7)	14	34	39		0.6	0.2	0.2	
(7,6)	12	23	31		0.5	0.3	0.2	
(8,14)	14	15	27	32	0.3	0.3	0.2	0.2
(8,9)	13	31	32		0.8	0.1	0.1	
(4,8)	13	23	34		0.4	0.3	0.3	
(7,9)	10	17	20		0.6	0.3	0.1	
(9,10)	16	18	36	39	0.3	0.3	0.2	0.2
(9,15)	12	13	25	32	0.4	0.3	0.2	0.1
(9,14)	19	24	29		0.4	0.3	0.3	
(10,13)	14	20	25	32	0.3	0.3	0.2	0.2
(10,15)	15	19	25		0.4	0.3	0.3	
(10,14)	23	34			0.9	0.1		
(11,13)	13	31	25		0.6	0.3	0.1	
(5,11)	18	19	20	23	0.3	0.3	0.3	0.1
(6,11)	10	19	39		0.5	0.4	0.1	
(8,12)	15	36	39		0.5	0.3	0.2	
(9,12)	16	22			0.7	0.3		
(12,14)	10	13	18	34	0.3	0.3	0.3	0.1
(13,15)	12	31			0.9	0.1		
(14,15)	14	16	32		0.5	0.3	0.2	

Alex1-a is an undirected graph with 8 nodes, 14 edges and optimal spanning tree

$$\tau^* = (1, 3)(2, 3)(2, 5)(3, 4)(5, 6)(6, 7)(7, 8).$$

The edge weight distribution is given in Table 3.

In all the simulations that are presented in the rest of this paper, an L_{R-I} scheme is used as the learning algorithm for updating the action selection probability vectors.

In the SSPP, each algorithm (the proposed eDLA-based algorithm and the DLA-based algorithm [7]) is tested on Graph2.

Each algorithm is terminated when the number of sampled paths is greater than 10000 or the probability of the selected path is greater than 90 %. The results for each algorithm are summarized in Table 4. Every value in this table is averaged over 50 runs. To compare the results, four performance measures are calculated: the average number of samples that should be taken from all edges in the graph (AS), the average number of iterations for a convergent run (AI), the average required time for a convergent run (AT) and percentage of convergent runs (PC).

A similar method is applied to solve the SMSTP. The results for each algorithm (the proposed eDLA-based algorithm and the LA-based algorithm [27]) are summarized in Table 5.

From the tables of results, the following points can be made:

1. The proposed network structure of LAs is flexible enough to be able to solve different network optimization problems. Furthermore, since most combinatorial optimization problems can be formulated as optimization problems on a weighted graph, the proposed eDLA

Table 3 Weight Distribution of Alex1-a

Edge	Weights		Probabilities	
(1,2)	70	94	0.95	0.05
(1,3)	25	52	0.80	0.20
(1,4)	42	61	0.70	0.30
(2,3)	15	43	0.80	0.20
(2,5)	26	50	0.85	0.15
(3,4)	21	68	0.90	0.10
(3,6)	65	75	0.60	0.40
(3,7)	89	78	0.70	0.30
(4,7)	90	96	0.95	0.05
(4,8)	89	96	0.85	0.15
(5,6)	32	67	0.80	0.20
(6,7)	16	42	0.70	0.30
(7,8)	3	15	0.60	0.40
(6,8)	98	45	0.80	0.20

Table 4 Simulation results of SSPP: best result for each algorithm is indicated by bold entries

	DLA-based [7]				Proposed eDLA-based Algorithm 2			
α	AS	AI	AT	PC	AS	AI	AT	PC
0.003	16825	3918	0.052	100 %	20863	4869	0.058	100 %
0.004	13244	3094	0.039	100 %	15553	3628	0.059	100 %
0.005	10125	2357	0.041	100 %	12422	2900	0.062	100 %
0.006	9377	2131	0.062	100 %	10287	2404	0.039	100 %
0.007	**7548**	**1769**	**0.045**	**100 %**	9050	2114	0.041	100 %
0.008	6881	1614	0.036	98 %	7892	1844	0.055	100 %
0.009	5878	1375	0.033	98 %	7052	1648	0.050	100 %
0.01	5485	1287	0.043	98 %	5968	1391	0.052	100 %
0.02	3309	796	0.047	90 %	3193	747	0.043	100 %
0.03	2127	504	0.054	86 %	2250	531	0.053	100 %
0.04	1993	475	0.070	76 %	1755	415	0.038	100 %
0.05	1593	387	0.047	76 %	**1354**	**319**	**0.038**	**100 %**
0.06	1338	320	0.061	68 %	1339	316	0.050	98 %
0.07	1394	328	0.120	58 %	953	222	0.038	100 %
0.08	1132	269	0.077	60 %	1060	257	0.068	100 %
0.09	758	180	0.062	66 %	1098	267	0.063	96 %

structure will also be able to solve combinatorial optimization problems.

2. The proposed variance-aware threshold improves the rate of convergence. This is due to including the variance of the obtained solutions in computing the threshold value. The proposed threshold value is a more realistic criterion for comparison than the dynamic threshold value that has been proposed in previous works [7].

   It seems that this new threshold is useful to prevent the eDLA-based algorithm from becoming stuck in local optima. The value of the thresholds in the proposed method and the one used by [7] are compared with the average path weight of the shortest path in Graph2 (Fig. 5).
3. The results show that the proposed algorithm can be used as a unified framework for solving a wide range of stochastic network optimization problems through sampling.
4. Comparing the percentage of convergence (PC column) in Tables 4 and 5 shows that the proposed variance-aware threshold value may improve the convergence rate of the algorithm. This improvement is such that convergence is almost always guaranteed, and is approximately independent of the learning rate.

Table 5 Simulation results for the SMSTP: best result for each algorithm is indicated by bold entries

	LA-based Algorithm [27]			eDLA-based Algorithm3		
α	AS	AI	PC	AS	AI	PC
0.007	79430	11348	100 %	96268	13753	100 %
0.008	78068	10393	100 %	84479	12069	100 %
0.009	61280	8755	100 %	75895	10843	100 %
0.01	**67330**	**7747**	**100 %**	65389	9342	100 %
0.02	65268	4064	88 %	31484	4498	100 %
0.03	98431	3456	68 %	18770	2682	100 %
0.04	128830	2783	60 %	13650	1950	100 %
0.05	139707	2457	50 %	11816	1688	100 %
0.06	191879	2103	36 %	9865	1409	100 %
0.07	170472	4058	16 %	**7101**	**1015**	**100 %**

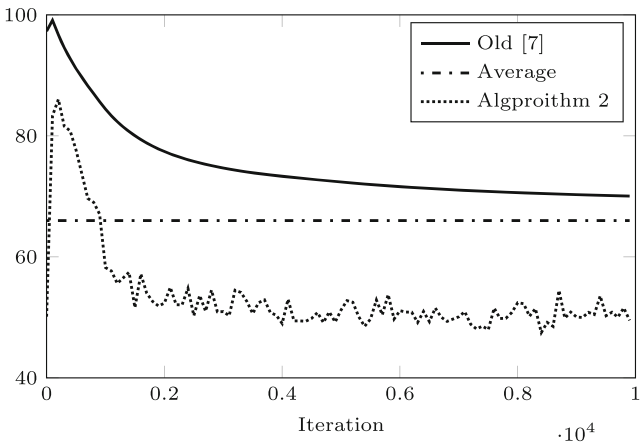


Fig. 5 Difference between the values of thresholds compared with the average path weight in Graph2

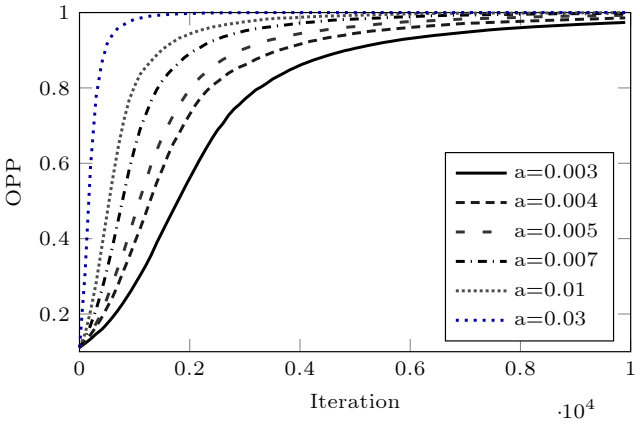


Fig. 6 Probability of the optimal path for Graph2 in the proposed eDLA-based algorithm for different learning rates

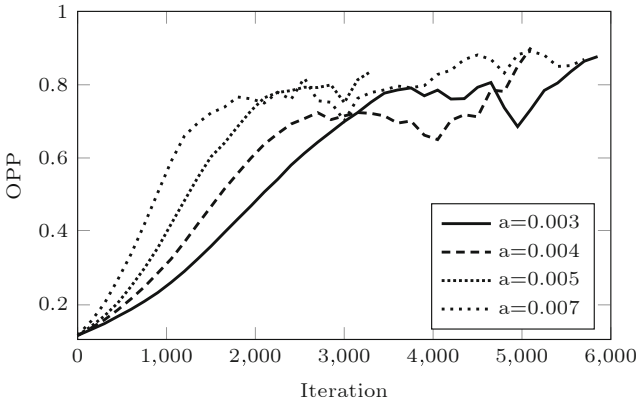


Fig. 8 Probability of the optimal path in Graph2 in the DLA-based algorithm for different learning rates

5. To compare the algorithms, the minimum number of samples and iterations required to guarantee 100 % convergence is considered. These cases are indicated by bold letters in Tables 4 and 5. In the case of SSPP, the results in Table 4 show that the proposed algorithm reduces the average number of required samples and the average number of iterations to $\frac{1}{5}$ (80 % improvement). The same conclusion is also true in the case of the SMSTP results.
6. The convergent behavior is illustrated by the any-time curve of the algorithm. This property is shown by plotting the probability of the optimal subgraph over time. To do this, the probability of the optimal subgraph at each step of the algorithm is defined as the product of the probabilities of automata actions that correspond to the edges in the optimal subgraph. The term any-time refers to the property that the search can be stopped at any time and the proposed algorithm will yield sub-optimal solutions. A point with coordinates (x, y) on the any-time curve gives us the average number of iterations x required to obtain the optimal solution with probability y. The any-time plots show the correctness of Theorem 1. Furthermore, they show some other features of the proposed algorithm.

   In another experiment, the proposed algorithm is used to find the optimal shortest path in Graph2. The algorithm terminates when the number of sampled paths is greater than or equal to 10000. The value of the optimal path probability at each iteration is computed and averaged over 10 simulations. The results are shown in Fig. 6 for different values of the learning rate (a). The different curves in Fig. 6 indicate that the average number of iterations required by the proposed algorithm for finding the optimal solution is dependent on the learning rate.
7. The any-time curves for the proposed algorithm and for the DLA-based algorithm [7] for SSPP are shown in Fig. 7 and Fig. 8 respectively. For each individual learning rate, the slope of the POP in the proposed algorithm is greater than that for the DLA-based algorithm. Furthermore, the any-time curve for the new algorithm

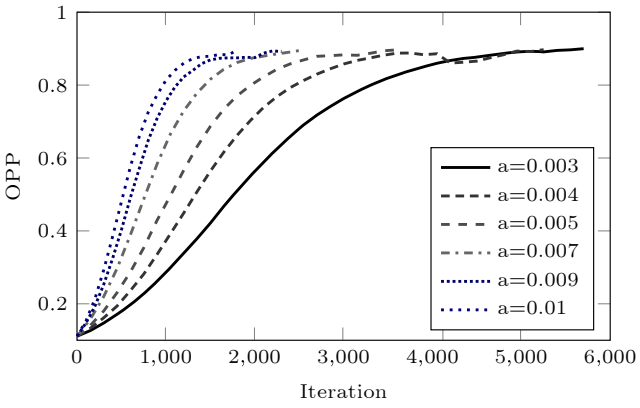


Fig. 7 Probability of the optimal path for Graph2 in the eDLA-based algorithm for different learning rates

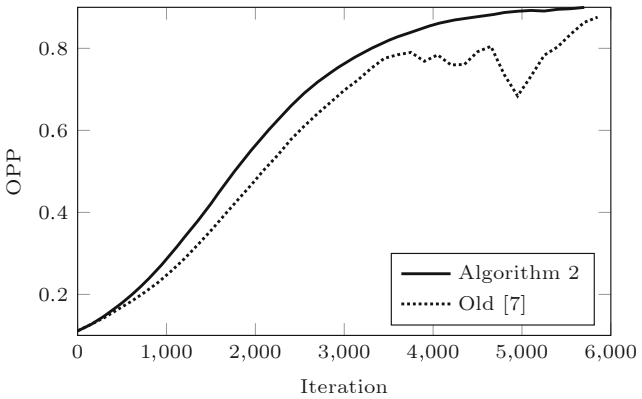


Fig. 9 POP curves for the proposed algorithm and for the old DLA-based algorithm for SSPP in Graph2 (learning rate = 0.003)

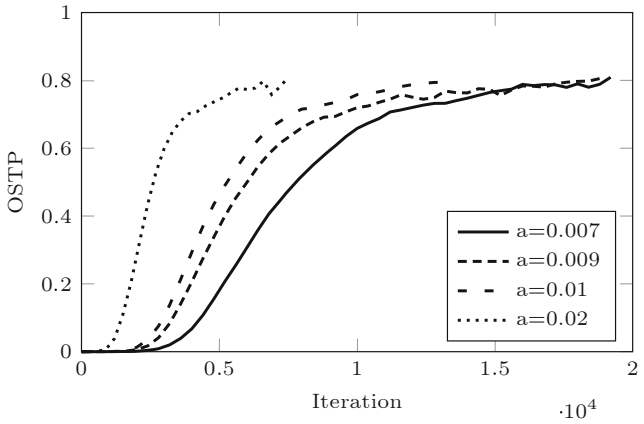


Fig. 10 Probability of the optimal spanning tree in Alex1-a in the eDLA-based algorithm for different learning rates

is more stable than that for the DLA-based algorithm (Fig. 9)

8. The any-time curve for the proposed algorithm for SMSTP in Alex1-a is shown in Fig. 10. As indicated in this figure, the speed of convergence to the optimal path is independent of the learning rate. With a low learning rate, the algorithm converges slowly but steadily to the optimal solution. High learning rates lead to rapid convergence to the optimal solution, but the algorithm may converge to a sub-optimal solution. In Fig. 11 the probability of the optimal spanning tree in the proposed eDLA-based algorithm and the old LA-based algorithm are compared. The slope of the curve for the new method is greater than that for the old method. Besides, the new method is more stable than the old one. As a result, at each stage, a sampled tree from the new method is more likely to be the optimal spanning tree than one from the old method.

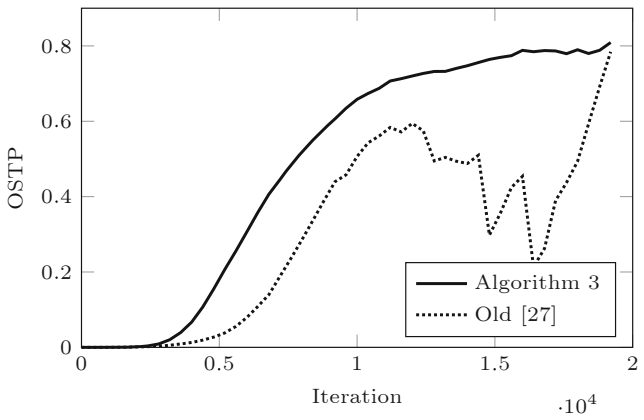


Fig. 11 Probability of the optimal spanning tree in Alex1-a in the proposed algorithm and in the old LA-based algorithm for SMSTP (learning rate = 0.07)

6 Conclusion

In this paper, eDLAs are introduced as a new method for the cooperation of LAs. In this network, each LA has an activity level that changes over time, according to the underlying problem and the set of communication rules. At each step of the algorithm, only one LA has the highest level called *Fire*.

A new adaptive procedure based on eDLAs is introduced to solve optimization problems in stochastic graphs. The proposed algorithm provides a method that can be used to determine a set of edges to be sampled, and consequently specifies a subgraph with optimal expected weight. The convergence of the proposed algorithm is proven.

Two sets of experiments are performed on the SSP and SMST problems. The results indicate the superiority of the proposed eDLA-based algorithm compared with previous automata-based algorithms. Furthermore, a new variance-aware method for computing the threshold value is introduced, which is useful for preventing the proposed algorithm from becoming stuck in local optima and improves the convergence of the algorithm significantly.

References

1. Narendra K. S., Thathachar M. L. A. A. (1974) Learning automata-a survey. IEEE Trans on Syst, Man Cybern (4),323–334
2. Thathachar ML, Sastry PS (2002) Varieties of learning automata: An overview. IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics: A publication of the IEEE Systems, Man, and cybernetics society 32:711–22
3. Meybodi M, Beigy H (2001) A sampling method based on distributed learning automata for stochastic shortest path problem. In: the 10th Iranian conference on electrical engineering, vol I
4. Jahanshahi M, Dehghan M, Meybodi MR (2012) LAMR: learning automata based multicast routing protocol for multi-channel multi-radio wireless mesh networks. Appl Intell 38:58–77
5. Yazidi A, Granmo O-C, Oommen BJ (2011) Service selection in stochastic environments: A learning-automaton based solution. Appl Intell 36:617–637
6. Beigy H, Meybodi M (2003) Solving stochastic shoretst path problem using Monte Carlo sampling method: a distributed learning automata approach. Springer-Verlag lecture notes in advances in soft computing: Neural networks and soft computing, pp 626–632
7. Beigy H, Meybodi MR (2006) Utilizing distributed learning automata to solve stochastic shortest path problems. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 14:591–615
8. Mollakhalili Meybodi M, Meybodi MR (2004) A New Distributed Learning Automata Based Algorithm for Solving Stochastic Shortest Path. In: 6th conference on intelligent systems, (kerman)
9. Alipour M, Meybodi M (2005) Solving traveling salesman problem using distributed learning automata. In: Proceedings of 10th Annual CSI Computer Conference, (Tehran, Iran), pp 759–761

10. Motevalian A, Meybodi M (2006) Solving maximal independent set problem using distributed learning automata. In: 14th Iranian electrical engineering conference (ICEE2006), vol 1. Tehran
11. Saati S, Meybodi M (2005) A self organizing model for document structure using distributed learning automata. In: Second international conference on information and knowledge technology (IKT2005)
12. Anari B, Meybodi MR (2007) A Method based on distributed learning automata for determining web documents structure. In: 12th annual CSI computer conference of Iran, pp 2276–2282
13. Mollakhalili Meybodi M, Meybodi M (2008) Link prediction in adaptive web sites using distributed learning automata. In: 13th annual CSI computer conference of Iran, Kish Island
14. Mollakhalili Meybodi M, Meybodi MR (2012) A distributed learning automata based approach for user modeling in adaptive hypermedia. In: Congress on electrical, computer and information technology, Mashhad
15. BaradaranHashemi A, Meybodi M (2007) Web Usage Mining Using Distributed Learning Automata, In: 12th Annual CSI Computer Conference of Iran, Tehran, pp 553–560
16. Akbari Torkestani J (2012) An adaptive focused web crawling algorithm based on learning automata. Appl Intell 37:586–601
17. Thathachar B, Harita MAL (1987) Learning automata with changing number of actions. IEEE Trans Syst Man Cybern Syst Hum 17(6):1095–1100
18. Sato T (1999) On some asymptotic properties of learning automaton networks. Techical Report
19. Lakshmivarahan S, Thathachar M (1976) Bounds on the convergence probabilities of learning automata, IEEE Trans Syst Man Cybern Syst Hum 6(11):756–763
20. Norman F (1968) On the linear model with two absorbing. J Math Psychol 5:225–241
21. Zhang X, Granmo O-C, Oommen BJ, Jiao L (2014) A formal proof of the ϵ-optimality of absorbing continuous pursuit algorithms using the theory of regular functions. In: Applied intelligence
22. Akbari Torkestani J (2012) An adaptive focused Web crawling algorithm based on learning automata. Appl Intell 37:586–601
23. Narendra, Kumpati S., M. L. A. A. Thathachar (1989) Learning automata: An introduction. Prentice Hall, Englewood Cliffs
24. Ross SM (2004) Introduction to probability and statistics dor engineers and scientists, 3rd edn. Elsevier Academic Press
25. Papoulis A (1991) Probability, random variables, and stochastic processes, 3rd edn., McGrawHill
26. Hutson KR, Shier DR (2005) Bounding distributions for the weight of a minimum spanning tree in stochastic networks. Oper Res 53(5):879–886
27. Akbari Torkestani J, Meybodi MR (2011) Learning automata-based algorithms for solving stochastic minimum spanning tree problem. Appl Soft Comput 11:4064–4077

Mohammad Reza Mollakhalili Meybodi received his M.Sc. degree in Computer Engineering from Amirkabir University of Technology, Tehran, Iran, in 2004, and the B.Sc. degree in Computer Engineering from Shahid Beheshti University in 2001. He joined the faculty of Computer engineering Department at Islamic Azad University (Maybod Branch) in 2004. His research interests include Communication Networks, Learning Systems, Randomized Algorithms, Graph Theory, Stochastic Optimization, and Soft Computing.

Mohammad Reza Meybodi received M.S. and Ph.D. degrees in Computer Science from Oklahoma University, USA, in 1980 and 1983, respectively. Currently, he is Full Professor in the Computer Engineering Department of Amirkabir University of Technology, in Tehran, Iran. Prior to his current position, he worked from 1983 to 1985 as Assistant Professor at Western Michigan University, and from 1985 to 1991 as Associate Professor at Ohio University, USA. His research interests include Learning Systems, Parallel Algorithms, Soft Computing and Software Development.